JUAN DAVID CHIMARRO AMAGUAÑA
RAMIRO GUSTAVO AGUIRRE INGA
JACKSON MARTIN LUZÓN MALDONADO
CARLOS ANDRÉS ACOSTA JARAMILLO
WILSON ALEXIS MÁRQUEZ COCA

DESARROLLO DE APLICACIONES MÓVILES

USANDO LAS APIS DE GOOGLE CLOUD PLATFORM

Religación Press
Ideas desde el Sur Global

int Instituto Nelson Torres

| Colección Software |

Desarrollo de Aplicaciones Móviles usando las APIs de Google Cloud Platform

Juan David Chimarro Amaguaña, Ramiro Gustavo Aguirre Inga,
Jackson Martin Luzón Maldonado, Carlos Andrés Acosta Jaramillo,
Wilson Alexis Márquez Coca

RELIGACION PRESS
QUITO · 2023

Religación Press, es una iniciativa del Centro de Investigaciones CICSHAL-RELIGACIÓN.
Diseño, diagramación y portada: Religación Press.
CP 170515, Quito, Ecuador. América del Sur.
Correo electrónico: press@religacion.com
www.religacion.com

Desarrollo de Aplicaciones Móviles usando las APIs de Google Cloud Platform

Mobile Application Development using Google Cloud Platform APIs
Desenvolvimento de aplicativos móveis usando APIs do Google Cloud Platform

---

Primera Edición:	2023
Editorial:	Religación Press
Materia Dewey:	005 - Programación. programas. datos de computadores
Clasificación Thema:	UMS - Programación de móviles y dispositivos de mano / bolsillo. Programación de "apps"
	ULP - Sistemas operativos para dispositivos móviles
	UDA - Software y aplicaciones de organización personal
BISAC:	COM005000 COMPUTERS / Business & Productivity Software / General
Público objetivo:	Profesional/Académico
Colección:	Software
Soporte:	Digital
Formato:	Epub (.epub)/PDF (.pdf)
Publicado:	2023-12-07
ISBN:	978-9942-642-37-0

Citar como (APA 7)

Chimarro Amaguaña, J.D., Aguirre Inga, R.G., Luzón Maldonado, J.M., Acosta Jaramillo, C.A., y Márquez Coca, W.A. (2023). *Desarrollo de Aplicaciones Móviles usando las APIs de Google Cloud Platform*. Religación Press. https://doi.org/10.46652/ReligacionPress.130

Patrocinio:

Instituto Superior Tecnológico Nelson Torres.

Cayambe, Ecuador. Avenida Luis Cordero, Vía a Ayora.

ISBN: 978-9942-642-37-0

https://press.religacion.com

Revisión por pares / Peer Review

Este libro fue sometido a un proceso de dictaminación por académicos externos. Por lo tanto, la investigación contenida en este libro cuenta con el aval de expertos en el tema, quienes han emitido un juicio objetivo del mismo, siguiendo criterios de índole científica para valorar la solidez académica del trabajo.

This book was reviewed by an independent external reviewers. Therefore, the research contained in this book has the endorsement of experts on the subject, who have issued an objective judgment of it, following scientific criteria to assess the academic soundness of the work.

Sobre los autores

Juan David Chimarro Amaguaña
Instituto Tecnológico Superior Nelson Torres | Cayambe | Ecuador
https://orcid.org/0000-0001-9454-8357
juan.chimarro@intsuperior.edu.ec
Docente Investigador en el área de Desarrollo de Software con una Maestría en Telemática e Ingeniería en Mecatrónica, cursando un Doctorado en Educación con sólida y contrastada experiencia en el manejo de multiplataformas de desarrollo de software y lenguajes de programación, desarrollo de aplicaciones móviles en tiempo real, computación en la nube (Cloud Computing).

Ramiro Gustavo Aguirre Inga
Instituto Tecnológico Superior Nelson Torres | Cayambe | Ecuador
https://orcid.org/0000-0001-8538-4178
ramiro.aguirre@intsuperior.edu.ec
Ingeniero en Sistemas de Información con 8 años de experiencia docente en educación superior y 7 años de trayectoria en Tecnologías de la Información y Comunicación (TIC). Actualmente, se encuentra cursando una maestría en ingeniería de software.

Jackson Martin Luzón Maldonado
Instituto Tecnológico Superior Nelson Torres | Cayambe | Ecuador
https://orcid.org/0000-0003-2349-2733
martin.luzon@intsuperior.edu.ec
Docente Investigador en el área de Desarrollo de Software con una Maestría en Tecnologías de la Información, de profesión Tecnólogo en Análisis de Sistemas, Ingeniero en Informática.

Carlos Andrés Acosta Jaramillo
Instituto Tecnológico Superior Nelson Torres | Cayambe | Ecuador
https://orcid.org/0000-0003-4336-259X
carlos.acosta@intsuperior.edu.ec
Docente Investigador en el área de Desarrollo de Software con una Maestría en Diseño y Gestión de proyectos Tecnológicos, cursando una Maestría en Ingeniería Matemática y Computación, y cursando un Doctorado en Educación, de profesión Ingeniero en Mecatrónica.

Wilson Alexis Márquez Coca
Instituto Tecnológico Superior Nelson Torres | Cayambe | Ecuador
https://orcid.org/0000-0001-7923-9417
alexis.marquez@intsuperior.edu.ec
Ingeniero en Sistemas e Informatica, su desempeño laboral en la Universidad Uniandes por 5 años como Jefe Departamental de telemática de la Sede Ibarra ha consolidado fundamentos técnicos en el area de tecnologías de la información. Actualmente se desempeña como docente en el Instituto superior Nelson Torres. En busca siempre de la modernización de conocimientos en el área de desarrollo de software.

Resumen

El desarrollo de aplicaciones móviles son los procedimientos y procesos establecidos que intervienen cuando se crea software para pequeños dispositivos informáticos inalámbricos, como tabletas y teléfonos inteligentes. Al igual que el desarrollo de aplicaciones web, los procesos de desarrollo de aplicaciones móviles tienen sus raíces en el desarrollo de software tradicional. Desarrollo de Software cuando se trata del desarrollo de aplicaciones móviles, uno requiere acceso a kits de desa-rrollo de software (SDK) que permiten a los programadores diseñar y probar su aplicación de código en un en-torno simulado controlado. Los SDK comúnmente utilizados son: Unity, Android SDK, Licencia de Desarrolla-dor iOS (necesaria para desarrollar aplicaciones para iOS).

Palabras claves: wireframing, software, android, aplicaciones, SDK.

Abstract

Mobile application development is the established procedures and processes involved when creating software for small wireless computing devices, such as tablets and smartphones. Like web application development, mo-bile application development processes have their roots in traditional software development. Software Development when it comes to mobile application development, one requires access to software devel-opment kits (SDKs) that allow programmers to design and test their code application in a controlled simulated environment. Commonly used SDKs are: Unity, Android SDK, iOS Developer License (required to develop iOS applications).

Keywords: wireframing, software, android, applications, SDK.

Resumo

O desenvolvimento de aplicativos móveis são os procedimentos e processos estabelecidos envolvidos na criação de software para pequenos dispositivos de computação sem fio, como tablets e smartphones. Assim como o de-senvolvimento de aplicativos da Web, os processos de desenvolvimento de aplicativos móveis têm suas raízes no desenvolvimento tradicional de software. Desenvolvimento de software Quando se trata de desenvolvimento de aplicativos móveis, é necessário ter acesso a kits de desenvolvimento de software (SDKs) que permitem aos programadores projetar e testar o código do aplicativo em um ambiente simulado controlado. Os SDKs comumente usados são: Unity, Android SDK, iOS Developer License (necessário para desenvolver aplicativos iOS).

Palavras-chave: wireframing, software, android, aplicativos, SDK.

Contenido

Tablas

Figuras

| Colección Software |

Desarrollo de Aplicaciones Móviles usando las APIs de Google Cloud Platform

Prólogo

Con el Desarrollo de Aplicaciones Móviles, aprenderás a crear desde cero tus propias aplicaciones para dispositivos móviles con sistema operativo correspomdientes.

Desde los temas más básicos, como descargar e instalar las herramientas necesarias o crear tu primer proyecto paso a paso, hasta temas avanzados como la localización GPS o la comunicación con servicios web.

El objetivo de esta guía académica es iniciarse en el desarrollo de aplicaciones móviles a través de programación de Android empleando el entorno de desarrollo Android Studio. Se requieren conceptos previos de programación en Java.

Se busca ir conociendo los rudimentos básicos de la programación en Android presentando los conceptos con ejercicios resueltos e invitando a la resolución de otros problemas propuesto.

Funciones específicas de la asignatura en la formación del profesional:

- Es una asignatura que permite al estudiante aprender la adaptación e innovación tecnológica ya que este campo comprende los procesos de exploración del conocimiento que permiten la adaptación, desarrollo e innovación de técnicas y tecnologías, y de la producción.

- Para definir el desarrollo de aplicaciones móviles, debemos empezar por definir móvil, que en este caso se refiere a dispositivos electrónicos portátiles como teléfonos inteligentes, tabletas, ordenadores portátiles, relojes inteligentes, lectores electrónicos y consolas de juegos portátiles. Con el término aplicación, nos referimos no solo al software nativo de esos dispositivos, sino también a los sistemas operativos, las plataformas y los lenguajes comunes que admiten esos dispo-

sitivos. Como se menciona, hay muchas maneras diferentes de abordar el desarrollo de aplicaciones móviles y muchas decisiones que tomar.

• El desarrollo de aplicaciones móviles son los procedimientos y procesos establecidos que intervienen cuando se crea software para pequeños dispositivos informáticos inalámbricos, como tabletas y teléfonos inteligentes. Al igual que el desarrollo de aplicaciones web, los procesos de desarrollo de aplicaciones móviles tienen sus raíces en el desarrollo de software tradicional. Realiza el desarrollo de sus aplicaciones usando plataformas actuales de programación.

• La asignatura permite desarrollar proyectos tanto en el enfoque productivo-empresarial como proyectos sociales, esto permite que se aporte a la sociedad con ideas de innovación.

Introducción

Con la asignatura de Desarrollo de Aplicaciones Móviles, aprenderás a crear desde cero tus propias aplicaciones para dispositivos móviles con sistema operativo correspondientes.

Desde los temas más básicos, como descargar e instalar las herramientas necesarias o crear tu primer proyecto paso a paso, hasta temas avanzados como la localización GPS o la comunicación con servicios web.

El objetivo de este material es iniciarse en el desarrollo de aplicaciones móviles a través de programación de Android empleando el entorno de desarrollo Android Studio. Se requieren conceptos previos de programación en Java.

Se busca ir conociendo los rudimentos básicos de la programación en Android presentando los conceptos con ejercicios resueltos e invitando a la resolución de otros problemas propuesto.

Funciones específicas de la asignatura en la formación del profesional:

- Es una asignatura que permite al estudiante aprender la adaptación e innovación tecnológica ya que este campo comprende los procesos de exploración del conocimiento que permiten la adaptación, desarrollo e innovación de técnicas y tecnologías, y de la producción.
- Para definir el desarrollo de aplicaciones móviles, debemos empezar por definir móvil, que en este caso se refiere a dispositivos electrónicos portátiles como teléfonos inteligentes, tabletas, ordenadores portátiles, relojes inteligentes, lectores electrónicos y consolas de juegos portátiles. Con el término aplicación, nos referimos no solo al software nativo de esos dispositivos, sino también a los sistemas operativos, las plataformas y los len-

guajes comunes que admiten esos dispositivos. Como se menciona, hay muchas maneras diferentes de abordar el desarrollo de aplicaciones móviles y muchas decisiones que tomar.

- El desarrollo de aplicaciones móviles son los procedimientos y procesos establecidos que intervienen cuando se crea software para pequeños dispositivos informáticos inalámbricos, como tabletas y teléfonos inteligentes. Al igual que el desarrollo de aplicaciones web, los procesos de desarrollo de aplicaciones móviles tienen sus raíces en el desarrollo de software tradicional. Realiza el desarrollo de sus aplicaciones usando plataformas actuales de programación.
- La asignatura permite desarrollar proyectos tanto en el enfoque productivo-empresarial como proyectos sociales, esto permite que se aporte a la sociedad con ideas de innovación.

Objetivos

Desarrollar Aplicaciones para dispositivos móviles orientados a satisfacer necesidades empresariales y tecnológicas específicas, aplicando buenas prácticas de programación.

Orientaciones básicas para el estudio

La educación presencial adaptada a una modalidad en línea permitirá fortalecer el aprendizaje autónomo, el cual se exige autodisciplina y alto grado de responsabilidad para cumplir con su responsabilidad académico, por ello para que tenga éxito en el proceso educativo, le sugerimos que sigan las orientaciones metodológicas que se detallan a continuación.

Proceso de enseñanza para el logro de competencias

- Comprende, desarrolla e implementa aplicaciones móviles utilizando tecnología.
- Comprende el manejo de componentes para el desarrollo de aplicaciones móviles.
- Comprende, identifica e implementa aplicaciones móviles con acceso a base de datos.

Capítulo 1
Conceptos básicos en el Desarrollo de Aplicaciones Móviles

1.1 Conceptos básicos en el Desarrollo de Aplicaciones Móviles

Aplicación Móvil (App)

Una aplicación móvil, una aplicación, una apli o una app (acortamiento del inglés application), es una aplicación informática diseñada para ser ejecutada en teléfonos inteligentes, tabletas y otros dispositivos móviles. Este tipo de aplicaciones permiten al usuario efectuar un variado conjunto de tareas —profesional, de ocio, educativas, de acceso a servicios, etc.—, facilitando las gestiones o actividades a desarrollar.

Desarrollo de Aplicaciones Móviles

Según lo definido por MicroServices, el desarrollo de aplicaciones móviles son los procedimientos y procesos establecidos que intervienen cuando se crea software para pequeños dispositivos informáticos inalámbricos, como tabletas y teléfonos inteligentes. Al igual que el desarrollo de aplicaciones web, los procesos de desarrollo de aplicaciones móviles tienen sus raíces en el desarrollo de software tradicional.

1.2 Sistemas operativos para dispositivos móviles

Por lo general, se encuentran disponibles a través de ciertas plataformas de distribución, o por intermedio de las compañías propietarias de los sistemas operativos móviles tales como Android, iOS, BlackBerry OS, Windows Phone, entre otros. Existen aplicaciones móviles gratuitas y otras de pago, donde en promedio el 20 a 30 % del coste de la aplicación se destina al distribui-

dor y el resto es para el desarrollador. El término app se volvió popular rápidamente, tanto que en 2010 fue listada por la American Dialect Society como la palabra del año.

Al ser aplicaciones residentes en los dispositivos están escritas en algún lenguaje de programación compilado, y su funcionamiento y recursos se encaminan a aportar una serie de ventajas tales como:

- Un acceso más rápido y sencillo a la información necesaria sin necesidad de los datos de autenticación en cada acceso.
- Un almacenamiento de datos personales que, a prioridad, es de una manera segura.
- Una gran versatilidad en cuanto a su utilización o aplicación práctica.
- La atribución de funcionalidades específicas.
- Mejorar la capacidad de conectividad, movilidad y disponibilidad de servicios y productos (usuario-usuario, usuario-proveedor de servicios, etc.).

Tipos de Aplicaciones Móviles

La popularidad exponencial de los teléfonos inteligentes y tabletas ha llevado al aumento de la creación de software en línea con el desarrollo de aplicaciones móviles. Los dos sistemas operativos líderes, iOS y Android, han marcado el ritmo en la estandarización de los diferentes tipos de desarrollo de aplicaciones móviles para programadores.

Estos diferentes tipos de aplicaciones móviles incluyen:

Aplicaciones Nativas

Las aplicaciones nativas están diseñadas para plataformas de dispositivos específicos, ya sea Android o iOS. Se descargan o instalan a través de una tienda de aplicaciones y se accede a través de un ícono en el dispositivo. Las aplicaciones nativas están diseñadas para aprovechar al máximo las características del dispositivo como el GPS, la cámara y las listas de contactos, entre otros. Cuando usted piensa en una aplicación móvil, la primera que le viene a la mente es probablemente una aplicación nativa. Ejemplos:

- Aplicaciones de redes sociales: Facebook, Twitter, Pinterest
- Juegos: Pokémon Go, Candy Crush
- Programas de navegación: Waze

Aplicaciones web o HTML5

Basándose en las tecnologías web universales y estandarizadas, como HTML5, JavaScript y CSS, las aplicaciones web se implementan como un sitio web que simplemente se ve y se siente como aplicaciones nativas. Funcionan y se ejecutan en un navegador escrito típicamente en HTML5.

Estas aplicaciones sólo aprovechan las funciones de GPS y cámara de un dispositivo.

Ejemplos: AliExpress, OLX y Twitter Lite.

Aplicaciones híbridas

Las aplicaciones híbridas de JavaScript, HTML y CSS son una combinación de aplicaciones nativas y web. Se obtienen de

una tienda de aplicaciones y aprovechan las características del dispositivo, como una aplicación nativa. Al igual que la aplicación web, se accede desde un navegador y se basa en HTML.

En pocas palabras, las aplicaciones híbridas se instalan como una aplicación nativa pero operan como una aplicación web.

Ejemplos:

- Streaming de motores de búsqueda como Netflix
- Redes sociales como Instagram y twitter
- Aplicaciones de criptomonedas como Cryptochange

Detrás de cada aplicación móvil que a usted le encanta, está el desarrollo de aplicaciones móviles detrás de ella. Detrás de las más de 1,5 millones de aplicaciones disponibles en todas las tiendas de apps, se han invertido innumerables investigaciones y tiempo para desarrollar la mejor aplicación móvil para ofrecer una experiencia de usuario mejorada.

1.3 Tecnologías de Desarrollo de Software

Cuando se trata del desarrollo de aplicaciones móviles, uno requiere acceso a kits de desarrollo de software (SDK) que permiten a los programadores diseñar y probar su aplicación de código en un entorno simulado controlado.

Los SDK comúnmente utilizados son:

- Unity
- Android SDK
- Licencia de Desarrollador iOS (necesaria para desarrollar aplicaciones para iOS)

Para crear una aplicación móvil exitosa, los programadores deben pasar por las siguientes fases:

1. *La investigación:* Refinación de la idea a través de la investigación
2. *Wireframing:* creación del marco esquelético de la aplicación
    - *Evaluación de viabilidad técnica:* teniendo en cuenta los sistemas de back-end de la aplicación
        - *Prototipo:* un prototipo rápido es el concepto de la aplicación en realidad
        - *Diseño:* incluye codificación y diseño de la interfaz
        - *Desarrollo:* desarrollo progresivo de la aplicación
    - *Pruebas:* prueba de funcionalidad y cualquier error que deba corregirse.
    - *Implementación:* presentación de la aplicación móvil final.

1.4 Arquitecturas específicas de desarrollo de aplicaciones móviles: Android Studio y simuladores

1.4.1 Android Studio

Android Studio es el entorno de desarrollo integrado oficial para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. La primera versión estable fue publicada en diciembre de 2014.

Está basado en el software IntelliJ IDEA de JetBrains y ha sido publicado de forma gratuita a través de la Licencia Apache 2.0. Está disponible para las plataformas Microsoft Windows, macOS y GNU/Linux. Ha sido diseñado específicamente para el desarrollo de Android.

Estuvo en etapa de vista previa de acceso temprano a partir de la versión 0.1, en mayo de 2013, y luego entró en etapa beta a partir de la versión 0.8, lanzada en junio de 2014. La primera compilación estable, la versión 1.0, fue lanzada en diciembre de 2014.2

Desde el 7 de mayo de 2019, Kotlin es el lenguaje preferido de Google para el desarrollo de aplicaciones de Android.3 Aún así, Android Studio admite otros lenguajes de programación, como Java y C ++

1.4.2 Simuladores

Podremos destacar que hoy en día la tienda Google Play cuenta con una inmensa variedad de juegos y aplicaciones de diferentes clasificaciones para Android. Si tu intención es jugar algo con gráficos robustos, podemos asegurar que vas a necesitar por lo menos de una tarjeta de video instalada en tu PC para que tengas la mejor experiencia, en ese caso no es relevante el emulador que uses. Los emuladores más populares para Android son:

- MeMU Play
- Genymotion
- LDPlayer
- Ko Player
- NoxPlayer

- YouWave
- BlueStacks
- Xamarin
- ArchON
- AndYroid
- Android Studio Emulator

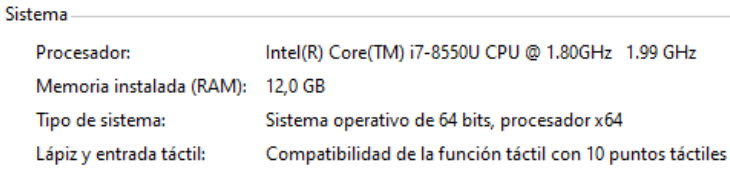
1.4.3 Instalación de las herramientas necesarias para programar para Android Studio

Objetivo: Instalar el software de Android Studio en el ordenador personal.

Tiempo de duración: 2 horas

Preparación previa: El ordenador debe tener las siguientes características.

Figura 1. Características del Computador.



Fuente: Propia.

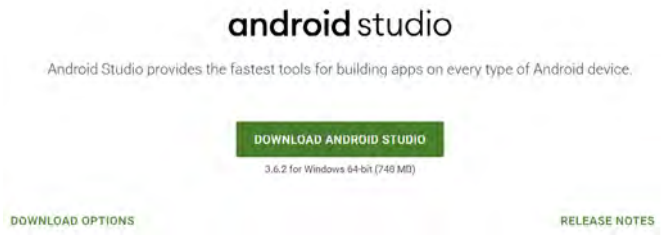
Para empezar con este curso de programación Android, vamos a describir los pasos básicos para disponer en nuestro PC del entorno y las herramientas necesarias para comenzar a programar aplicaciones para la plataforma Android.

En esta entrada me voy a centrar en la instalación de Android Studio sobre Windows.

Paso 1. Descarga e instalación de Android Studio y el SDK de Android

Descargaremos *Android Studio* accediendo a su página principal en la web de desarrolladores de Android. Descargaremos la versión más reciente del instalador correspondiente a nuestro sistema operativo pulsando el botón verde «DOWNLOAD ANDROID STUDIO» y aceptando en la pantalla siguiente los términos de la licencia.

Figura 2. Entorno de descarga de Android Studio.



Fuente: https://developer.android.com/studio

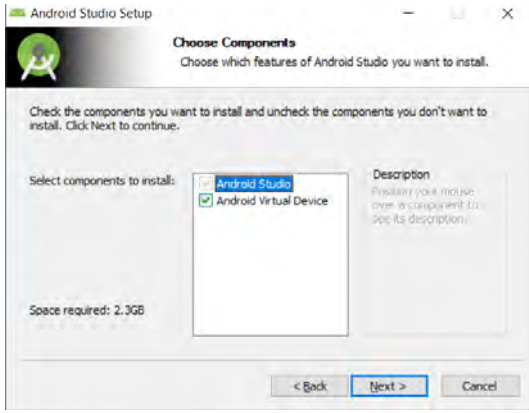
Para instalar la aplicación ejecutamos el instalador descargado (en mi caso el fichero se llama «*android-studio-ide-192.6308749-windows.exe*«) y comenzamos a seguir el asistente de instalación.

Figura 3. Paso 1: Instalar Android Studio.

Fuente: Propia.

Durante este proceso inicial se instalará el IDE y algunos componentes adicionales para el desarrollo sobre la plataforma. En el primer paso del asistente seleccionaremos los componentes a instalar. Los dejaremos todos marcados, en este caso dos: el propio IDE Android Studio y un *Virtual Device* que más adelante veremos qué significa.

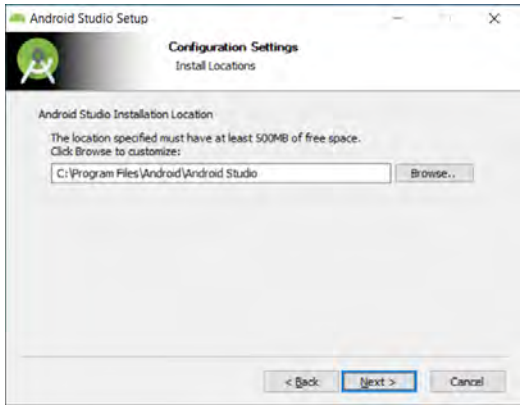
Figura 4. Paso 2: Instalar Android Studio.



Fuente: Propia.

En el siguiente paso seleccionaremos la ruta de instalación de Android Studio.

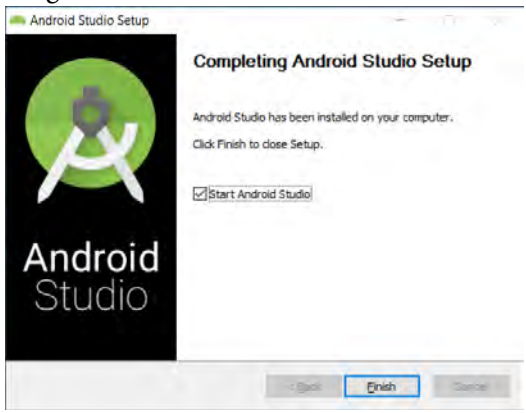
Figura 5. Paso 3: Instalar Android Studio.



Fuente: Propia.

El resto de pasos de este asistente inicial los aceptaremos sin modificar ninguna opción por defecto, hasta llegar al último paso donde marcaremos la opción «Start Android Studio» y pulsaremos el botón «Finish» de forma que se iniciará automáticamente la aplicación.

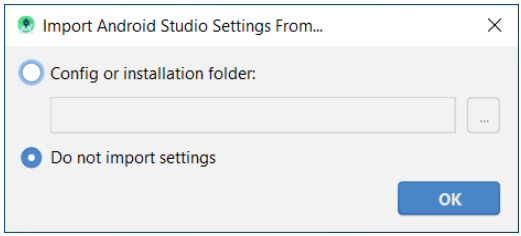
Figura 6. Paso 4: Instalar Android Studio.



Fuente: Propia.

Es posible que nos aparezca en este momento un cuadro de diálogo consultando si queremos reutilizar la configuración de alguna versión anterior del entorno. Para realizar una instalación limpia seleccionaremos la opción «Do not import settings».

Figura 7. Paso 5: Instalar Android Studio.



Fuente: Propia.

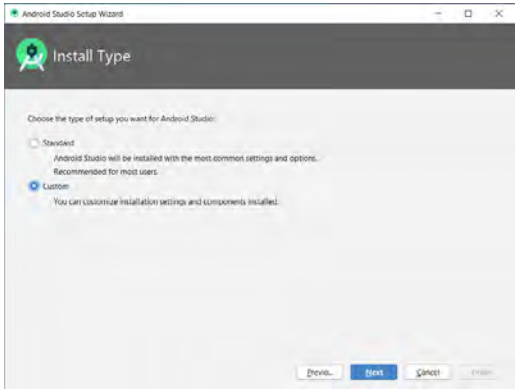
Tras esto, se iniciará el asistente de inicio de Android Studio.

Figura 8. Paso 6: Instalar Android Studio.

Fuente: Propia.

Pulsamos «Next» y en el siguiente paso seleccionamos el modo de instalación «Custom».

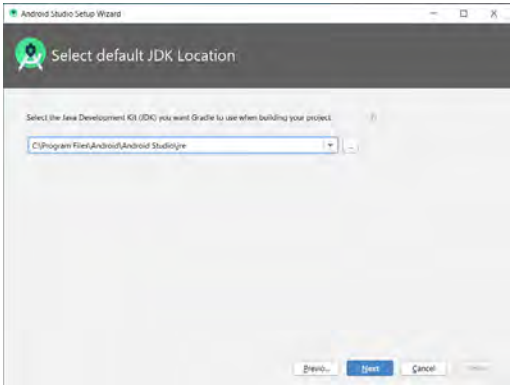
Figura 9. Paso 7: Instalar Android Studio.



Fuente: Propia.

En el siguiente paso seleccionaremos el runtime de java que queremos utilizar, podemos seleccionar un JRE que ya tengamos instalado en nuestra máquina o dejar el valor que aparece por defecto para usar el JRE que se instalará en la carpeta del propio Android Studio.

Figura 10. Paso 8: Instalar Android Studio.

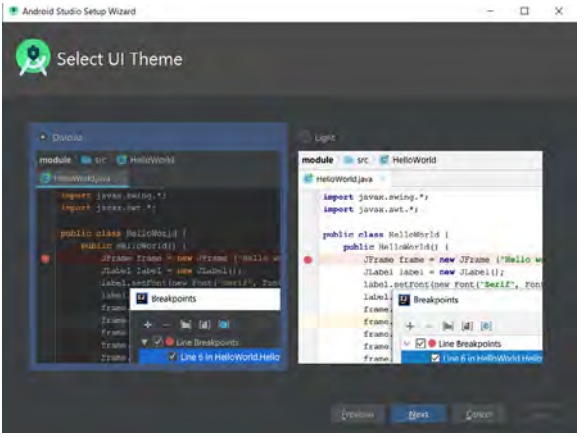


Fuente: Propia.

En el siguiente paso tendremos que decidir el *tema visual* que utilizará la aplicación. Mi recomendación personal es

utilizar el tema oscuro, llamado «Darcula», aunque de cualquier forma es algo que podremos cambiar más adelante.

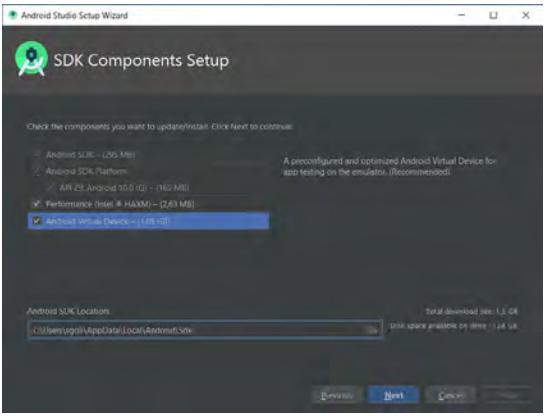
Figura 11. Paso 9: Instalar Android Studio.



Fuente: Propia.

En la siguiente pantalla del asistente seleccionaremos los componentes adicionales que queremos instalar. Marcaremos todos los componentes disponibles, y en el campo «Android SDK Location» indicaremos la ruta donde queremos instalar el SDK de Android.

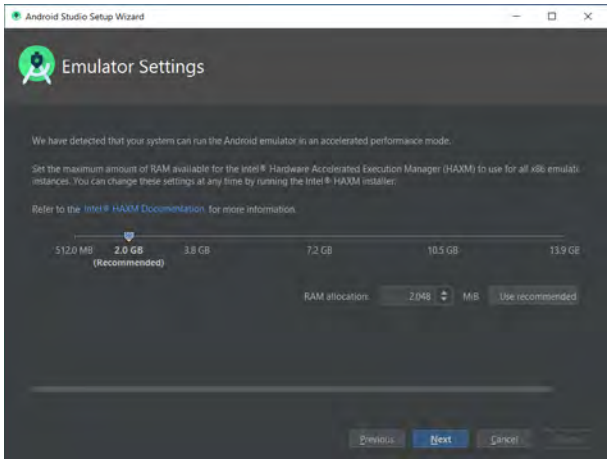
Figura 12. Paso 10: Instalar Android Studio.



Fuente: Propia.

Si nuestro sistema está preparado para ello, en la siguiente pantalla nos aparecerá la configuración del componente «Intel HAXM)». Intel HAXM (*Hardware Accelerated Execution Manager*) es un sistema de virtualización que nos ayudará a mejorar el rendimiento del *emulador de Android* (más adelante hablaremos de esto), y siempre que nuestro sistema lo soporte es muy recomendable instalarlo. En este paso del asistente se podrá indicar la cantidad de memoria que reservaremos para este componente, donde dejaremos seleccionada la opción por defecto (2.0 Gb en mi caso).

Figura 13. Paso 11: Instalar Android Studio.



Fuente: Propia.

Pasamos al siguiente paso, revisamos el resumen de opciones seleccionadas durante el asistente, y pulsamos el botón «Finish» para comenzar con la descarga e instalación de los elementos necesarios. Esperaremos a que finalice dicho proceso y pulsaremos de nuevo el botón «Finish» para terminar por fin con la instalación inicial.

Tras finalizar el asistente de inicio nos aparecerá la pantalla

de bienvenida de Android Studio.

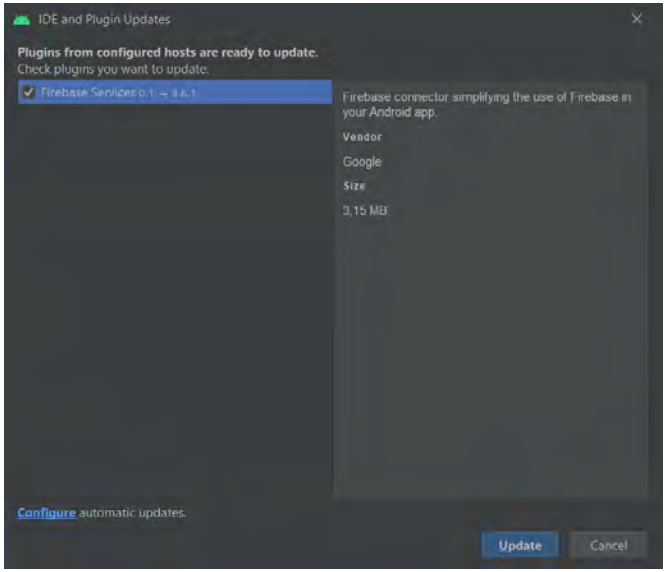
Figura 14. Paso 12: Instalar Android Studio.

Fuente: Propia

Paso 2 (Opcional). Actualización de Android Studio

Podemos comprobar si existe alguna actualización de Android Studio (o alguno de sus componentes) pulsando la opción «Check for Updates» que aparece dentro del menú inferior «Configure». En caso de existir alguna actualización se nos mostrará información sobre ella en una ventana similar a la siguiente (en este caso se informa de la disponibilidad de una nueva versión de un plugin del IDE):

Figura 15. Paso 13: Instalar Android Studio.



Fuente: Propia.

Para instalar la actualización simplemente pulsaríamos el botón «Update». Tras la actualización necesitaremos reiniciar Android Studio para aplicar los cambios y volver a aparecer en la pantalla de bienvenida.

Paso 3. Instalar/actualizar componentes del SDK de Android

El siguiente paso será revisar los componentes que se han instalado del SDK de Android Studio e instalar/actualizar componentes adicionales si fuera necesario para el desarrollo de nuestras aplicaciones.

Para ello accederemos de nuevo al menú «Configure» y pulsaremos esta vez sobre la opción «SDK Manager», lo que nos permitirá acceder al *SDK Manager* de Android. Con esta he-

rramienta podremos instalar, desinstalar, o actualizar todos los componentes disponibles como parte del SDK de Android. Encontraremos los componentes disponibles agrupados en dos pestañas principales: SDK Platforms y SDK Tools.

La primera de ellas, SDK Platforms, permite seleccionar los componentes y librerías necesarias para desarrollar sobre cada una de las versiones concretas de Android. Así, si quisiéramos probar nuestras aplicaciones por ejemplo sobre Android 8.0 y Android 10.0 tendríamos que descargar las dos plataformas correspondientes a dichas versiones. Mi consejo personal es siempre instalar al menos 2 plataformas: la correspondiente a la última versión disponible de Android, y la correspondiente a la mínima versión de Android que queremos que soporte nuestra aplicación, esto nos permitirá al menos probar nuestras aplicaciones sobre ambas versiones para intentar asegurarnos de que funcionará correctamente.

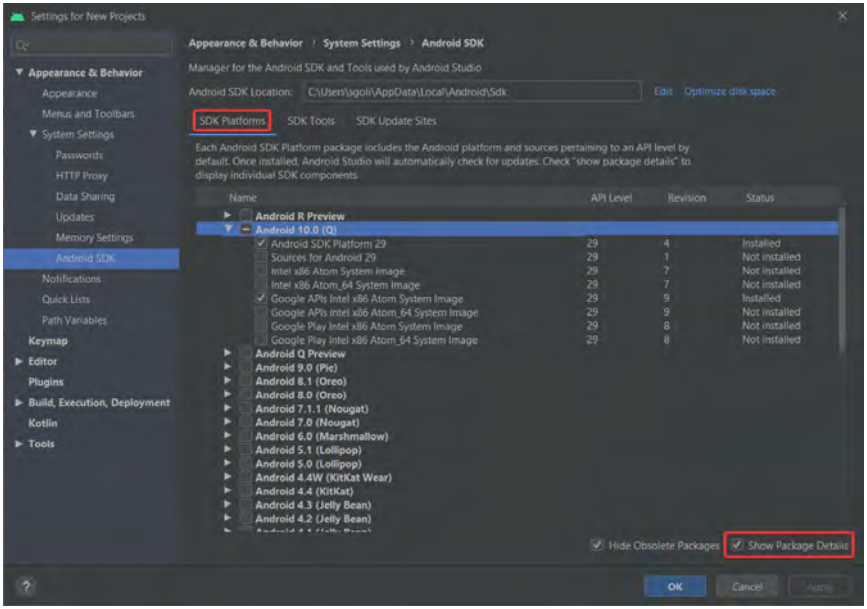
Para ver los subcomponentes de cada plataforma podemos marcar la opción «Show Package Details» situada en la parte inferior de la ventana. El proceso de instalación de Android Studio ya debe haber instalado por nosotros la última versión disponible de Android (en este caso Android 10).

Para cada versión de Android que queramos tener instalada seleccionaremos al menos los dos elementos siguientes:

- Android SDK Platform
- Google APIs Intel x86 Atom System Image

Marcaremos en la lista estos dos componentes para cada versión de Android que deseemos tener instalada en nuestro sistema y pulsaremos el botón «Apply».

Figura 16. Paso 14: Instalar Android Studio.



Fuente: Propia.

En mi caso dejaré tan solo la versión de Android 10, como se ve en la imagen anterior, aunque intentaré que todo lo descrito en este curso funcione al menos desde Android 8.0

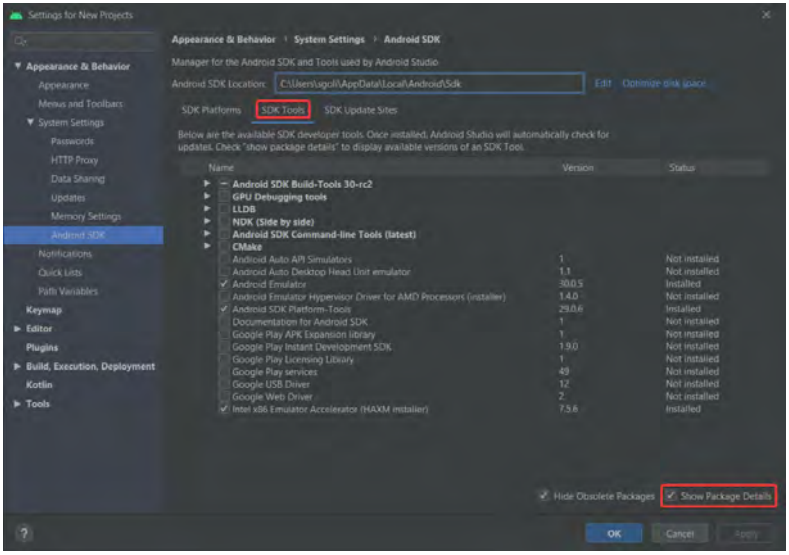
En la segunda de las pestañas indicadas, SDK Tools, podremos seleccionar el resto de componentes necesarios para nuestros desarrollos. Los indispensables por el momento (que ya deberían aparecer instalados por defecto) serán los siguientes:

- Android SDK Build-Tools
- Android SDK Platform-Tools
- Android Emulator

Además de éstos, también aparecerá ya instalado el componente «Intel x86 Emulator Accelerator (HAXM)», que ya comen-

tamos anteriormente, si lo seleccionamos durante el asistente de instalación.

Figura 17. Paso 15: Instalar Android Studio.



Fuente: Propia.

A modo de resumen y/o referencia, en mi caso particular tengo instalados los siguientes componentes/versiones:

- **SDK Tools**:
- Android SDK Build-Tools 29.0.3
- Android SDK Platform-Tools 29.0.6
- Android Emulator 30.0.5
- Intel x86 Emulator Accelerator (HAXM installer) 7.5.6
- **SDK Platforms**:
- Android 10.0 (Q)

- Android SDK Platform 29 (revision 4)
- Google APIs Intel x86 Atom System Image (revision 9)

Si instalamos o actualizamos algún componente adicional, además de los ya instalados por defecto, es recomendable reiniciar Android Studio para que todos los cambios se apliquen correctamente.

Y con este paso ya tendríamos preparadas todas las herramientas necesarias para comenzar a desarrollar aplicaciones Android. En próximas entradas veremos como crear un nuevo proyecto, la estructura y componentes de un proyecto Android, y crearemos y probaremos sobre el emulador una aplicación sencilla para poner en práctica todos los conceptos aprendidos.

1.4.4 Pasos para crear el primer proyecto Android Studio

Una vez que iniciamos el entorno del Android Studio aparece el diálogo principal:

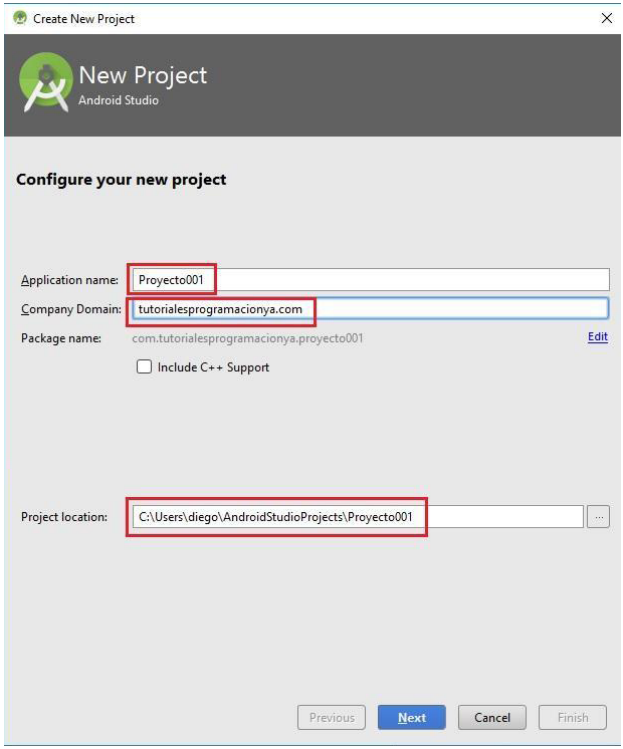
Figura 18. Paso 1: Crear una aplicación móvil en Android Studio.

Fuente: Propia.

Elegimos la opción "Start a New Android Studio project"

Ahora aparecerán una serie de ventanas para configurar el proyecto, el primer diálogo debemos especificar el Nombre de la aplicación, la url de nuestra empresa (que será el nombre del paquete que asigna java para los archivos fuentes) y la ubicación en el disco de nuestro proyecto:

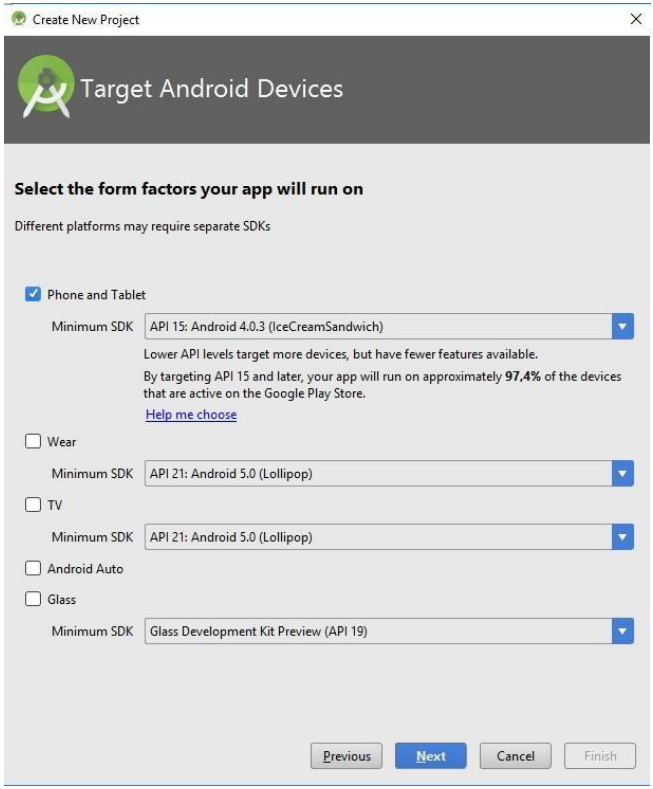
Figura 19. Paso 2: Crear una aplicación móvil en Android Studio.



Fuente: Propia.

En el segundo diálogo procedemos a especificar la versión de Android mínima donde se ejecutará la aplicación que desarrollemos (dejaremos la versión 4.0.3):

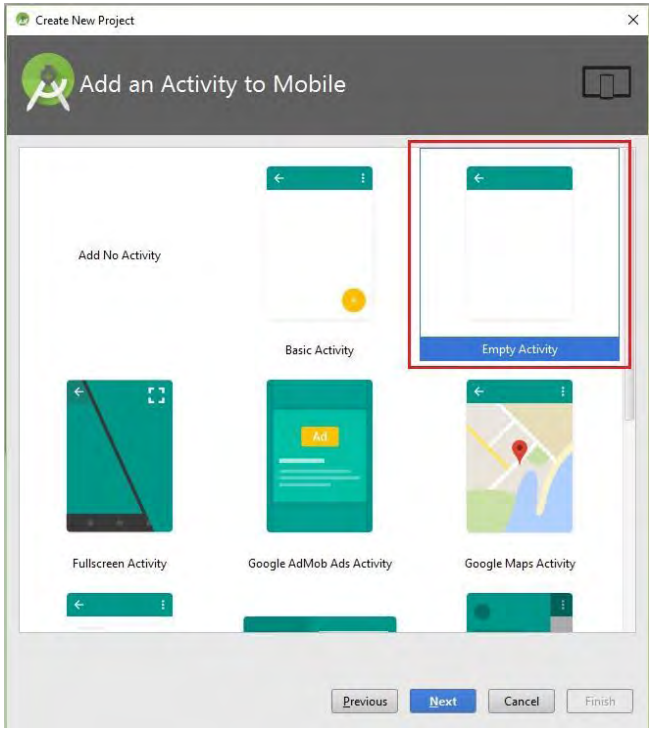
Figura 20. Paso 3: Crear una aplicación móvil en Android Studio.



Fuente: Propia.

El tercer diálogo especificamos el esqueleto básico de nuestra aplicación, seleccionaremos “Empty Activity” si tenemos el Android Studio 2.2:

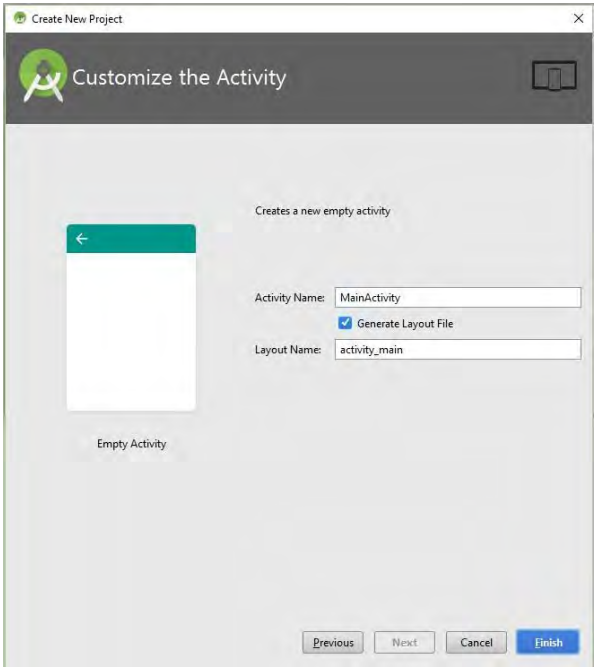
Figura 21. Paso 4: Crear una aplicación móvil en Android Studio.



Fuente: Propia.

Finalmente el último diálogo tenemos que indicar el nombre de la ventana principal de la aplicación (Activity Name) y otros datos más que veremos a lo largo del curso (dejaremos con los nombres por defecto que propone Android Studio):

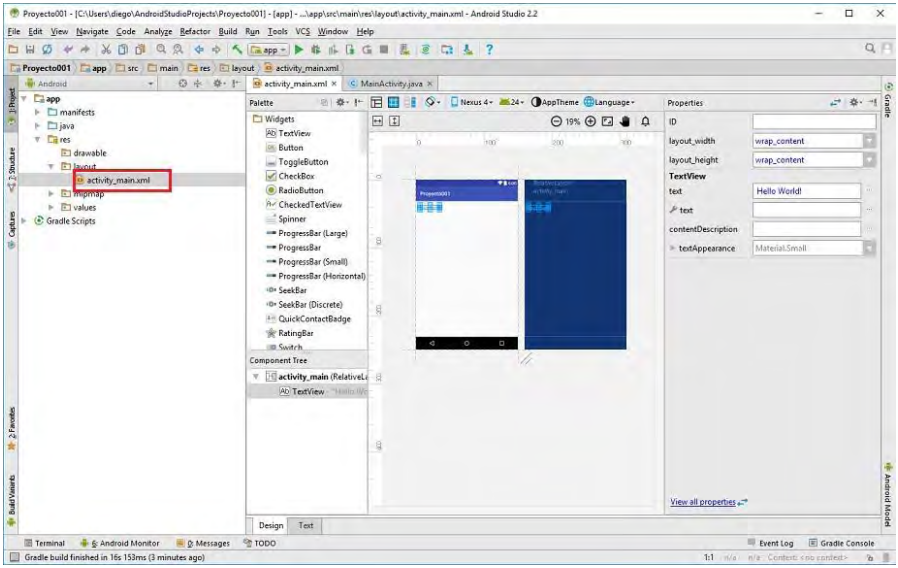
Figura 22. Paso 5: Crear una aplicación móvil en Android Studio.



Fuente: Propia.

Tenemos finalmente creado nuestro primer proyecto en Android Studio y podemos ahora ver el entorno del Android Studio para codificar la aplicación:

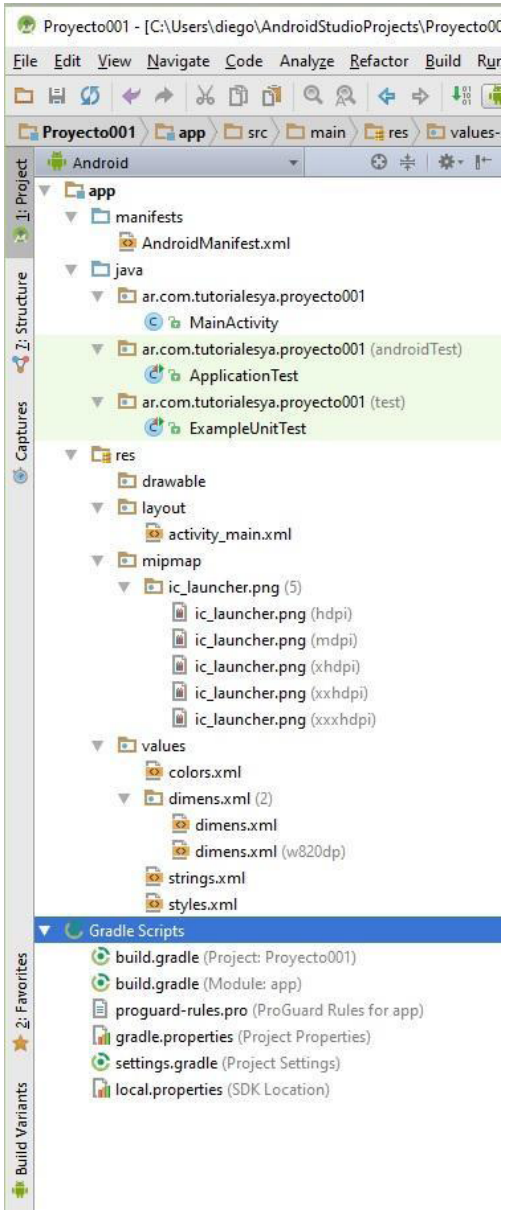
Figura 23. Paso 6: Crear una aplicación móvil en Android Studio.



Fuente: Propia.

El Android Studio nos genera todos los directorios y archivos básicos para iniciar nuestro proyecto, los podemos ver en el lado izquierdo del entorno de desarrollo:

Figura 24. Paso 7: Crear una aplicación móvil en Android Studio.



Fuente: Propia.

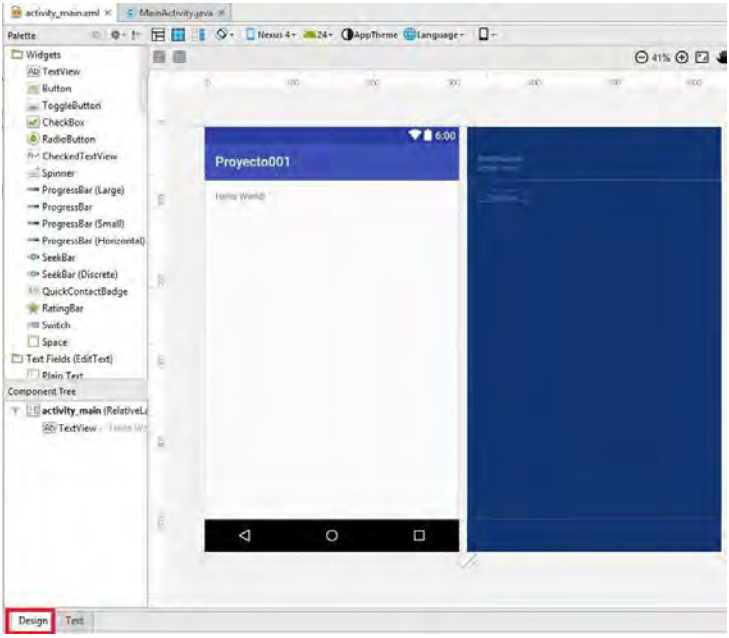
No haremos en este momento un análisis del significado y objetivo de cada uno de estas secciones y archivos generados, sino a medida que avancemos con este curso iremos viendo en forma puntual y profunda.

La interfaz visual de nuestro programa para Android se almacena en un archivo XML en la carpeta res, subcarpeta layout y el archivo se llama activity_main.xml. En esta carpeta tenemos creada nuestra primer pantalla.

Al seleccionar este archivo el Android Studio nos permite visualizar el contenido en "Design" o "Text" (es decir en vista de diseño o en vista de código):

Vista de diseño:

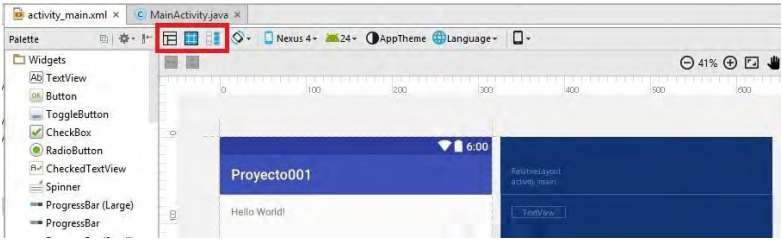
Figura 25. Paso 8: Crear una aplicación móvil en Android Studio.



Fuente: Propia.

A partir de la versión 2.2 del Android Studio tenemos la vista "blueprint" que nos nuestra una interfaz simplificada muy útil cuando tenemos pantallas complejas que veremos más adelante. Podemos ver solo la vista de diseño o "blueprint" seleccionando alguno de los botones que aparecen aquí:

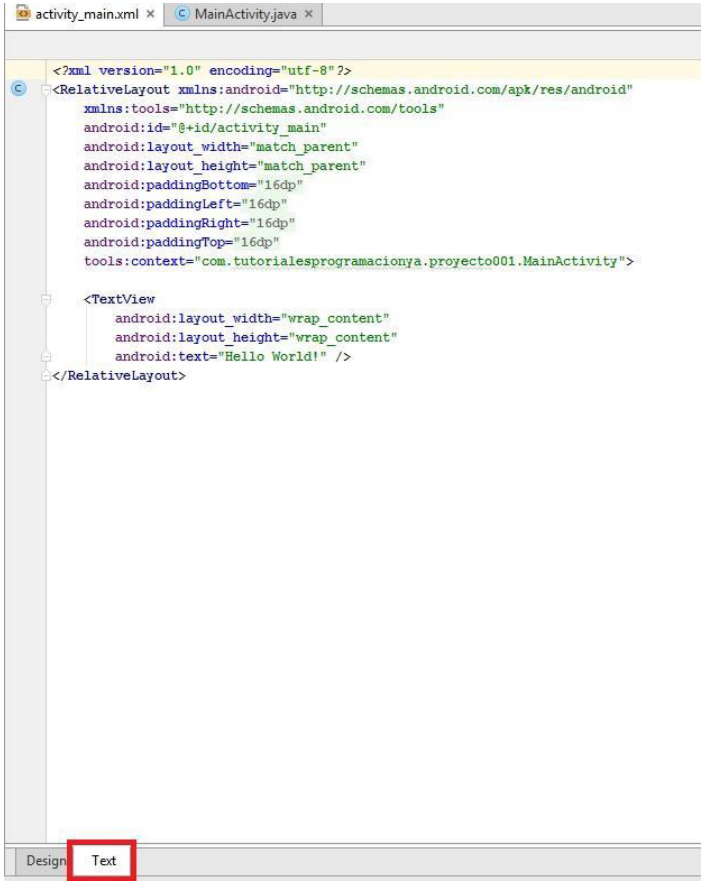
Figura 26. Paso 9: Crear una aplicación móvil en Android Studio.



Fuente: Propia.

Vista de código:

Figura 27. Paso 10: Crear una aplicación móvil en Android Studio.



Fuente: Propia.

El Android Studio ya insertó un control de tipo Relative-Layout que permite ingresar controles visuales alineados a los bordes y a otros controles que haya en la ventana (más adelante analizaremos este layout).

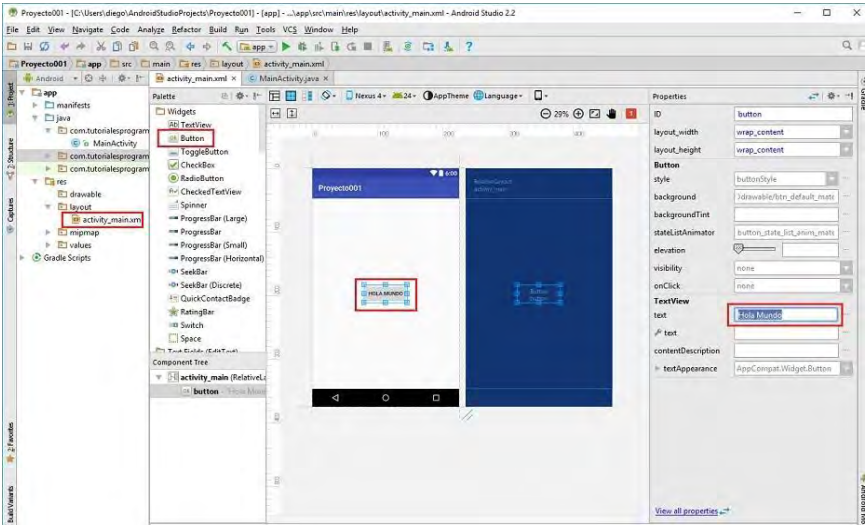
Ya veremos que podemos modificar todo este archivo para

que se adapte a la aplicación que queremos desarrollar.

A lo largo de este curso iremos viendo los objetivos de cada una de las secciones que cuenta el Android Studio para implementar la interfaz, codificar en java las funcionalidades de la aplicación etc.

Antes de probar la aplicación en el emulador de un dispositivo Android procederemos a hacer un pequeño cambio a la interfaz que aparece en el celular: borraremos la label que dice "Hello World" (simplemente seleccionando con el mouse dicho elemento y presionando la tecla delete, podemos seleccionarla de cualquiera de las dos interfaces "Design" o "blueprint") y de la "Palette" arrastraremos un "Button" al centro del celular y en la ventana "Properties" estando seleccionado el "Button" cambiaremos la propiedad "text" por la cadena "Hola Mundo":

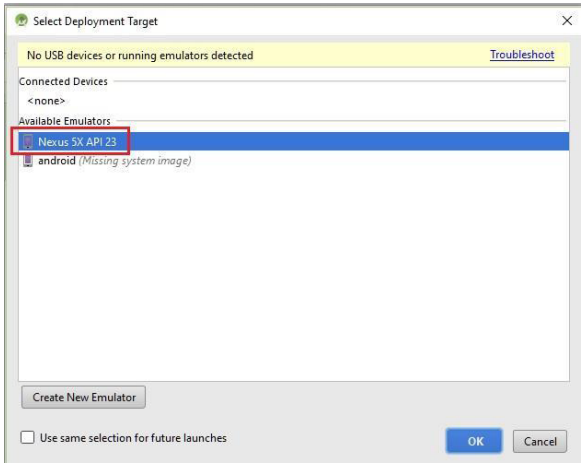
Figura 28. Paso 11: Crear una aplicación móvil en Android Studio.



Fuente: Propia.

Para ejecutar la aplicación presionamos el triángulo verde o seleccionamos del menú de opciones “Run -> Run app” y en este diálogo procedemos a dejar seleccionado el emulador por defecto que aparece (Nexus 5X) y presionamos el botón “OK” (si no tiene ningún emulador puede crear uno):

Figura 29. Paso 12: Crear una aplicación móvil en Android Studio.



Fuente: Propia.

Luego de un rato aparecerá el emulador de Android en pantalla (el arranque del emulador puede llevar más de un minuto), es IMPORTANTE tener en cuenta que una vez que el emulador se ha arrancado no lo debemos cerrar cada vez que hacemos cambios en nuestra aplicación o codificamos otras aplicaciones, sino que volvemos a ejecutar la aplicación con los cambios y al estar el emulador corriendo el tiempo que tarda hasta que aparece nuestro programa en el emulador es muy reducido.

Cuando terminó de cargarse el emulador debe aparecer nuestra aplicación ejecutándose:

Resultados:

Figura 30. Paso 13: Crear una aplicación móvil en Android Studio

Fuente: Propia

1.5 Técnicas de entrada y salida

1.5.1 Controles: EditText y Button

Un elemento de la interfaz de usuario en el que el usuario puede tocar o hacer clic para realizar una acción.

Para mostrar un botón en una actividad, agregue un botón al archivo XML de diseño de la actividad:

<Botón

android: id = "@ + id / button_id"

android: layout_height = "wrap_content"

android: layout_width = "wrap_content"

```
        android: text = "@ string / self_destruct" />
```

Para especificar una acción cuando se presiona el botón, establezca un detector de clics en el objeto del botón en el código de actividad correspondiente:

```
public class MyActivity extiende la actividad {
  protected void onCreate (Bundle SavedInstanceState) {
    super.onCreate (SavedInstanceState);
    setContentView (R.layout.content_layout_id);
    Botón final button = findViewById (R.id.button_id);
    button.setOnClickListener (nuevo View.OnClickListener () {
    public void onClick (Ver v) {
    // El código aquí se ejecuta en el hilo principal después de que el
    usuario presiona el botón
    }
  });
 }
}
```

El fragmento anterior crea una instancia de View.OnClickListenery conecta al oyente al botón que usa setOnClickListener(View.OnClickListener). Como resultado, el sistema ejecuta el código que escribe onClick(View)después de que el usuario presiona el botón.

Cada botón se diseña utilizando el fondo de botón predeterminado del sistema, que a menudo es diferente de una versión de la plataforma a otra. Si no está satisfecho con el estilo de botón predeterminado, puede personalizarlo. Para obtener más detalles y ejemplos de código, consulte la guía Cómo diseñar su botón .

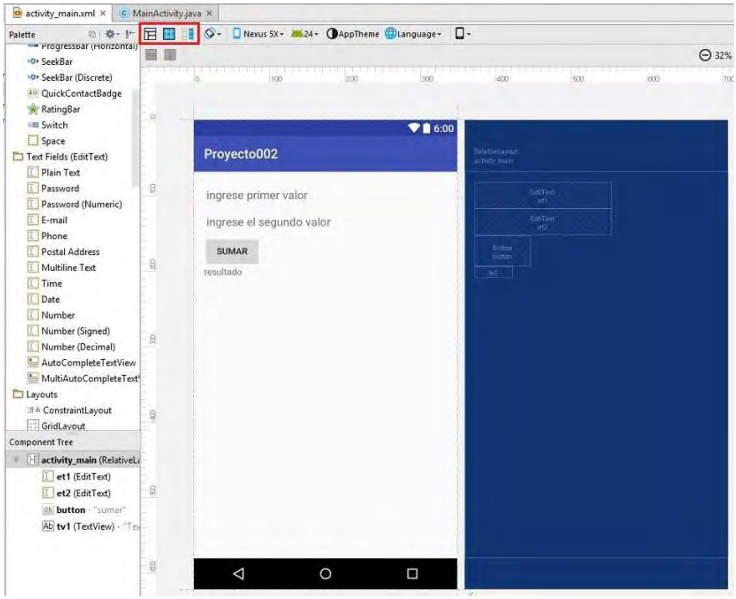
Para los atributos disponibles en el botón ver todo el estilo XML Button Attributes, TextView Attributes, View Attributes. Consulte la guía de estilos y temas para aprender a implementar y organizar anulaciones a los atributos relacionados con el estilo.

1.5.1.1 Problema: Capturar el clic de un botón

Confeccionar un programa que permita la carga de dos números enteros en controles de tipo EditText (Number). Mostrar dentro de los mismos controles EditText mensajes que soliciten la carga de los valores. Disponer un Button para sumar los dos valores ingresados. Mostrar el resultado en un control de tipo TextView.

La interfaz visual debe quedar algo semejante a esto:

Figura 31. Paso 1: Control del Button en Android Studio



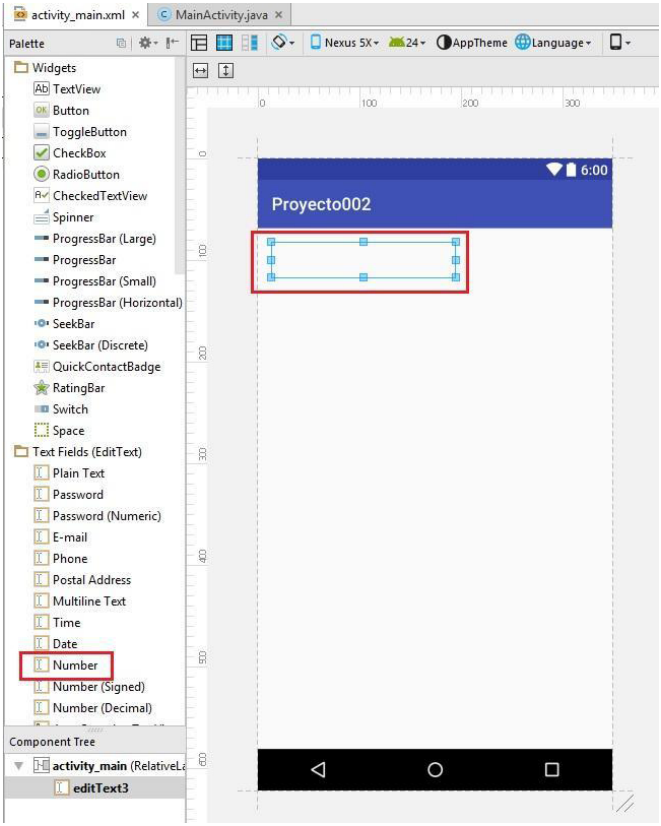
Fuente: Propia

Recordar que si queremos ocultar o volver a mostrar el diseño "blueprint" tenemos tres íconos en la parte superior del diseñador.

Crear un proyecto llamado: Ejercicio002.

Veamos paso a paso como creamos la interfaz visual de nuestro programa. Primero borramos el TextView que aparece por defecto cuando se crea un proyecto con el Android Studio. Ahora desde la ventana "Palette" seleccionamos de la pestaña "Text Fields (EditText)" el control "Number" (es de la clase EditText) y lo arrastramos a la ventana de diseño de nuestra interfaz a la parte superior izquierda:

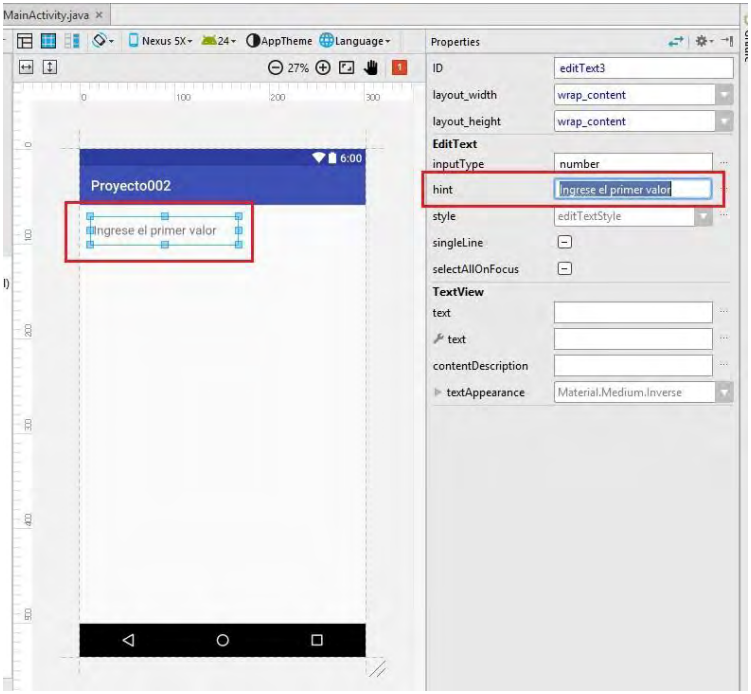
Figura 32. Paso 2: Control del Button en Android Studio.



Fuente: Propia

Ahora lo seleccionamos y en la ventana de propiedades (Properties) especificamos la propiedad hint, disponemos el texto "Ingrese el primer valor":

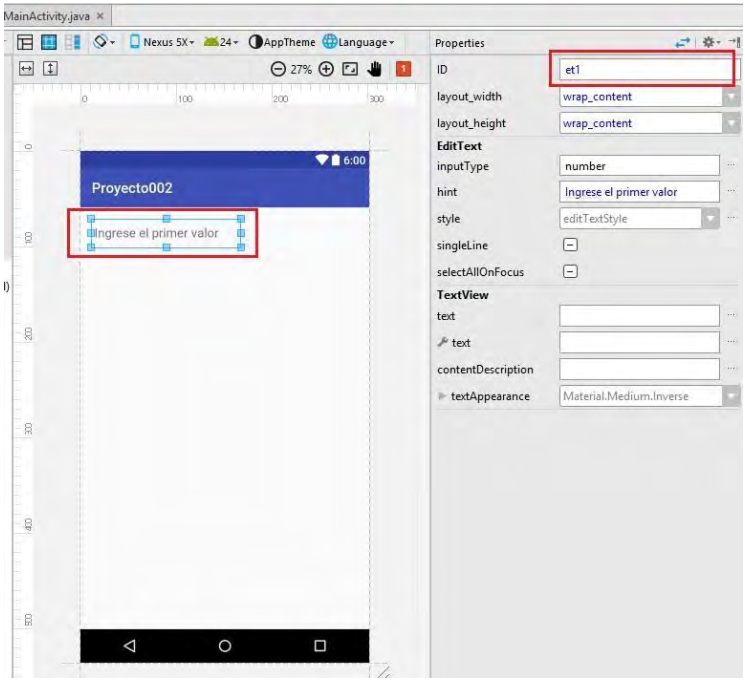
Figura 33. Paso 3: Control del Button en Android Studio.



Fuente: Propia

También vamos a especificar la propiedad "id", y le asignaremos el valor et1.

Figura 34. Paso 4: Control del Button en Android Studio.

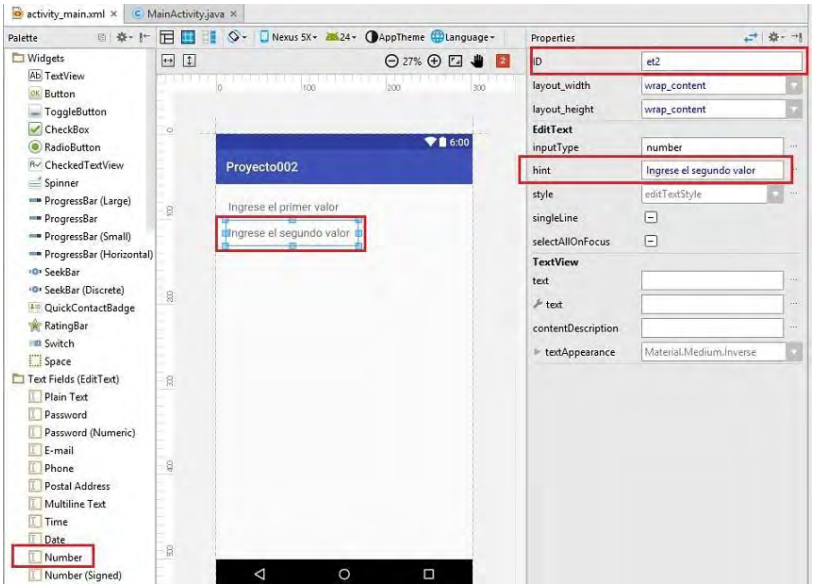


Fuente: Propia

Hemos entonces asignado como nombre a este objeto: et1 (recordemos que se trata de un objeto de la clase EditText), este nombre haremos referencia posteriormente desde el programa en Java.

Efectuamos los mismos pasos para crear el segundo EditText de tipo "Number" (iniciamos las propiedades respectivas) Definimos el id con el nombre et2 y la propiedad hint con el mensaje "Ingrese el segundo valor", el resultado visual debe ser algo semejante a esto:

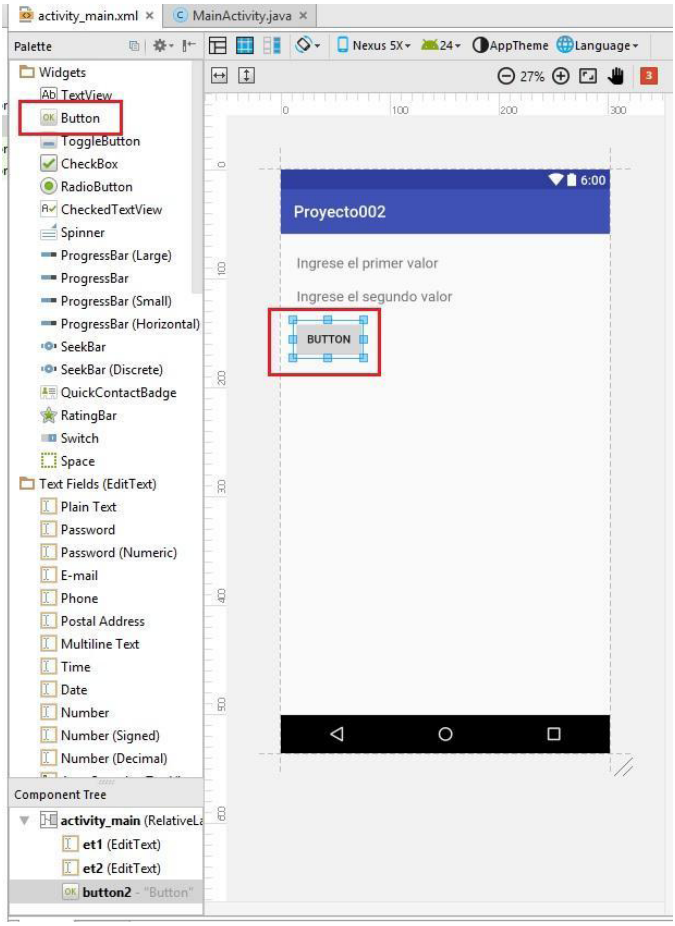
Figura 35. Paso 5: Control del Button en Android Studio.



Fuente: Propia

De la pestaña “Widgets” arrastramos un control de tipo “Button”:

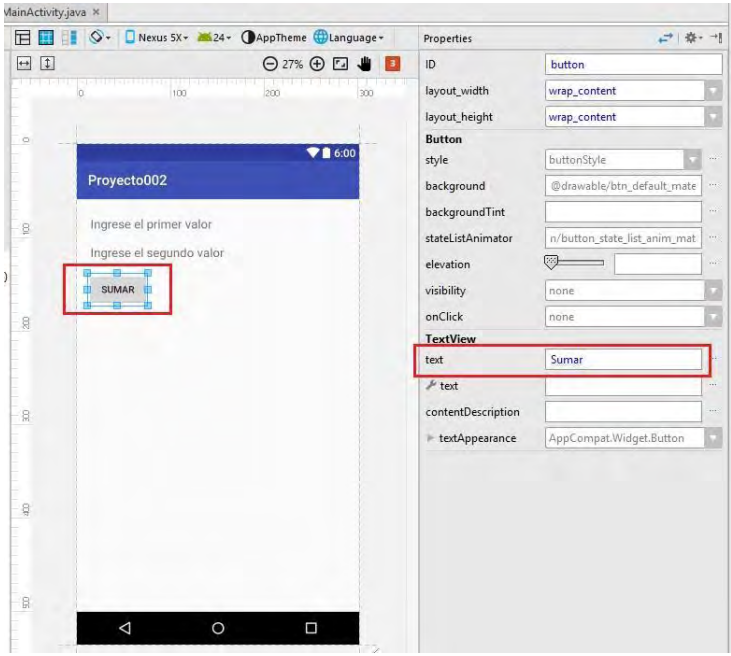
Figura 36. Paso 6: Control del Button en Android Studio.



Fuente: Propia

Iniciamos la propiedad text con el texto “Sumar” y la propiedad id la dejamos con el valor ya creado llamado “button”:

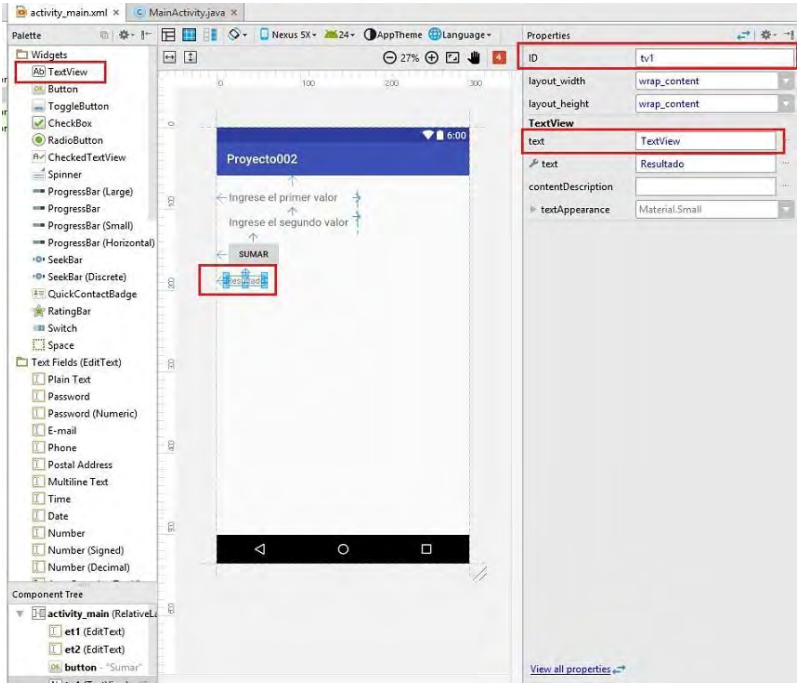
Figura 37. Paso 7: Control del Button en Android Studio.



Fuente: Propia

Para terminar con nuestra interfaz visual arrastramos una componente de tipo “TextView” de la pestaña “Widgets”. Definimos la propiedad id con el valor “tv1” y la propiedad text con el texto “Resultado”:

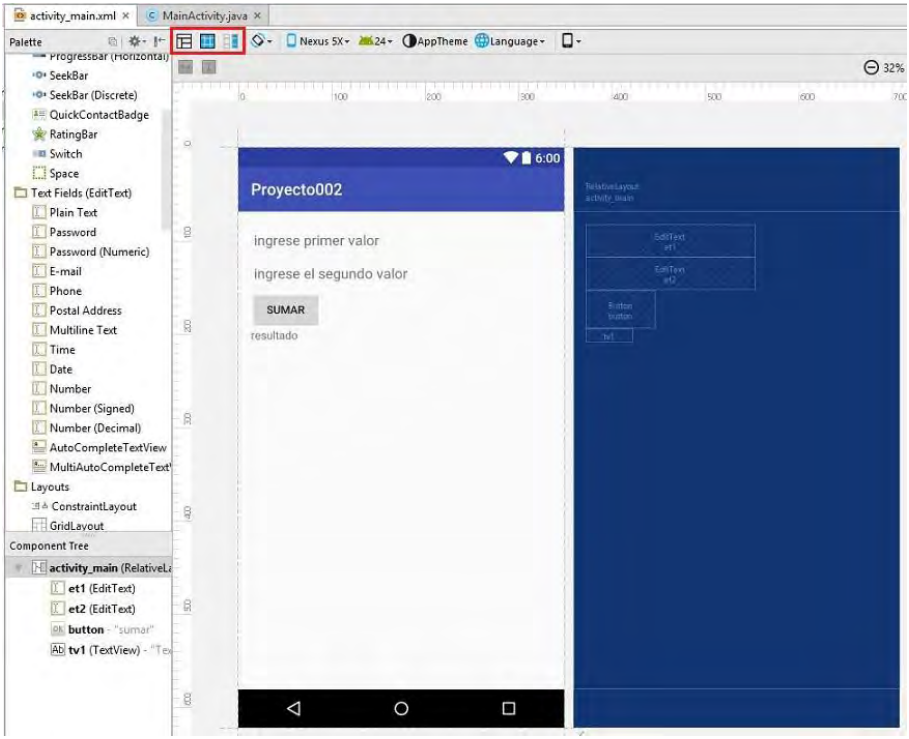
Figura 38. Paso 8: Control del Button en Android Studio.



Fuente: Propia

La interfaz final debe ser semejante a esta:

Figura 39. Paso 9: Control del Button en Android Studio.



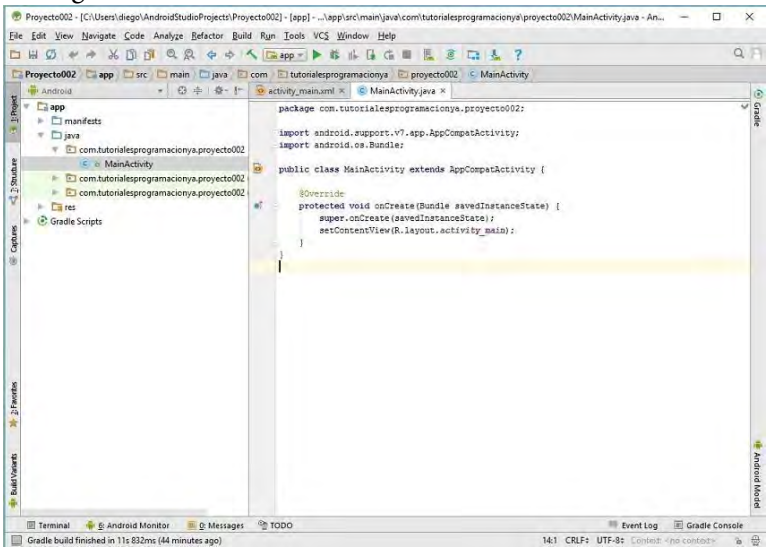
Fuente: Propia

Si en este momento ejecutamos la aplicación aparece la interfaz visual correctamente pero cuando presionemos el botón no mostrará la suma.

Hasta ahora hemos trabajado solo con el archivo xml (activity_main.xml) donde se definen los controles visuales de la ventana que estamos creando.

Abrimos seguidamente el archivo MainActivity.java que lo podemos ubicar en la carpeta app\java\com\tutorialesprogramacionya\proyecto002\MainActivity:

Figura 40. Paso 10: Control del Button en Android Studio



Fuente: Propia

La clase MainActivity hereda de la clase AppCompatActivity. La clase AppCompatActivity representa una ventana de Android y tiene todos los métodos necesarios para crear y mostrar los objetos que hemos dispuesto en el archivo xml.

El código fuente de la clase MainActivity.java es:

```
package com.tutorialesprogramacionya.proyecto002;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
}
```

Como mínimo se debe sobrescribir el método onCreate heredado de la clase AppCompatActivity donde procedemos a llamar al método setContentView pasando como referencia una valor almacenado en una constante llamada activity_main contenida en una clase llamada layout que a su vez la contiene una clase llamada R (veremos más adelante que el Android Studio se encarga de crear la clase R en forma automática y sirve como puente entre el archivo xml y nuestra clase MainActivity)

Captura de eventos

Ahora viene la parte donde definimos variables en java donde almacenamos las referencias a los objetos definidos en el archivo XML.

Definimos tres variables, dos de tipo EditText y finalmente una de tipo TextView (estas dos clases se declaran en el paquete android.widget, es necesario importar dichas clases para poder definir las variables de dichas clases, la forma más fácil de importar las clases es una vez que definimos el objeto por ejemplo private EditText et1; veremos que aparece en rojo el nombre de la clase y nos invita el Android Studio a presionar las teclas "Alt" e "Intro" en forma simultánea. Luego el Android Studio codifica automáticamente la línea que importa la clase: import android.widget.EditText;):

```
package com.tutorialesprogramacionya.proyecto002;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.EditText;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
  private EditText et1;
  private EditText et2;
```

```
  private TextView tv1;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
}
```

Recordar que la forma más fácil de importar las clases EditText y TextView es escribir las tres líneas:

private EditText et1;

private EditText et2;

private TextView tv1;

y luego presionar las teclas "Alt" y "Enter" en cada nombre de clase que se debe importar Esto hace que se escriban automáticamente los import:

import android.widget.EditText;

import android.widget.TextView;

Los nombres que le dí a los objetos en este caso coinciden con la propiedad id (no es obligatorio):

private EditText et1;

private EditText et2;

private TextView tv1;

Para la clase Button no es necesario definir un atributo.

En el método onCreate debemos enlazar estas variables con los objetos definidos en el archivo XML, esto se hace llamando al método findViewById:

```
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    et1=(EditText)findViewById(R.id.et1);
    et2=(EditText)findViewById(R.id.et2);
    tv1=(TextView)findViewById(R.id.tv1);
  }
```

Al método findViewById debemos pasar la constante creada en la clase R (recordemos que se crea automáticamente esta clase) el nombre de la constante si debe ser igual con el nombre de la propiedad del objeto creado en el archivo XML. Como el método findViewById retorna un objeto de tipo View luego debemos utilizar el operador cast (es decir le antecedemos entre paréntesis el nombre de la clase)

Ya tenemos almacenados en las variables las referencias a los tres objetos que se crean al llamar al método:setContentView(R.layout.main); .

Ahora planteamos el método que se ejecutará cuando se presione el botón (el método debe recibir como parámetro un objeto de la clase View) En nuestro ejemplo lo llamé sumar:

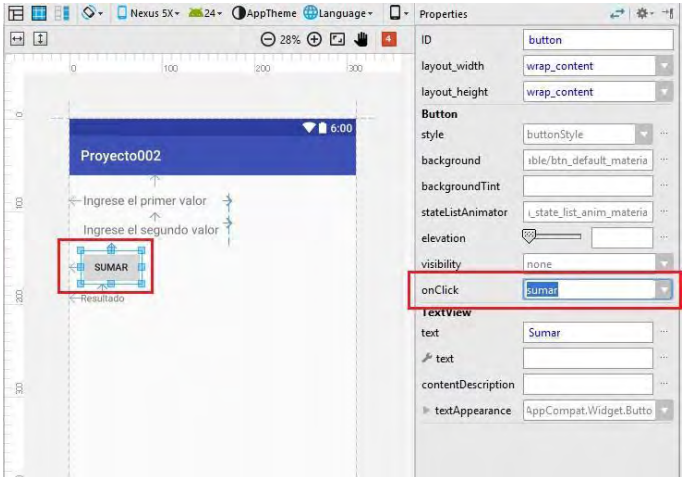
```
package com.tutorialesprogramacionya.proyecto002;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
  private EditText et1;
  private EditText et2;
  private TextView tv1;
  @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        et1=(EditText)findViewById(R.id.et1);
        et2=(EditText)findViewById(R.id.et2);
        tv1=(TextView)findViewById(R.id.tv1);
    }
    //Este método se ejecutará cuando se presione el botón
    public void sumar(View view) {
    }
}
```

Debemos importar lar clase View (presionamos las teclas "Alt" y luego "Enter" en forma simultanea)

Ahora debemos ir al archivo XML (vista de diseño) e inicializar la propiedad onClick del objeto button con el nombre del método que acabamos de crear (este paso es fundamental para que el objeto de la clase Button pueda llamar al método sumar que acabamos de crear):

Figura 41. Paso 11: Control del Button en Android Studio



Fuente: Propia

Finalmente implementaremos la lógica para sumar los dos valores ingresados en los controles EditText:

```
public void sumar(View view) {
    String valor1=et1.getText().toString();
    String valor2=et2.getText().toString();
    int nro1=Integer.parseInt(valor1);
    int nro2=Integer.parseInt(valor2);
    int suma=nro1+nro2;
    String resu=String.valueOf(suma);
    tv1.setText(resu);
}
```

Extraemos el texto de los dos controles de tipo EditText y los almacenamos en dos variables locales de tipo String. Convertimos los String a tipo entero, los sumamos y el resultado lo enviamos al TextView donde se muestra la suma (previo a convertir la suma a String)

La clase completa queda entonces como:

```
package com.tutorialesprogramacionya.proyecto002;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
    private EditText et1;
    private EditText et2;
    private TextView tv1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    et1=(EditText)findViewById(R.id.et1);
    et2=(EditText)findViewById(R.id.et2);
    tv1=(TextView)findViewById(R.id.tv1);
  }
    //Este método se ejecutará cuando se presione el botón
    public void sumar(View view) {
    String valor1=et1.getText().toString();
    String valor2=et2.getText().toString();
    int nro1=Integer.parseInt(valor1);
    int nro2=Integer.parseInt(valor2);
    int suma=nro1+nro2;
    String resu=String.valueOf(suma);
    tv1.setText(resu);
  }
}
```

Si ejecutamos nuestro programa podemos ver ahora que los controles EditText muestran los mensajes “Ingrese primer valor” e “Ingrese segundo valor” (la propiedad hint de los EditText muestran un mensaje que se borra automáticamente cuando el operador carga los enteros):

Figura 42. Paso 12: Control del Button en Android Studio

Fuente: Propia

Luego de cargar dos valores al presionar el botón aparece en el TextView el resultado de la suma de los dos EditText :

Figura 43. Paso 13: Control del Button en Android Studio

Fuente: Propia

1.5.2 Controles RadioGroup y RadioButton

El objetivo de este concepto es practicar la implementación de un programa que requiera controles de tipo RadioButton para seleccionar una actividad. Aprenderemos como agrupar un conjunto de RadioButton y verificar cual está seleccionado.

1.5.2.1 Problema:

Realizar la carga de dos números en controles de tipo EditText. Mostrar mensajes que soliciten la carga de los valores dentro de los mismos EditText (propiedad hint). Disponer dos controles de tipo RadioButton para seleccionar si queremos sumar o restar dichos valores. Finalmente mediante un control de tipo Button efectuamos la operación respectiva. Mostramos el resultado en un TextView.

1.5.3 Control CheckBox

El objetivo de este concepto es seguir practicando lo visto hasta ahora para la creación de un proyecto con Android Studio e incorporar el control visual CheckBox

1.5.3.1 Problema:

Realizar la carga de dos números en controles de tipo EditText ("Number"). Mostrar en las propiedades "hint" de cada componente un mensaje que solicite la carga de los valores. Disponer dos controles de tipo CheckBox para seleccionar si queremos sumar y/o restar dichos valores. Finalmente mediante un control de tipo Button efectuamos la operación respectiva. Mostramos el o los resultados en un TextView.

1.5.4 Control Spinner

El objetivo de este concepto es seguir practicando lo visto hasta ahora e incorporar el control visual Spinner.

El control Spinner muestra una lista de String y nos permite seleccionar uno de ellos. Cuando se lo selecciona se abre y muestra todos sus elementos para permitir seleccionar uno de ellos.

Problema:

Realizar la carga de dos números en controles de tipo EditText ("Number"). Mostrar un mensaje que solicite la carga de los valores iniciando la propiedad "hint" de cada control. Disponer un control de tipo Spinner que permita seleccionar si queremos sumar, restar, multiplicar o dividir dichos valores. Finalmente mediante un control de tipo Button efectuamos la operación respectiva. Mostramos el resultado en un TextView.

1.5.5 Control ListView (con una lista de String)

El control ListView a diferencia del Spinner que se cierra luego de seleccionar un elemento permanecen visibles varios elementos (se lo utiliza cuando hay que mostrar muchos elementos)

En este primer ejemplo mostraremos una lista de String (es decir cada elemento de la lista es un String, veremos más adelante que podemos tener listas de objetos de otro tipo: imágenes, íconos, varios String por elemento etc.)

Si la lista no entra en el espacio que hemos fijado para el ListView nos permite hacer scroll de los mismos.

El control ListView se encuentra en la pestaña "Containers".

1.5.5.1 Problema:

Disponer un ListView con los nombres de paises de sudamérica. Cuando se seleccione un país mostrar en un TextView la cantidad de habitantes del país seleccionado.

1.5.6 Control ImageButton

Hemos visto la creación de objetos de la clase Button, ahora veremos otra clase muy similar a la anterior llamada ImageButton que tiene la misma filosofía de manejo con la diferencia que puede mostrar una imagen en su superficie.

1.5.6.1 Problema:

Disponer un objeto de la clase ImageButton que muestre una imagen de un teléfono. Cuando se presione mostrar en un control TextView el mensaje "Llamando".

1.5.7 Notificaciones sencillas mediante la clase Toast

Android permite mostrar una ventana emergente temporal que informa al usuario mediante un mensaje que aparece en la pantalla por un lapso pequeño de tiempo (luego de pasado un tiempo la ventana desaparece).

Esta ventana que se superpone a la interfaz que se está mostrando en ese momento se administra mediante una clase llamada Toast.

1.5.7.1 Problema:

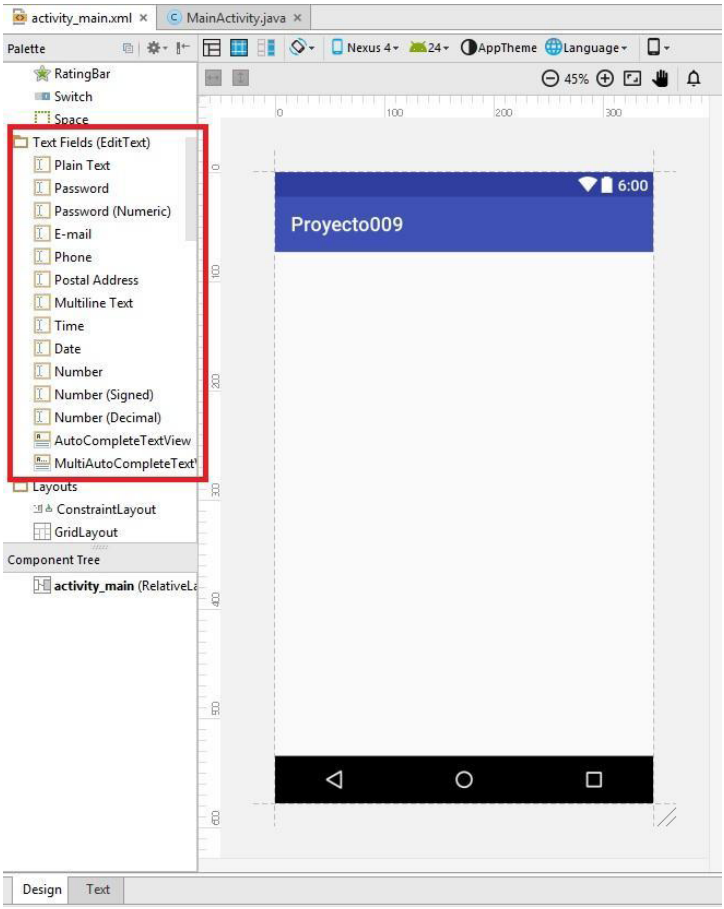
Generar un número aleatorio entre 1 y 100000. Mostrar mediante una ventana emergente dicho número por un lapso de tiempo. Luego mediante un control EditText (“Number”) pedir al operador que ingrese el número que vio en la pantalla. Cuando se presione un botón controlar el número generado aleatoriamente con el que ingresó el usuario y mediante otro Toast informar si acertó o no.

1.5.8 Control EditText

Desde el primer problema hemos estado utilizando el control que permite en Android ingresar valores por teclado. La clase que administra la entrada de caracteres por teclado es EditText.

Pero en la palette de componentes vemos que hay muchos tipos de EditText:

Figura 44. Palette de componentes EditText



Fuente: Propia

Como podemos ver en la pestaña "Text Fields (EditText)" se encuentran todos los tipos de EditText que nos ofrece Android para utilizar en nuestras aplicaciones: Password, E-mail, Number, etc.

Dependiendo del tipo de entrada de datos que necesitemos utilizaremos un tipo específico de EditText.

1.5.8.1 Problema

Confeccionar una aplicación para android que permita ingresar el nombre de usuario y su clave en dos controles de tipo EditText.

Verificar al presionar un botón si se ingresó algún texto en la clave. Si no se ingresó texto informar mediante una notificación dicha situación.

1.5.9 Lanzar un segundo "Activity"

Hasta ahora todos los programas han tenido una sola ventana (Activity). Es muy común que una aplicación tenga más de una ventana. Para implementar esto en Android debemos plantear otros dos archivos uno xml con la interfaz y otro java con la lógica (tengamos en cuenta que cuando utilizamos Android Studio automáticamente cuando creamos un proyecto nos crea el archivo XML y el código java del primer Activity)

Vamos a ver en este concepto los pasos que debemos dar para crear otro Activity y como activarlo desde el Activity principal.

1.5.9.1 Problema:

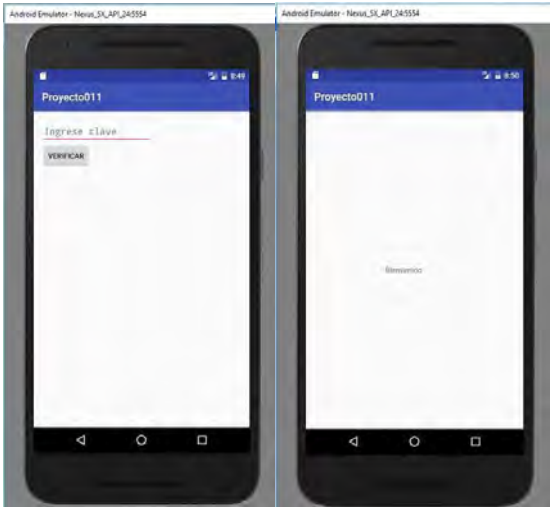
Confeccionar un programa que muestre en la ventana principal un botón que al ser presionado muestre otra ventana (Activity) mostrando un TextView con el nombre del programador de la aplicación y un botón para cerrar la ventana o actividad y que vuelva al primer Activity.

1.5.9.2 Problema propuesto

Realizar un programa que contenga dos Activity. En el primero que solicite el ingreso de una clave (control Password) Si ingresa la clave "abc123" activar el segundo Activity mostrando en un TextView un mensaje de bienvenida (mostrar en Toast si se ingresa la clave incorrecta en el primer Activity).

En tiempo de ejecución los dos Activity deben mostrarse algo similar a esto:

Figura 45. Ejercicio propuesto lanzar Activity



Fuente: Propia

1.5.10 Lanzar un segundo "Activity" y pasar parámetros

Hemos visto en el concepto anterior que un programa puede tener más de una ventana representando cada ventana con una clase que hereda de AppCompatActivity.

Una situación muy común es que la primer ventana necesite enviar datos a la segunda para que a partir de estos proceda a efectuar una acción.

1.5.10.1 Problema:

Confeccionar un programa que solicite el ingrese de una dirección de un sitio web y seguidamente abrir una segunda ventana que muestre dicha página.

Para resolver este problema utilizaremos el control visual WebView que nos permite mostrar el contenido de un sitio web.

1.5.11 Evaluación 1

1. La definición de Aplicación Móvil
2. Los sistemas operativos para dispositivos móviles
3. Al ser aplicaciones residentes en los dispositivos móviles. ¿Cuál de las siguientes opciones es una ventaja?
4. ¿Qué permite a los programadores los kits de desarrollo de software (SDK)?
5. ¿Que caracteriza a una aplicación nativa?
6. Ejemplos de aplicaciones híbridas.
7. La interfaz visual de nuestro programa para Android se almacena en un archivo XML, Que contenido nos permite visualizar
8. ¿Cuales son emuladores para Android?
9. En Android Studio, cual es el método que se usa para enlazar las variables de los objetos definidos en el archivo XML con las variables de la clase en JAVA.
10. ¿Cual control me permite mostrar una lista de String y nos permite seleccionar uno de ellos?

1.6 Entornos de Desarrollo

1.6.1 Instalación del programa Android en un dispositivo real

Normalmente uno cuando desarrolla aplicaciones en Android hace toda la programación, depuración y pruebas en un dispositivo virtual en la pc. Ahora vamos a ver los pasos para probar la aplicación en un dispositivo Android real.

La primera forma que veremos de probar nuestra aplicación es copiar el archivo con extensión APK a nuestro dispositivo.

1. Primero en el ícono de configuración de nuestro teléfono o tablet android seleccionamos la opción "Aplicaciones" -> y marcamos la opción "Origen desconocido (Permitir la instalación de aplicaciones no pertenecientes al mercado)"
2. Desde nuestro equipo de escritorio enviamos un mail a nuestro celular adjuntando el archivo con extensión apk que se encuentra en el directorio app/build/outputs/apk de nuestro proyecto.
3. Abramos el mail desde nuestro celular y seleccionemos el archivo adjunto. Confirmamos que queremos instalarlo.

1.6.2 Layout (LinearLayout)

Android organiza las componentes visuales (Button, EditText, TextView etc.) en pantalla mediante contenedores llamados Layout.

Hasta ahora no nos ha preocupada como organizar una pantalla, sino nos hemos centrado en la funcionalidad de cada programa que implementamos.

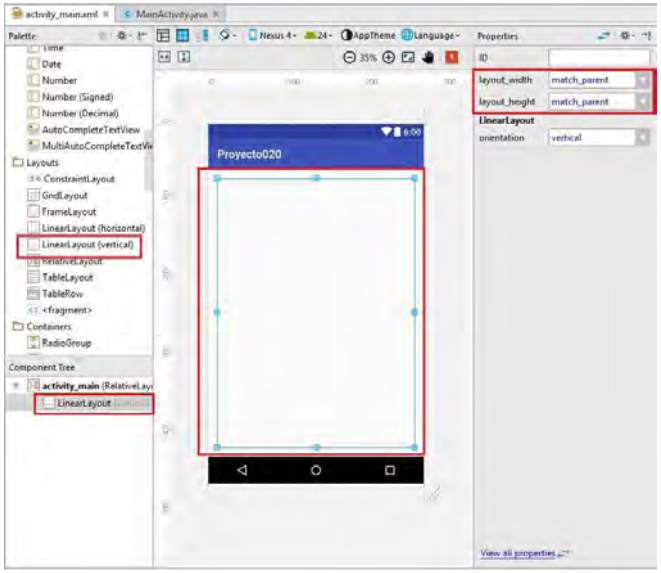
Ahora comenzaremos a preocuparnos como organizar y disponer las componentes dentro de la pantalla.

LinearLayout es uno de los diseños más simples y más empleado. Simplemente establece los componentes visuales uno junto al otro, ya sea horizontal o verticalmente.

Creemos un proyecto, borremos el TextView que agrega por defecto el Android Studio.

De la "Palette" de componentes de Android Studio arrastraremos de la pestaña "Layouts" el objeto de la clase "Linear Layout (Vertical)":

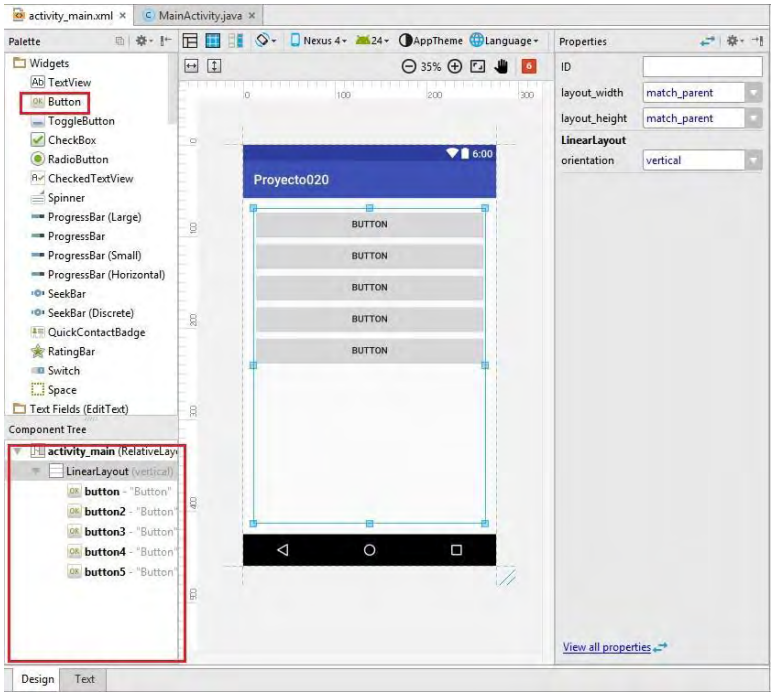
Figura 46. Paso 1: Manejo Linear Layout



Fuente: Propia

Ahora podemos ver que sucede cuando disponemos cinco botones dentro del contenedor LinearLayout (Todos los botones se disponen obligatoriamente uno debajo del otro y no hay posibilidad de ubicarlos con el mouse):

Figura 47. Paso 2: Manejo Linear Layout



Fuente: Propia

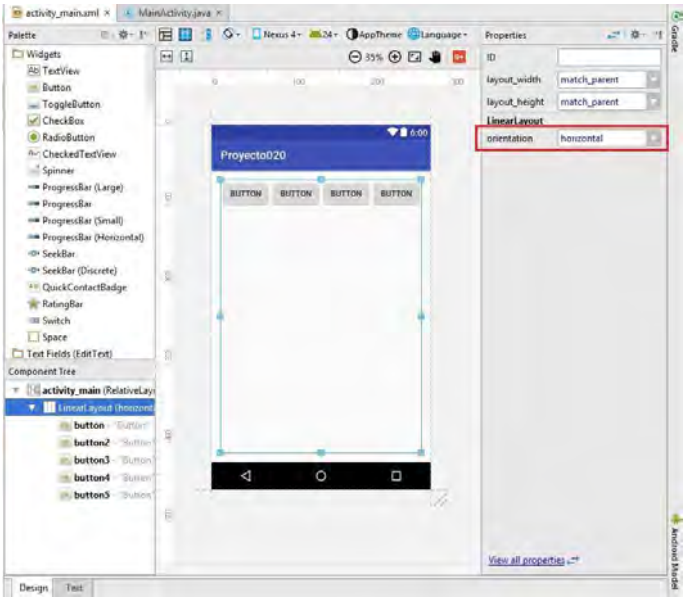
El primer cambio que podemos hacer a esta tipo de Layout es cambiar la propiedad “orientation” por el valor horizontal (esto hace que los botones se dispongan uno al lado del otro y no hay posibilidad de disponer controles uno debajo de otro)

Hay que tener en cuenta que si los controles no entran en pantalla los mismos no se ven y si son botones como en este caso es imposible hacer clic sobre los mismos:

También debemos seleccionar cada Button y cambiar la propiedad layout_width por el valor "wrap_content" para que cada botón no ocupe todo el ancho del dispositivo.

Luego podemos ver que inclusive si hay muchos botones no todos se ven:

Figura 48. Paso 3: Manejo Linear Layout



Fuente: Propia

Tengamos en cuenta que en la "Palette" de Android Studio aparecen dos opciones para disponer "Linear Layout(Horizontal)" o "Linear Layout(Vertical)" la única diferencia es que se inicializa la propiedad "orientation" con el valor: horizontal o vertical.

Eliminando el RelativeLayout de la raíz de la aplicación.

Si no necesitamos que la raíz de la interfaz defina un RelativeLayout podemos disponer en su lugar por ejemplo un Li-

nearLayout. Modificaremos el programa hecho hasta ahora de tal forma que la interfaz sea solo un LinearLayout con orientación vertical y con 5 botones en su interior.

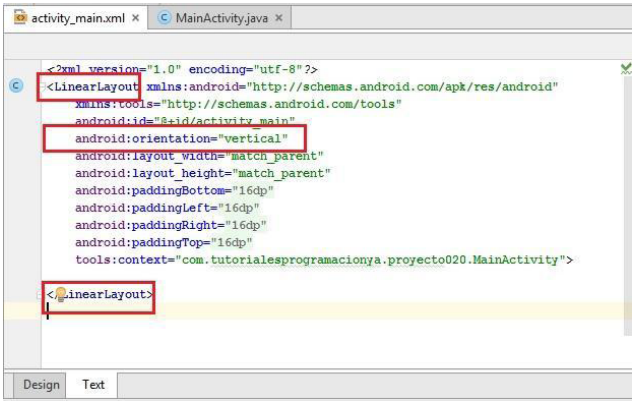
No lo podemos hacer en forma visual con Android Studio, debemos modificar manualmente el código del archivo XML, cambiamos la vista "Design" por "Text":

Figura 49. Paso 4: Manejo Linear Layout

Fuente: Propia

Procedemos a modificar el nombre del Layout y definir e inicializar la propiedad "orientation" (Remplazamos el nombre RelativeLayout por LinearLayout y agregamos la propiedad android:orientation="vertical"):

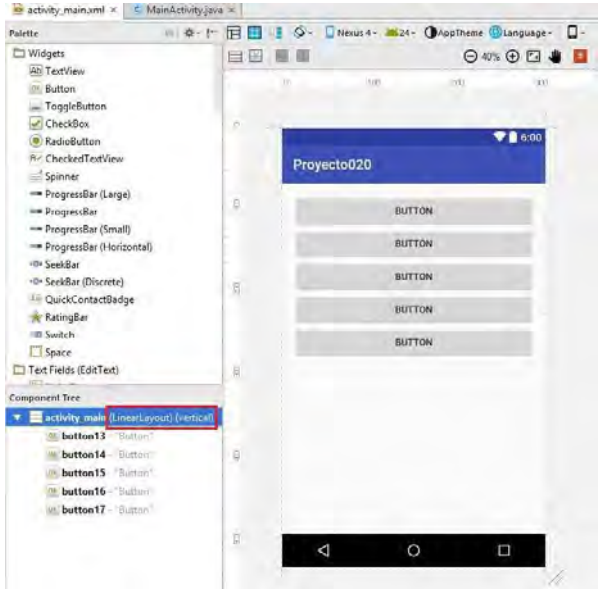
Figura 50. Paso 5: Manejo Linear Layout



Fuente: Propia

Ahora podemos disponer los botones en la interfaz y aparecerán uno debajo del otro similar a como empezamos pero será mas eficiente ya que no hay un RelativeLayout en la raíz de la interfaz:

Figura 51. Paso 6: Manejo Linear Layout



Fuente: Propia

1.6.3 Layout (TableLayout)

El Layout de tipo TableLayout agrupa componentes en filas y columnas. Un TableLayout contiene un conjunto de componentes de tipo TableRow que es el que agrupa componentes visuales por cada fila (cada fila puede tener distinta cantidad de componentes visuales)

1.6.3.1 Problema

Disponer 9 botones en forma de un tablero de TaTeTi. Utilizar un TableLayout, tres TableRow y nueve botones.

1.6.4 Layout (FrameLayout)

El control de tipo FrameLayout dispone dentro del contenedor todos los controles visuales alineados al vértice superior izquierdo,centrado, vértice inferior derecho etc. (tiene nueve posiciones posibles).

Si disponemos dos o más controles los mismos se apilan.

Por ejemplo si disponemos dentro de un FrameLayout un ImageView y un Button luego el botón se superpone a la imagen.

Una actividad posible del control FrameLayout es disponer una serie de controles visuales no visibles e ir alternando cual se hace visible.

1.6.4.1 Problema:

Disponer un ImageView y un Button dentro de un layout FrameLayout. Cuando se inicia la aplicación mostrar solo el botón y al ser presionado ocultar el botón y hacer visible la imagen que muestra el ImageView.

1.6.5 Layout (ScrollView y LinearLayout)

El ScrollView junto con un LinearLayout nos permite disponer una cantidad de componentes visuales que superan la cantidad de espacio del visor del celular o tablet. Luego el usuario puede desplazar con el dedo la interfaz creada.

1.6.5.1 Problema:

Crear un proyecto y disponer un control de tipo ScrollView (que se encuentra en la pestaña "Containers") como podemos controlar luego veremos que nuestro Scroll View tiene dentro un control "Linear Layout".

1.6.6 Icono de la aplicación

Cuando creamos un proyecto para implementar una aplicación con el entorno de desarrollo Android Studio, éste nos crea un ícono por defecto:

Los íconos e imágenes se almacenan en la carpeta res, hay cinco carpetas llamadas:

mipmap-mdpi

mipmap-hdpi

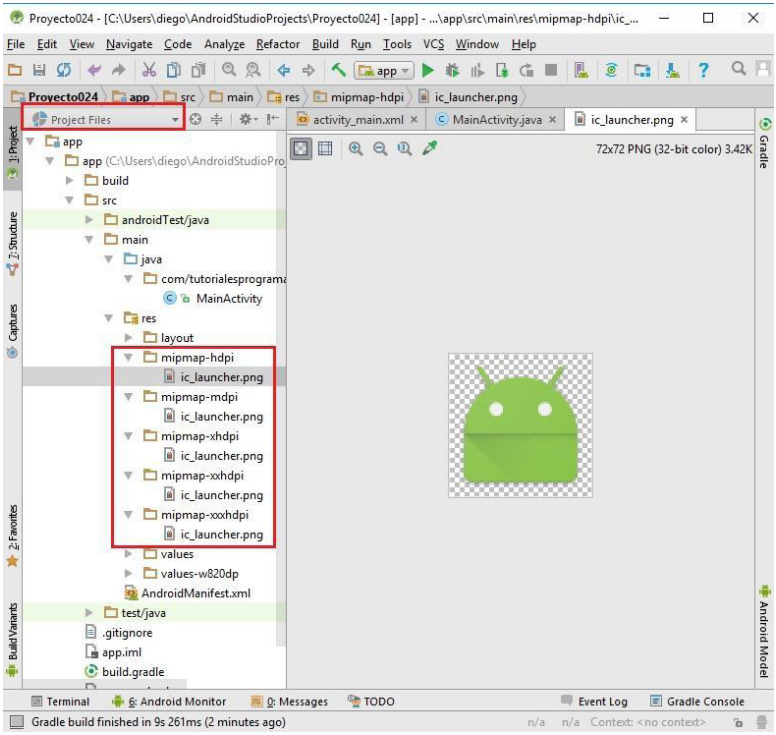
mipmap-xhdpi

mipmap-xxhdpi

mipmap-xxxhdpi

Y en cada una de estas hay un archivo llamado ic_launcher.png (para ver las carpetas podemos seleccionar "Project Files"):

Figura 52. íconos e imágenes en la carpeta res



Fuente: Propia

Como las resoluciones de los dispositivos Android pueden ser muy distintos (un celular, una tablet, un televisor etc.) se recomienda proporcionar múltiples copias de cada imagen de recursos a diferentes resoluciones y almacenarlos en las carpetas nombradas respetando las siguientes reglas:

res/mipmap-mdpi/

El ícono debe ser de 48*48 píxeles.

res/mipmap-hdpi/

150% del tamaño de las imágenes almacenadas en la carpeta mipmap-mdpi

El ícono debe ser de 72*72 píxeles.

res/mipmap-xhdpi/

200% del tamaño de las imágenes almacenadas en la carpeta mipmap-mdpi

El ícono debe ser de 96*96 píxeles.

res/mipmap-xxhdpi/

300% del tamaño de las imágenes almacenadas en la carpeta mipmap-mdpi

El ícono debe ser de 144*144 píxeles.

res/mipmap-xxxhdpi/

400% del tamaño de las imágenes almacenadas en la carpeta mipmap-mdpi

El ícono debe ser de 192*192 píxeles.

1.6.6.1 Problema

Crear una aplicación, dibujar y almacenar cinco archivos llamados ic_launcher.png (borrar los actuales). Tener en cuenta que el archivo que se almacena en la carpeta mipmap-mdpi debe ser de 48 píxeles, el de la carpeta mipmap-hdpi debe ser de 72 píxeles de ancho y alto, el de la carpeta mipmap-xhdpi debe ser de 96x96 píxeles, el de la carpeta mipmap-xxhdpi debe ser de 144*144 píxeles y finalmente el de la carpeta mipmap-xxxdpi debe ser de 192x192 pixeles.

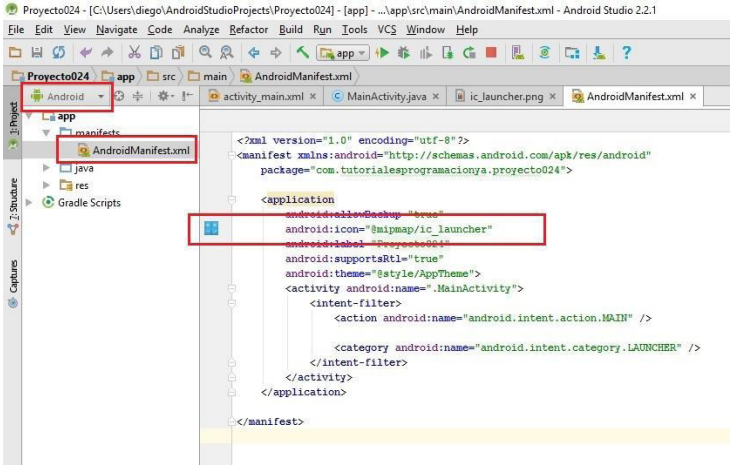
Ejecutar la aplicación y ver el ícono nuevo.

Figura 53. Vista icono de una aplicación

Fuente: Propia

En el archivo AndroidManifest.xml es donde indicamos el nombre del ícono de la aplicación:

Figura 54. Nombre del icono en AndroidManifest.xml



Fuente: Propia

Como vemos tenemos el nombre del archivo ic_launcher (no debemos indicar extensión)

Importante

Los nombres de archivos solo pueden tener caracteres en minúsculas, números y el guión bajo, cualquier otro caracter generará un error cuando tratemos de ejecutar la aplicación. Si bien podemos utilizar números dentro del nombre del archivo no puede ubicarse un número como primer caracter del nombre del archivo (esto es debido a que el Android Studio genera un archivo de recursos y define una variable con dicho nombre del archivo)

1.7 View y sus características

1.7.1 Componente ActionBar (Básica)

A partir de la versión 4.0 se introduce para el desarrollo de aplicaciones para celulares Android la componente ActionBar.

La componente ActionBar es la barra que aparece en la parte superior de gran cantidad de aplicaciones. Esta componente busca estandarizar entre distintas aplicaciones desarrolladas para Android la ubicación de botones de acciones, menú de opciones etc.

En el lado izquierdo aparece el nombre de la aplicación y del lado derecha aparece un botón que al ser presionado nos muestra un menú desplegable:

Figura 55. Vista Action Bar

Fuente: Propia

1.7.1.1 Problema:

Confeccionar una aplicación que muestre en el ActionBar el título "Este mundo" y luego tres opciones en el menú desplegable del ActionBar.

La interfaz del ActionBar y el menú desplegable debe ser similar a:

Figura 56. Paso 1 Herramienta Action Bar

Fuente: Propia

Crear un proyecto:

Veamos los pasos que debemos dar para obtener este resultado:

El título del ActionBar lo debemos modificar abriendo el archivo strings que se encuentra en la carpeta res/values y su contenido por defecto al crear el proyecto es:

```
<resources>
    <string name="app_name">Proyecto047</string>
</resources>
```

Podemos cambiar el contenido de la constante "app_name":

```
<resources>
    <string name="app_name">Este mundo</string>
</resources>
```

Recordemos que este archivo de recursos se utiliza para agrupar todos los mensajes que aparecen en pantalla y facilitar la implementación de aplicaciones para varios idiomas independiente de los algoritmos de la misma.

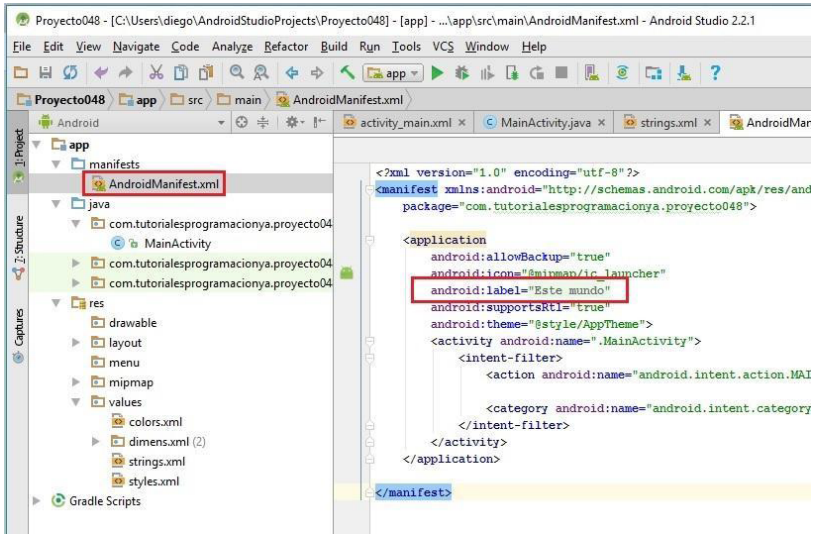
En este archivo se crea un string llamado "app_name" con el valor "Este mundo".
Luego este string se asocia con el título de la aplicación con la propiedad label en el archivo AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.tutorialesprogramacionya.proyecto047">


  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
```

```
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER"
        />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Desde el Editor del Android Studio cuando vemos el archivo AndroidManifest.xml nos sustituye visualmente las constantes por los valores de las constantes para ver que realmente se almacena:

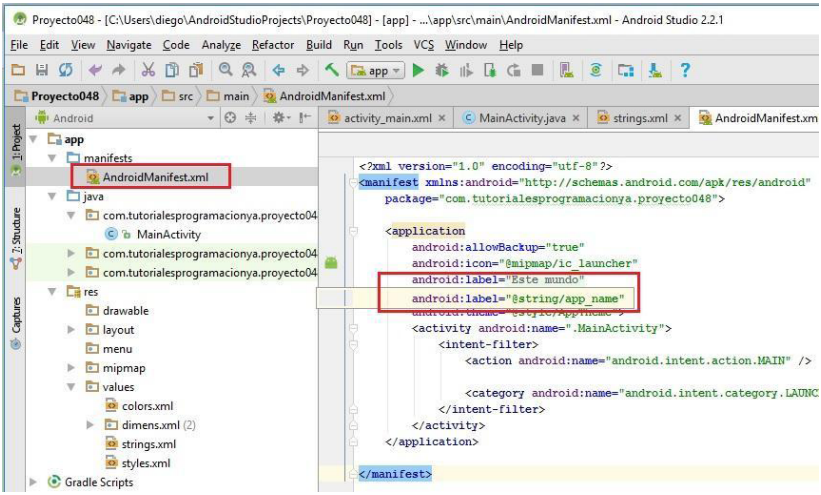
Figura 57. Paso 2 Herramienta Action Bar



Fuente: Propia

Pero si posicionamos el mouse sobre el valor veremos que está definida una constante con la sintaxis @string/[nombre de constante]:

Figura 58. Paso 3 Herramienta Action Bar



Fuente: Propia

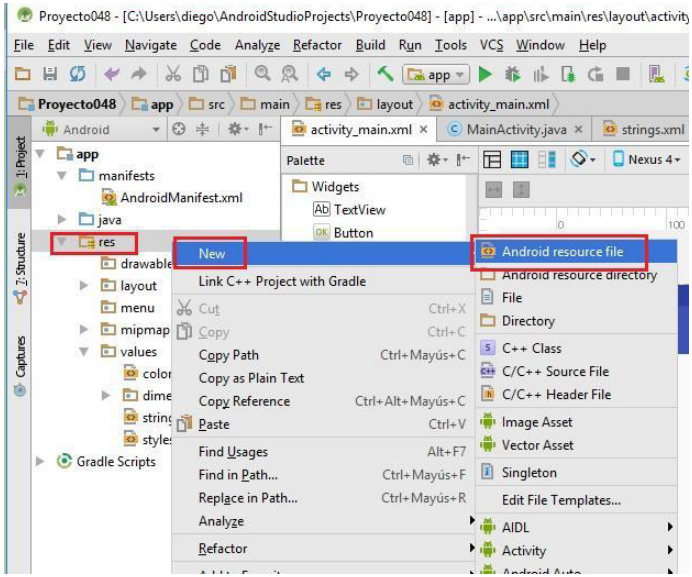
Y si hacemos clic veremos el contenido real del valor con el que se inicializa la propiedad.

En este archivo XML ya está inicializada la propiedad label de la marca application con el valor definido en el archivo xml (como vemos acá hacemos referencia al string app_name):

android:label=”@string/app_name”

Ahora tenemos que definir las opciones de nuestro menú desplegable, debemos crear un archivo XML con las opciones, para esto presionamos el botón derecho sobre la carpeta res y seleccionamos la opción New -> Android resource file:

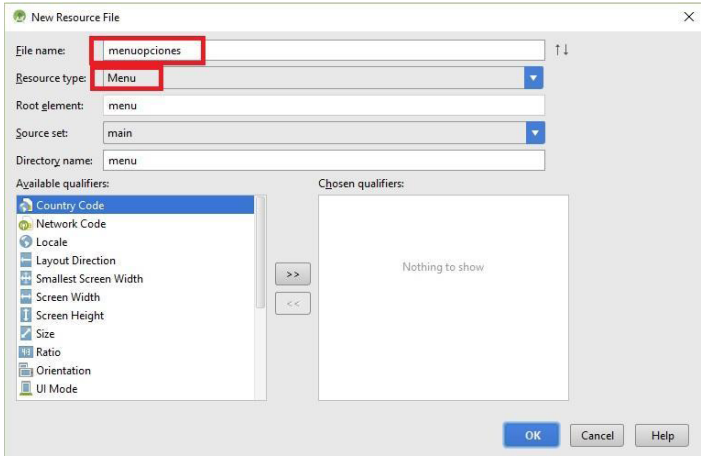
Figura 59. Paso 4 Herramienta Action Bar



Fuente: Propia

En este diálogo indicamos el nombre del archivo a crear “menuopciones” y el tipo de recurso (Resource type) de tipo Menu:

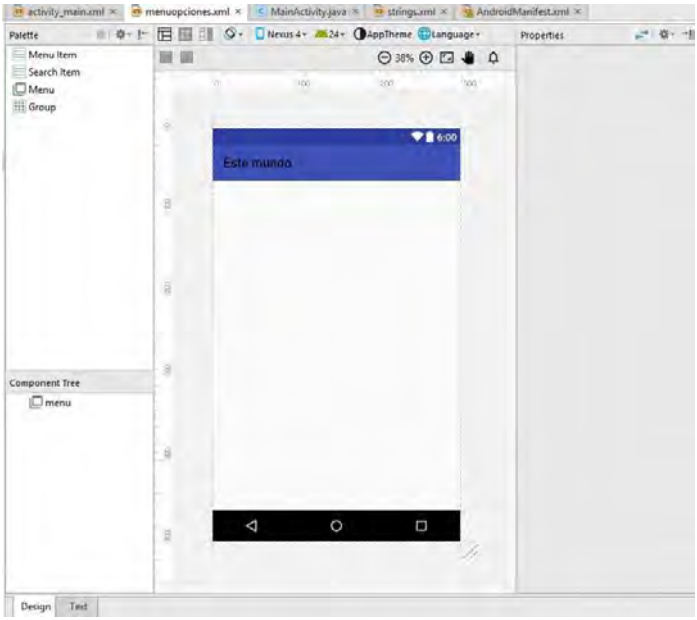
Figura 60. Paso 5 Herramienta Action Bar



Fuente: Propia

Aparece ahora una herramienta que nos permite crear el menú de opciones en forma visual (Esta herramienta está disponible desde el Android Studio 2.3.x):

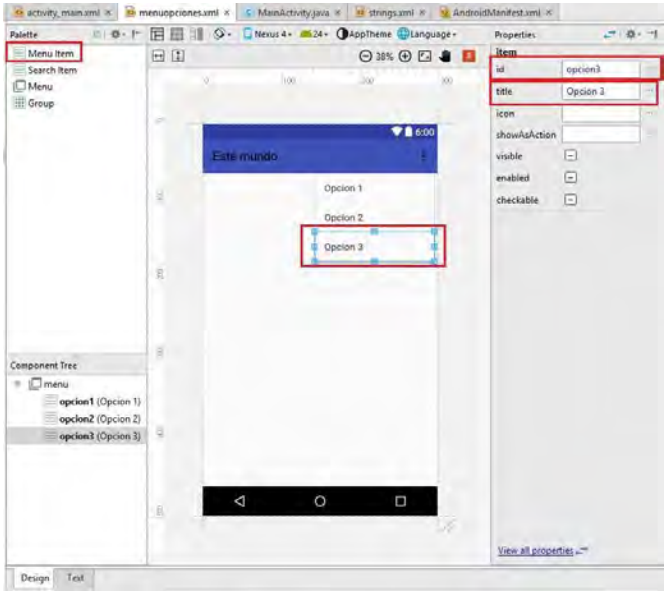
Figura 61. Paso 6 Herramienta Action Bar



Fuente: Propia

Arrastramos Menu Item a nuestra ventana y configuramos las propiedades:

Figura 62. Paso 7 Herramienta Action Bar



Fuente: Propia

Podemos ver el archivo menuopciones.xml como texto seleccionando la opción "Text" que aparece en la parte inferior:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:android="http://schemas.android.com/apk/res/android">

  <item android:title="Opcion 1"
    android:id="@+id/opcion1" />
  <item android:title="Opcion 2"
    android:id="@+id/opcion2" />
  <item android:title="Opcion 3"
    android:id="@+id/opcion3" />
</menu>
```

Versiones anteriores de Android Studio requería que se escribiera directamente el xml.

1. La funcionalidad de nuestro programa será mostrar un Toast cuando se seleccione alguno de las opciones del menú. El código java de la clase debe ser:

```
package com.tutorialesprogramacionya.proyecto047;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {


  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menuopciones, menu);
    return true;
  }


  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id==R.id.opcion1) {
      Toast.makeText(this,"Se seleccionó la primer opción",Toast.LENG-
      TH_LONG).show();
    }
    if (id==R.id.opcion2) {
      Toast.makeText(this,"Se seleccionó la segunda opción",Toast.LEN-
      GTH_LONG).show();
```

```
    }
        if (id==R.id.opcion3) {
        Toast.makeText(this,"Se seleccionó la tercer opción", Toast.LENG-
        TH_LONG).show();
        }
        return super.onOptionsItemSelected(item);
    }
}
```

Como podemos observar hemos sobreescrito los métodos onCreateOptionsMenu y onOptionsItemSelected:

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menuopciones, menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
        if (id==R.id.opcion1) {
            Toast.makeText(this,"Se seleccionó la primer opción",Toast.LENG-
            TH_LONG).show();
        }
        if (id==R.id.opcion2) {
            Toast.makeText(this,"Se seleccionó la segunda opción",Toast.LEN-
            GTH_LONG).show();
        }
        if (id==R.id.opcion3) {
            Toast.makeText(this,"Se seleccionó la tercer opción",Toast.LENG-
            TH_LONG).show();
        }
        return super.onOptionsItemSelected(item);
    }
```

El método onCreateOptionsMenu lee el archivo menuopciones.xml donde definimos el menú de opciones:

1.7.1.2 Problema:

Confeccionar el mismo problema del ActionBar con el menú de opciones pero utilizando constantes para cada opción del menú y también constantes para los tres mensajes que aparecen cuando se seleccionan cada una de las opciones.

1.7.2 Componente ActionBar (Botones de acción)

Ahora veremos otra característica que podemos agregarle a la barra de acción (ActionBar). Como habíamos visto la barra de acción muestra a la izquierda un título y del lado derecho un botón que muestra un menú desplegable. Podemos disponer opciones de dicho menú que se muestren directamente en la barra de acción para que tengamos acceso a dichas opciones en forma directo sin tener que abrir el menú desplegable.

1.7.2.1 Problema:

Confeccionar una aplicación que muestre como título "ActionBar" y luego dos botones de acción y tres opciones en el menú desplegable del ActionBar.

La interfaz del ActionBar, el título, los botones de acción y el menú desplegable debe ser similar a:

Figura 63. Componente ActionBar (Botones de acción)

Fuente: Propia

1.7.2.2 Problema Propuesto

Volver a codificar el problema anterior para mostrar dos botones y tres opciones en el menú del ActionBar localizando todos los mensajes en el archivo strings.xml

1.7.3 Componente ActionBar (Ocultarlo y mostrarlo)

La barra de acción se muestra en la parte superior para acceder a las opciones de nuestro programa, pero hay situaciones donde necesitemos ocultarla para tener más espacio. Para ocultar y mostrar el ActionBar disponemos de dos métodos:

show()
hide()

1.7.3.1 Problema:

Confeccionar una aplicación que mediante dos botones se permita ocultar el ActionBar o mostrarlo. Disponer tres opciones en el menú desplegable del ActionBar.

1.8 List Item, Content Providers

1.8.1 Componente ListView (con un ImageView y un TextView)

En uno de los primeros conceptos vimos como utilizar el control ListView para mostrar una lista de String y poder seleccionar uno de ellos.

Vimos que era bastante sencillo implementar la lista ya que cada elemento de la lista es un único TextView (String) y el API de Android ya trae todo preconfigurado para implementar este tipo de ListView.

Ahora veremos los pasos para poder implementar un ListView que muestre en cada item de la lista un objeto de la clase ImageView y un TextView.

1.8.1.1 Problema:

Confeccionar una aplicación que muestre una lista de nombres de personas y mediante una imagen mostrar si es un hombre o una mujer.

La interfaz visual de la aplicación a implementar debe ser similar a esta:

Figura 64. Componente ListView

Fuente: Propia

1.8.2 Agregar y eliminar elementos de un ListView

Una situación muy común es tener que agregar elementos a un ListView en forma dinámica (es decir durante la ejecución de la aplicación), de forma similar podemos necesitar eliminar elementos.

1.8.2.1 Problema:

Confeccionar una aplicación que muestre un ListView con números telefónicos y sus titulares. Permitir agregar y eliminar números durante la ejecución del programa.

Para agregar presionar un botón y para eliminarlo hacer una presión del item larga con el dedo y mostrar un diálogo para confirmar el borrado.

Componente ListView (grabar sus datos con la clase SharedPreferences)

Como sabemos cada vez que iniciamos una aplicación que utiliza un ListView debemos cargar sus elementos. En el concepto anterior vimos que podemos agregar elementos en forma dinámica. Ahora veremos como almacenan los datos mediante la clase SharedPreferences (vimos en un concepto anterior que esta clase nos facilita guardar datos en un archivo XML mediante una clave,valor)

En el momento que inicia la aplicación procederemos a leer el contenido del archivo de preferencias y a añadir los datos al ListView. Durante la ejecución del programa cuando se agreguen o eliminen elementos del ListView también procederemos a hacerlo del archivo de preferencias.

Problema:

Confeccionaremos una aplicación similar al concepto anterior que muestre un ListView con números telefónicos y sus titulares, que permita agregar y eliminar números durante la ejecución del programa. Le agregaremos lo necesario para que los datos reflejados en el ListView se almacenen en un archivo de preferencias para evitar que se borren cuando se finaliza la aplicación.

Cada vez que arranque el programa procederemos a leer el archivo de preferencias y poblaremos el ListView.

1.9 Tipos de Alertas y Notificaciones

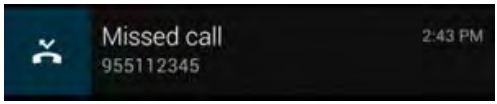
1.9.1 Notificaciones en Android (II): Barra de Estado

Hace algún tiempo, ya tratamos dentro de este curso un primer mecanismo de notificaciones disponibles en la plataforma Android: los llamados Toast. Como ya comentamos, este tipo de notificaciones, aunque resultan útiles y prácticas en muchas ocasiones, no deberíamos utilizarlas en situaciones en las que necesitemos asegurarnos la atención del usuario ya que no requiere de ninguna intervención por su parte, se muestran y desaparecen automáticamente de la pantalla.

En este nuevo artículo vamos a tratar otro tipo de notificaciones algo más persistentes, las notificaciones de la barra de estado de Android. Estas notificaciones son las que se muestran en nuestro dispositivo por ejemplo cuando recibimos un mensaje SMS, cuando tenemos actualizaciones disponibles, cuando tenemos el reproductor de música abierto en segundo plano, … Estas notificaciones constan de un icono y un texto mostrado en la barra de estado superior, y adicionalmente un mensaje algo más descriptivo y una marca de fecha/hora que podemos consultar desplegando la bandeja del sistema. A modo de ejemplo, cuando tenemos una llamada perdida en nuestro terminal, se nos muestra por un lado un icono en la barra de estado superior (más un texto que aparece durante unos segundos).

… y un mensaje con más información al desplegar la bandeja del sistema, donde en este caso concreto se nos informa del evento que se ha producido («Missed call«), el número de teléfono asociado, y la fecha/hora del evento. Además, al pulsar sobre la

notificación se nos dirige automáticamente al historial de llamadas.



Pues bien, aprendamos a utilizar este tipo de notificaciones en nuestras aplicaciones.

1.9.1.1 Problema:

Vamos a construir para ello una aplicación de ejemplo, como siempre lo más sencilla posible para centrar la atención en lo realmente importante. En este caso, el ejemplo va a consistir en un único botón que genere una notificación de ejemplo en la barra de estado, con todos los elementos comentados y con la posibilidad de dirigirnos a la propia aplicación de ejemplo cuando se pulse sobre ella.

1.9.2 Notificaciones en Android (III): Diálogos

Durante este curso ya hemos visto dos de las principales alternativas a la hora de mostrar notificaciones a los usuarios de nuestra aplicaciones: los toast y las notificaciones de la barra de estado. En este último artículo sobre notificaciones vamos a comentar otro mecanismo que podemos utilizar para mostrar o solicitar información puntual al usuario. El mecanismo del que hablamos son los cuadros de diálogo.

En principio, los diálogos de Android los podremos utilizar con distintos fines, en general:

- Mostrar un mensaje.
- Pedir una confirmación rápida.

- Solicitar al usuario una elección (simple o múltiple) entre varias alternativas.

De cualquier forma, veremos también cómo personalizar completamente un diálogo para adaptarlo a cualquier otra necesidad.

 1.9.3 Notificaciones en Android (IV): Snackbar

Un nuevo tipo de notificación, que ha tomado especial relevancia sobre todo a raíz de la aparición de Android 5 Lollipop y *Material Design*, son los llamados *snackbar*. Un snackbar debe utilizarse para mostrar feedback sobre alguna operación realizada por el usuario, y es similar a un *toast* en el sentido de que aparece en pantalla por un corto periodo de tiempo y después desaparece automáticamente, aunque también presenta algunas diferencias importantes, como por ejemplo que puede contener un botón de texto de acción.

1.9.4 Evaluación 2

1. En Android cual es la función del método setOnItem-ClicListener de la clase ListView
2. ¿Cual clase en Android permite mostrar una ventana emergente temporal que informa al usuario mediante un mensaje que aparece en la pantalla por un lapso pequeño de tiempo?
3. ¿Cuál es el tipo de EditText que nos ofrece Android para utilizar en nuestras aplicaciones?
4. ¿Cuál es la clase en Android que me permite lanzar la ventana de un segundo Activity?
5. ¿Qué método de la clase Intent me permite enviar datos de la primera ventana a la segunda para que a partir de estos proceda a efectuar una acción?
6. ¿Cuál es el archivo de manifiesto describe información esencial como el nombre del paquete de la aplicación, los componentes de la aplicación, los permisos que necesita la aplicación para acceder a las partes protegidas del sistema o a otras aplicaciones?
7. ¿Cuál es la clase que encapsula todo lo relacionado a pintar píxeles, líneas, rectángulos?
8. En la clase Paint llamamos al método setARGB para definir el color del pincel el primer parámetro indica el valor de transparencia. ¿Cuál de las siguientes configuraciones indicamos que no hay transparencia?
9. ¿Cuál es método de la clase Canvas donde indicamos la cantidad de rojo, verde a azul?

10. ¿Cuál es método de la clase Canvas que permite mostrar un archivo jpg, png, etc?
11. ¿Cuál es el permiso que debemos configurar en el archivo "AndroidManifest.xml" para que nuestra aplicación pueda acceder a Internet?
12. La plataforma de Android nos da varias facilidades para el almacenamiento permanente de datos dentro de la aplicación. ¿Cuál de las siguientes opciones no es uno de estos métodos?
13. Cuando guardamos datos en el archivo de preferencias de la clase SharedPreferences. ¿Cuál es el método que podemos usar para almacenar el siguiente dato ("activo", true)?
14. Para leer los bytes o datos de un archivo de texto, cual es la clase que se debe usar:
15. ¿Cómo se llama la herramienta nativa de Android para almacenar datos en una base de datos?
16. ¿Cuál es la clase auxiliar para gestionar la creación de bases de datos nativa y la gestión de versiones en Android?

Capítulo 2
Interfaces de audio y video

2.1. Interfaces de audio y video

2.1.1 Reproducción de audio (archivo contenido en la aplicación)

Veremos los pasos para reproducir un archivo mp3 (otros formatos soportados por Android son: Ogg, Wav)

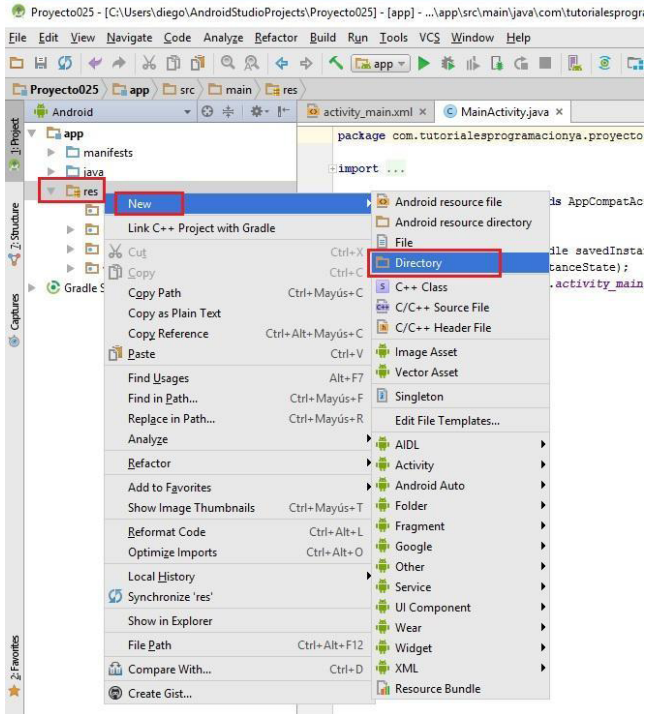
2.1.1.1 Problema:

Primero crear un nuevo proyecto

Disponer dos botones con las etiquetas: Gato y León, luego cuando se presione reproducir el archivo de audio respectivo. Los archivos de sonidos almacenarlos en la misma aplicación.

Luego de crear el proyecto procedemos a crear una carpeta llamada raw que dependa de la carpeta res, almacenamos los dos archivos mp3 en dicha carpeta (para crear la carpeta presionamos el botón derecho del mouse sobre la carpeta res y seleccionamos New -> Directory):

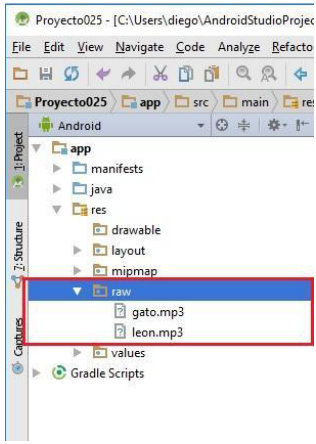
Figura 65. Paso 1 uso de Media Player



Fuente: Propia

Luego copiamos los archivos a la carpeta (en Android Studio funciona el Copy/Paste desde el administrador de archivos del sistema operativo Windows):

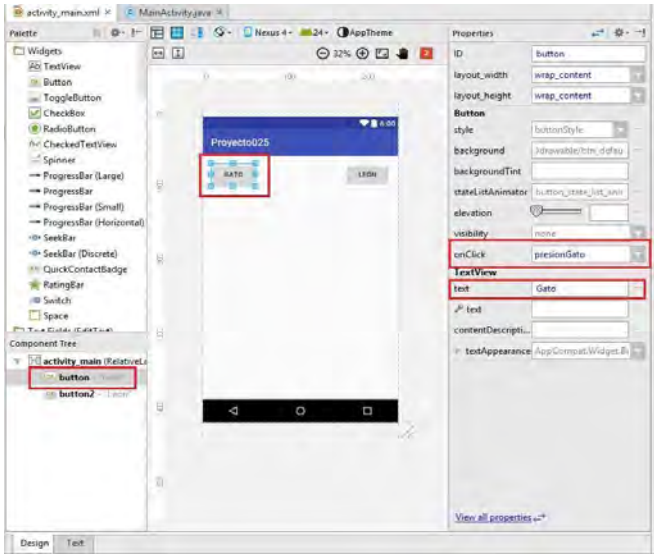
Figura 66. Paso 2 uso de Media Player



Fuente: Propia

Creamos una interfaz con dos botones e inicializamos las propiedades text y onClick de cada botón:

Figura 67. Paso 3 uso de Media Player



Fuente: Propia

El código fuente es:

```
package com.tutorialesprogramacionya.proyecto025;
import android.media.MediaPlayer;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
  public void presionGato(View v) {
    MediaPlayer mp = MediaPlayer.create(this, R.raw.gato);
    mp.start();
  }
  public void presionLeon(View v) {
    MediaPlayer mp = MediaPlayer.create(this, R.raw.leon);
    mp.start();
  }
}
```

Cuando copiamos los archivos mp3 se genera luego en la clase R la referencia a los dos archivos y posteriormente los podemos rescatar cuando creamos un objeto de la clase MediaPlayer:

```
MediaPlayer mp=MediaPlayer.create(this,R.raw.gato);
```

Seguidamente llamamos al método start:

```
mp.start();
```

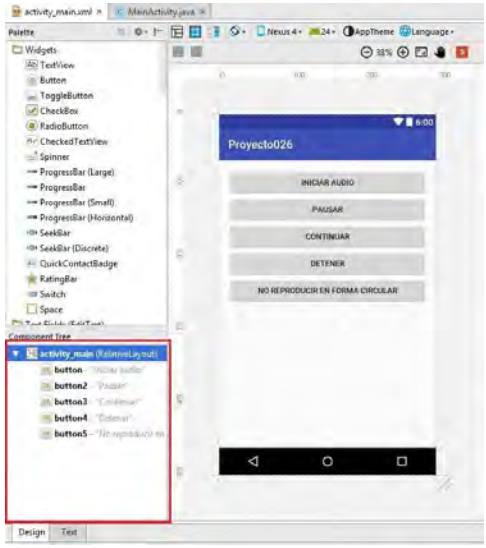
2.1.2 Reproducción, pausa, continuación y detención de un archivo de audio

Confeccionar una aplicación que permita Iniciar un archivo mp3, detener, continuar, detener en forma definitiva y activación o no de la reproducción en forma circular.

Crear un archivo mp3 con el programa Audacity contando del 1 al 30.

Primero creamos un proyecto y definimos los 5 botones y métodos a ejecutar cuando se presionen los botones respectivos:

Figura 68. Paso 4 uso de Media Player



Fuente: Propia

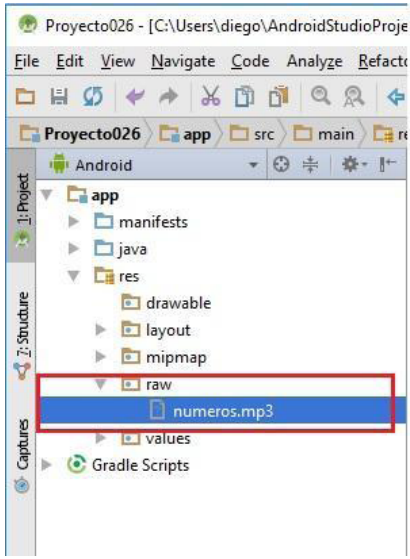
Iniciamos la propiedad onClick de cada botón:

button (onClick=”iniciar”)
button2 (onClick=”pausar”)

button3 (onClick=”continuar”)
button4 (onClick=”detener”)
button5 (onClick=”circular”)

Creamos la carpeta raw y almacenamos en la misma el archivo mp3 creado previamente:

Figura 69. Paso 5 uso de Media Player



Fuente: Propia

El código fuente es:

```
package com.tutorialesprogramacionya.proyecto026;
import android.media.MediaPlayer;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
```

```
MediaPlayer mp;
Button b5;
int posicion = 0;

@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   b5=(Button)findViewById(R.id.button5);
   Toast.makeText(this,b5.getText().toString(),Toast.LENGTH_
   LONG).show();
}
public void destruir() {
   if (mp != null)
      mp.release();
}
public void iniciar(View v) {
   destruir();
   mp = MediaPlayer.create(this, R.raw.numeros);
   mp.start();
   String op = b5.getText().toString();
   if (op.equals("no reproducir en forma circular"))
      mp.setLooping(false);
   else
      mp.setLooping(true);
}
public void pausar(View v) {
   if (mp != null && mp.isPlaying()) {
      posicion = mp.getCurrentPosition();
      mp.pause();
   }
}
public void continuar(View v) {
   if (mp != null && mp.isPlaying() == false) {
      mp.seekTo(posicion);
      mp.start();
```

```
        }
    }
    public void detener(View v) {
        if (mp != null) {
            mp.stop();
            posicion = 0;
    }
  }
  public void circular(View v) {
    detener(null);
    String op = b5.getText().toString();
    if (op.equals("no reproducir en forma circular"))
      b5.setText("reproducir en forma circular");
    else
      b5.setText("no reproducir en forma circular");
  }
}
```

Primero definimos tres atributos uno de la clase MediaPlayer para administrar el archivo mp3, un entero donde se almacena la posición actual de reproducción en milisegundos (para poder continuarla en el futuro) y la referencia de un objeto de la clase Button:

```
MediaPlayer mp;
Button b5;
int posicion = 0;
```

El método destruir verifica con un if si el objeto de la clase MediaPlayer está creado procede a liberar recursos del mismo llamando al método release:

```
public void destruir() {
  if(mp!=null)
    mp.release();
}
```

El método iniciar que se ejecuta al presionar el botón "iniciar" primero llama al método destruir (para el caso que el mp3 este en ejecución actualmente) seguidamente creamos un objeto de la clase MediaPlayer llamando al método create (en este hacemos referencia al archivo que copiamos a la carpeta raw) Llamamos al método start. Por último extraemos el texto del quinto botón y verificamos si la reproducción debe ejecutarse en forma circular (en forma indefinida una y otra vez):

```
public void iniciar(View v) {
   destruir();
   mp = MediaPlayer.create(this,R.raw.numeros);
   mp.start();
   String op=b5.getText().toString();
   if (op.equals("no reproducir en forma circular"))
      mp.setLooping(false);
   else
      mp.setLooping(true);
}
```

El método pausar verifica que el objeto de la clase MediaPlayer este creado y en ejecución, en caso afirmativo recuperamos la posición actual de reproducción y llamamos seguidamente al método pause:

```
public void pausar(View v) {
   if(mp != null && mp.isPlaying()) {
      posicion = mp.getCurrentPosition();
      mp.pause();
   }
}
```

El método continuar verifica que el objeto de la clase MediaPlayer este creado y la propiedad isPlaying retorne false para proceder a posicionar en que milisegundo continuar la reproducción:

```
public void continuar(View v) {
   if(mp != null && mp.isPlaying()==false) {
      mp.seekTo(posicion);
      mp.start();
   }
}
```

El método detener interrumpe la ejecución del mp3 e inicializa el atributo posicion con cero:

```
public void detener(View v) {
   if(mp != null) {
      mp.stop();
      posicion = 0;
   }
}
```

Cuando se presiona el botón que cambia si la reproducción se efectúa en forma circular o no procedemos a extraer su texto y según dicho valor almacenamos el valor opuesto:

```
public void circular(View v) {
   detener(null);
   String op=b5.getText().toString();
   if (op.equals("no reproducir en forma circular"))
      b5.setText("reproducir en forma circular");
   else
      b5.setText("no reproducir en forma circular");
}
```

2.1.3 Reproducción de audio (archivo localizado en internet)

Ahora vamos a ver los pasos para reproducir un archivo almacenado en un servidor de internet.

2.1.3.1 Problema:

Disponer un botón con la etiqueta: "gato", luego cuando se presione reproducir el archivo de audio respectivo. El archivo de sonido se encuentra almacenado en http://www.javaya.com.ar/recursos/gato.mp3

2.1.4 Reproducción de audio utilizando el reproductor propio de Android (vía Intent)

Otra forma de ejecutar un archivo mp3 es mediante el reproductor interno de Android. Esta aplicación reproduce todos los formatos soportados por Android y tiene una interfaz que le será familiar al usuario de nuestra aplicación.

2.1.4.1 Problema:

Disponer un botón con la etiqueta: "ejecutar mp3 con el reproductor propio de android", luego cuando se presione reproducir el archivo de audio respectivo con el reproductor de Android via Intent. El archivo de sonido almacenarlo en la tarjeta SD, utilizaremos el archivo "gato.mp3" que lo subimos a la tarjeta SD en conceptos anteriores.

2.1.5 Grabación de audio mediante el grabador provisto por Android (via Intent)

La forma más sencilla de capturar audio en Android es mediante el grabador que provee el sistema operativo Android. Invocamos la aplicación de grabación y luego recuperamos el audio grabado.

Tiene como ventaja que la interfaz le es familiar al usuario, ya que muchas aplicaciones utilizan esta característica.

2.1.5.1 Problema:

Disponer dos objetos de la clase Button con las etiquetas "grabar" y "reproducir". Cuando se presione el primer botón proceder a activar la grabadora provista por Android. Cuando se presione el segundo botón reproducir el audio grabado.

2.1.6 Captura de audio mediante la clase MediaRecorder

Otra forma de grabar audio en Android es el empleo de la clase MediaRecorder. Esta clase nos da más libertades a la hora de construir una aplicación que requiere grabar audio.

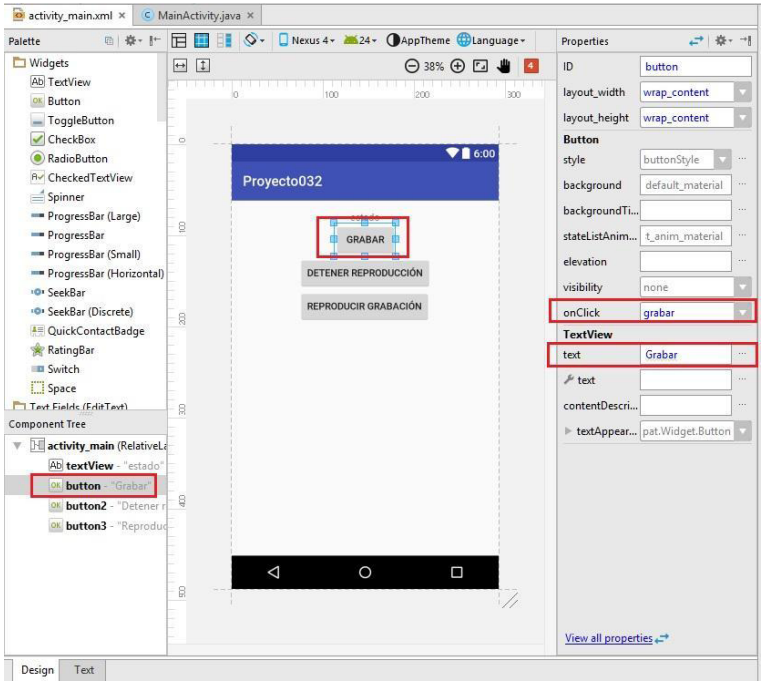
2.1.6.1 Problema:

Crear un proyecto, disponer tres objetos de la clase Button con las etiquetas "Grabar", "Detener Grabación" y "Reproducir Grabación". Disponer además un TextView para informar del estado actual.

Cuando se presione el botón "Grabar" permitir registrar todos los sonidos hasta que se presione el botón "Detener Grabación". Cuando se presione el botón "Reproducir Grabación" emitir el archivo de audio previamente generado.

La interfaz visual a implementar es la siguiente:

Figura 70. Paso 1 Uso del Media Recorder



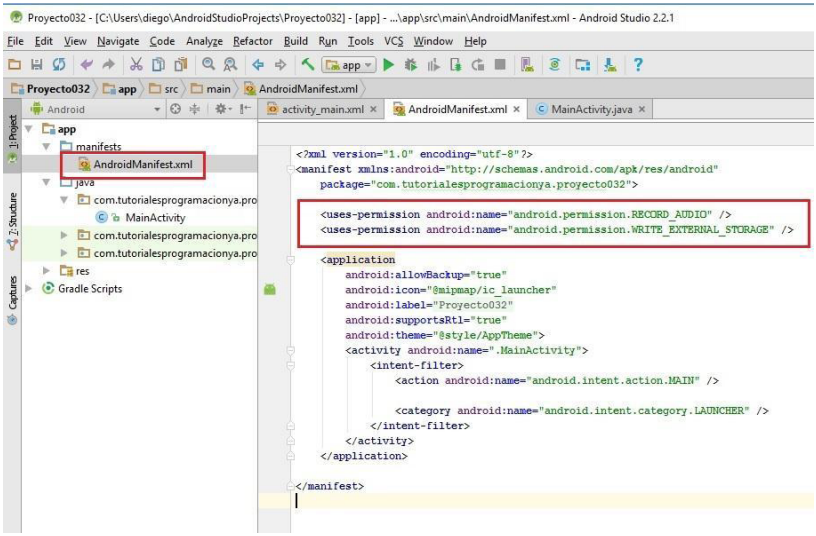
Fuente: Propia

Tener en cuenta de no olvidar definir los tres métodos para los tres botones: grabar, detener y reproducir en las propiedades onClick de cada botón.

También debemos modificar el archivo AndroidManifest.xml donde debemos indicar que nuestra aplicación accederá a la grabadora de sonido y a la tarjeta SD donde se almacenará el archivo de sonido.

Esto lo hacemos seleccionando el archivo AndroidManifest.xml y escribiendo los permisos respectivos:

Figura 71. Paso 2 Uso del Media Recorder



Fuente: Propia

El código fuente es:

```
package com.tutorialesprogramacionya.proyecto032;
import android.media.MediaPlayer;
import android.media.MediaRecorder;
import android.os.Environment;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import java.io.File;
import java.io.IOException;
public class MainActivity extends AppCompatActivity implements MediaPlayer.OnCompletionListener{
   TextView tv1;
   MediaRecorder recorder;
```

```
MediaPlayer player;
File archivo;
Button b1, b2, b3;
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   tv1 = (TextView) this.findViewById(R.id.textView);
   b1 = (Button) findViewById(R.id.button);
   b2 = (Button) findViewById(R.id.button2);
   b3 = (Button) findViewById(R.id.button3);
}


public void grabar(View v) {
   recorder = new MediaRecorder();
   recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
   recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_
   GPP);
   recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_
   NB);
   File path = new File(Environment.getExternalStorageDirectory()
      .getPath());
   try {
      archivo = File.createTempFile("temporal", ".3gp", path);
   } catch (IOException e) {
   }
   recorder.setOutputFile(archivo.getAbsolutePath());
   try {
      recorder.prepare();
   } catch (IOException e) {
   }
   recorder.start();
   tv1.setText("Grabando");
   b1.setEnabled(false);
```

```
        b2.setEnabled(true);
    }

    public void detener(View v) {
        recorder.stop();
        recorder.release();
        player = new MediaPlayer();
        player.setOnCompletionListener(this);
        try {
            player.setDataSource(archivo.getAbsolutePath());
        } catch (IOException e) {
        }
        try {
            player.prepare();
        } catch (IOException e) {
        }
        b1.setEnabled(true);
        b2.setEnabled(false);
        b3.setEnabled(true);
        tv1.setText("Listo para reproducir");
    }
    public void reproducir(View v) {
        player.start();
        b1.setEnabled(false);
        b2.setEnabled(false);
        b3.setEnabled(false);
        tv1.setText("Reproduciendo");
    }
    public void onCompletion(MediaPlayer mp) {
        b1.setEnabled(true);
        b2.setEnabled(true);
        b3.setEnabled(true);
        tv1.setText("Listo");
    }
}
```

Declaramos un objeto de la clase MediaRecorder para grabar audio:

MediaRecorder recorder;

Declaramos un objeto de la clase MediaPlayer para reproducir el archivo de sonido generado:

MediaPlayer player;

Declaramos un objeto de la clase File que hace referencia al archivo que se creará:

File archivo;

Declaramos las variables que harán referencia a los tres botones y al TextView:

TextView tv1;

Button b1,b2,b3;

En el método onCreate obtenemos la referencia de los cuatro objetos creados en el archivo XML:

tv1 = (TextView) this.findViewById(R.id.textView);

b1 = (Button) findViewById(R.id.button);

b2 = (Button) findViewById(R.id.button2);

b3 = (Button) findViewById(R.id.button3);

El método más importante de este concepto es el grabar:

```
public void grabar(View v) {
  recorder = new MediaRecorder();
  recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
  recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_
  GPP);
  recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_
```

```
    NB);
    File path = new File(Environment.getExternalStorageDirectory().ge-
    tPath());
    try {
       archivo = File.createTempFile("temporal", ".3gp", path);
    } catch (IOException e) {
    }
    recorder.setOutputFile(archivo.getAbsolutePath());
    try {
       recorder.prepare();
    } catch (IOException e) {
    }
    recorder.start();
    tv1.setText("Grabando");
    b1.setEnabled(false);
    b2.setEnabled(true);
  }
```

Creamos un objeto de la clase MediaRecorder:

```
recorder = new MediaRecorder();
```

Seguidamente definimos el micrófono como fuente de audio:

```
recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
```

Luego llamamos al método setOutputFormat especificando que el archivo será almacenado con la especificación 3GPP y con extensión .3gp

```
recorder.setOutputFormat(MediaRecorder.OutputFormat.
THREE_GPP);
```

Especificamos el codec a emplear llamando al método setAudioEncoder:

```
recorder.setAudioEncoder(MediaRecorder.AudioEncoder.
AMR_NB);
```

Obtenemos el path de la tarjeta SD y creamos un archivo temporal con extensión 3gp:

```
File path = new File(Environment.getExternalStorageDirectory().get-
Path());
try {
   archivo = File.createTempFile("temporal", ".3gp", path);
} catch (IOException e) {
}
```

Con el método setOutputFile de la clase MediaRecorder le indicamos el archivo donde debe almacenarse la grabación:

```
recorder.setOutputFile(archivo.getAbsolutePath());
```

Llamamos al método prepare y finalmente al método start para comenzar la grabación:

```
try {
   recorder.prepare();
} catch (IOException e) {
}
recorder.start();
```

El método detener:

```
public void detener(View v) {
   recorder.stop();
   recorder.release();
   player = new MediaPlayer();
   player.setOnCompletionListener(this);
   try {
      player.setDataSource(archivo.getAbsolutePath());
   } catch (IOException e) {
   }
   try {
      player.prepare();
   } catch (IOException e) {
   }
   b1.setEnabled(true);
```

```
    b2.setEnabled(false);
    b3.setEnabled(true);
    tv1.setText("Listo para reproducir");
}
```

Llamamos primero al método stop de la clase MediaRecorder y liberamos los recursos consumidos llamando a release:

```
recorder.stop();
recorder.release();
```

Creamos un objeto de la clase MediaPlayer para poder reproducir el archivo de audio que acabamos de grabar. Indicamos mediante el método setOnCompletionListener la referencia de la clase que será informada cuando el audio finalice:

```
player = new MediaPlayer();
player.setOnCompletionListener(this);
```

Referenciamos el archivo a que debe reproducir:

```
try {
    player.setDataSource(archivo.getAbsolutePath());
} catch (IOException e) {
}
```

Finalmente llamamos al método prepare de la clase MediaPlayer:

```
try {
    player.prepare();
} catch (IOException e) {
}
```

El método reproducir simplemente llama al método start de la clase MediaPlayer para iniciar la reproducción del archivo previamente grabado:

```
public void reproducir(View v) {
   player.start();
   b1.setEnabled(false);
   b2.setEnabled(false);
   b3.setEnabled(false);
   tv1.setText("Reproduciendo");
}
```

El método onCompletion se ejecuta cuando termina de reproducirse el archivo de audio:

```
public void onCompletion(MediaPlayer mp) {
   b1.setEnabled(true);
   b2.setEnabled(true);
   b3.setEnabled(true);
   tv1.setText("Listo");
}
```

2.1.7 Tomar una foto y grabarla en un archivo (mediante Intent)

La forma más sencilla de capturar una foto en Android es mediante el software que provee el sistema operativo Android. Invocamos la aplicación de tomar fotos mediante la clase Intent.

Tiene como ventaja que la interfaz le es familiar al usuario, ya que muchas aplicaciones utilizan esta característica y nos facilita mucho la codificación.

2.1.7.1 Problema:

Confeccionaremos una aplicación que permita ingresar un nombre de archivo con extensión jpg en un EditText. Luego al presionar un botón lanzar el Activity que proporciona Android

para tomar una foto. Grabar el archivo en la memoria externa del dispositivo. Disponer un segundo botón para recuperar un archivo jpg almacenado en el dispositivo.

Finalmente crearemos un segundo Activity para mostrar en un ListView todos los nombres de archivos de imágenes almacenados por la aplicación y al presionar uno de ellos procederemos a mostrarlo en un ImageView.

2.1.8 Tomar un video y grabarlo en un archivo (mediante Intent)

La forma más sencilla de capturar un video en una aplicación desarrollada en Android es mediante el software que provee el sistema operativo Android. Invocamos la aplicación de tomar videos mediante la clase Intent.

Tiene como ventaja que la interfaz le es familiar al usuario, ya que muchas aplicaciones utilizan esta característica y nos facilita mucho la codificación.

2.1.8.1 Problema:

Confeccionaremos una aplicación que permita ingresar un nombre de archivo con extensión mp4 en un EditText. Luego al presionar un botón lanzar el Activity que proporciona Android para tomar un video. Grabar el archivo en la memoria externa del dispositivo. Disponer un segundo botón para recuperar un archivo mp4 almacenado en el dispositivo.

Finalmente crearemos un segundo Activity para mostrar en un ListView todos los nombres de archivos de videos almacenados por la aplicación y al presionar uno de ellos procederemos a mostrarlo en un VideoView.

2.2 Almacenamiento en dispositivos móviles

2.2.1 Almacenamiento de datos mediante la clase SharedPreferences

La plataforma de Android nos da varias facilidades para el almacenamiento permanente de datos (es decir que los mismos no se borran cuando se apaga o cierra la aplicación)

Según el tipo de necesidades utilizaremos alguno de estos métodos:

- Mediante la clase SharedPreferences.
- Mediante archivos de Texto.
- En una base de datos con acceso a SQL.

No será raro que una aplicación utilice más de uno de estos métodos para el almacenamiento de datos.

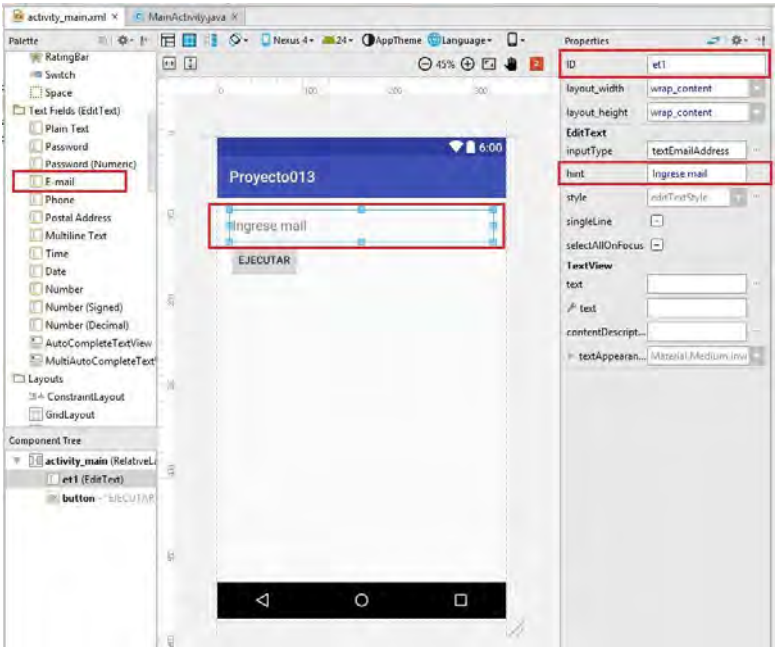
Cuando tenemos que almacenar una cantidad limitada de datos es adecuado utilizar la clase SharedPreferences. Por ejemplo configuraciones de la aplicación como pueden ser colores de pantalla, nivel actual en un juego, datos iniciales de controles de entrada etc.

2.2.1.1 Problema 1

Confeccionar un programa que solicite el ingrese del mail de una persona. Guardar el mail ingresado utilizando la clase SharedPreferences. Cada vez que se inicie la aplicación almacenar en el control EditText el último mail ingresado. Disponer un botón para almacenar el mail ingresado y finalizar el programa.

Crearemos un nuevo proyecto, la interfaz visual a implementar es:

Figura 72. Paso 1: Almacenamiento Clase Shared Preference



Fuente: Propia

Tenemos las componentes:

Disponemos un EditText (ID=”et1”, hint=”Ingrese mail”)

Disponemos un Button (onClick=”ejecutar”, text=”EJECUTAR”)

El código java es:

```
package com.tutorialesprogramacionya.proyecto013;
package com.tutorialesprogramacionya.proyecto013;

import android.content.Context;
import android.content.SharedPreferences;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
import android.view.View;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {
   private EditText et1;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      et1=(EditText)findViewById(R.id.et1);
         SharedPreferences prefe=getSharedPreferences("datos", Context.
MODE_PRIVATE);
      et1.setText(prefe.getString("mail",""));
   }

   public void ejecutar(View v) {
      SharedPreferences preferencias=getSharedPreferences("datos",Con-
text.MODE_PRIVATE);
      SharedPreferences.Editor editor=preferencias.edit();
      editor.putString("mail", et1.getText().toString());
      editor.commit();
      finish();
   }
}
```

Obtenemos la referencia del EditText:

```
et1=(EditText)findViewById(R.id.et1);
```

Obtenemos una referencia de un objeto de la clase SharedPreferences a través del método getSharedPreferences. El primer parámetro es el nombre del archivo de preferencias y el segundo

la forma de creación del archivo (MODE_PRIVATE indica que solo esta aplicación puede consultar el archivo XML que se crea)

```
SharedPreferences prefe=getSharedPreferences("datos",Context.
MODE_PRIVATE);
```

Para extraer los datos del archivo de preferencias debemos indicar el nombre a extraer y un valor de retorno si dicho nombre no existe en el archivo de preferencias (en nuestro ejemplo la primera vez que se ejecute nuestro programa como es lógico no existe el archivo de preferencias lo que hace que Android lo cree, si tratamos de extraer el valor de mail retornará el segundo parámetro es decir el String con una cadena vacía:

```
et1.setText(prefe.getString("mail",""));
```

Cuando se presiona el botón "confirmar" lo que hacemos es grabar en el archivo de preferencias el contenido del EditText en una variable llamada "mail":

```
public void ejecutar(View v) {
    SharedPreferences preferencias=getSharedPreferences("datos",Con-
text.MODE_PRIVATE);
    Editor editor=preferencias.edit();
    editor.putString("mail", et1.getText().toString());
    editor.commit();
    finish();
}
```

Debemos crear un objeto de la clase Editor y obtener la referencia del objeto de la clase SharedPreferences que acabamos de crear. Mediante el método putString almacenamos en mail el valor del String cargado en el EditText. Luego debemos llamar al método commit de la clase Editor para que el dato quede almacenado en forma permanente en el archivo de preferencias. Esto

hace que cuando volvamos a arrancar la aplicación se recupere el último mail ingresado.

Recordemos que el método finish de la clase Activity finaliza la actividad actual (como tenemos una aplicación con una sola actividad finalizará completamente nuestro programa.

Este proyecto lo puede descargar en un zip desde este enlace: proyecto013.zip

Cuando guardamos datos en el archivo de preferencias podemos almacenar distintos tipos de datos según el método que llamemos en el momento de grabar:

```
editor.putInt("edad",3);

editor.putBoolean("activo", true);

editor.putFloat("altura", 2.3f);
```

Cuando los recuperamos debemos indicar también que tipo de datos extraemos:

```
int e=prefe.getInt("edad", 0);

boolean acti=prefe.getBoolean("activo", false);

float alt=prefe.getFloat("altura", 0f);
```

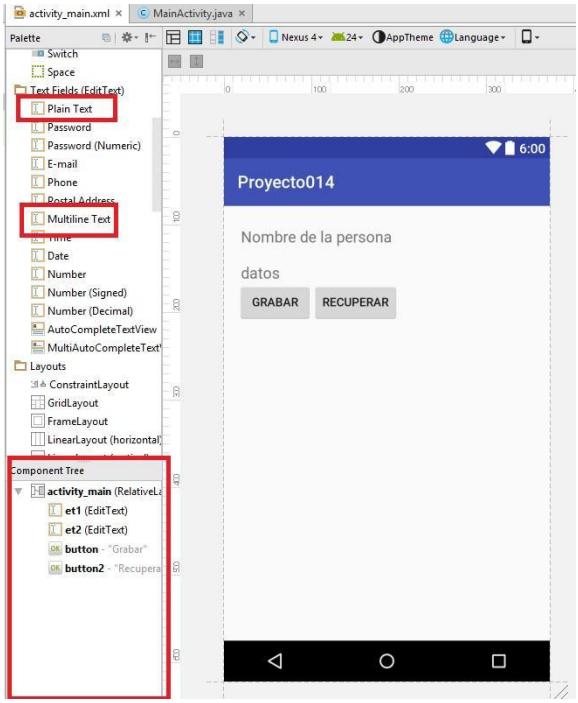
2.2.1.2 Problema 2

Confeccionar un programa que permita administrar una agenda personal. Nuestra clave será el nombre de la persona.

Crearemos un nuevo proyecto.

La interfaz visual a implementar será similar a esta:

Figura 73. Paso 2: Almacenamiento Clase Shared Preference



Fuente: Propia

EditText de tipo Plain Text (ID=et1, hint="Nombre de la persona", text="")

EditText de tipo Multiline Text (ID=et2, hint="datos")

Button (ID="button", onClick="grabar", text="GRABAR")

Button (ID="button2", onClick="recuperar", text="RECUPERAR")

El código fuente en java es:

```
package com.tutorialesprogramacionya.proyecto014;


import android.content.Context;
import android.content.SharedPreferences;
```

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
   private EditText et1,et2;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      et1=(EditText)findViewById(R.id.et1);
      et2=(EditText)findViewById(R.id.et2);
   }

   public void grabar(View v) {
      String nombre=et1.getText().toString();
      String datos=et2.getText().toString();
     SharedPreferences preferencias=getSharedPreferences("agenda", Con-
text.MODE_PRIVATE);
      SharedPreferences.Editor editor=preferencias.edit();
      editor.putString(nombre, datos);
      editor.commit();
          Toast.makeText(this,"Datos grabados", Toast.LENGTH_LONG).
show();
   }

   public void recuperar(View v) {
      String nombre=et1.getText().toString();
        SharedPreferences prefe=getSharedPreferences("agenda", Context.
```

```
MODE_PRIVATE);
      String d=prefe.getString(nombre, "");
      if (d.length()==0) {
         Toast.makeText(this,"No existe dicho nombre en la agenda", Toast.
LENGTH_LONG).show();
      }
      else {
         et2.setText(d);
      }
   }
}
```

Definimos dos objetos de la clase EditText donde se ingresan el nombre de la persona y los datos de dicha persona:

```
private EditText et1,et2;
```

Cuando se presiona el botón grabar:

```
  public void grabar(View v) {
      String nombre=et1.getText().toString();
      String datos=et2.getText().toString();
         SharedPreferences preferencias=getSharedPreferences("agenda",
Context.MODE_PRIVATE);
      SharedPreferences.Editor editor=preferencias.edit();
      editor.putString(nombre, datos);
      editor.commit();
         Toast.makeText(this,"Datos grabados",Toast.LENGTH_LONG).
show();
   }
```

Extraemos los dos datos de los EditText, creamos un objeto de la clas SharedPReferences con el nombre de "agenda". Creamos un objeto de la clase Editor y procedemos a grabar en el archivo de preferencias mediante putString:

editor.putString(nombre, datos);

Significa que en el archivo de preferencias se almacena una entrada con el nombre de la persona y los datos de dicha persona.

Por otro lado tenemos la lógica para recuperar los datos de una persona de la agenda:

```
public void recuperar(View v) {
    String nombre=et1.getText().toString();
     SharedPreferences prefe=getSharedPreferences("agenda", Context.
MODE_PRIVATE);
    String d=prefe.getString(nombre, "");
    if (d.length()==0) {
      Toast.makeText(this,"No existe dicho nombre en la agenda",Toast.
LENGTH_LONG).show();
    }
    else {
       et2.setText(d);
    }
 }
```

Abrimos el archivo de preferencias y llamamos al método getString buscando el nombre ingresado en el et1. En el caso que lo encuentre retorna el dato asociado a dicha clave.

En el emulador podemos ver como ingresamos y recuperamos datos de la agenda:

Figura 74. Paso 3: Almacenamiento Clase Shared Preference

Fuente: Propia

Este proyecto lo puede descargar en un zip desde este enlace: proyecto014.zip

2.2.1.3 Problema propuesto

Realizar un programa que genere un número aleatorio entre 1 y 50, pedir que el operador lo adivine, informar si ganó o si el número es mayor o menor al ingresado. Cuando el operador lo adivine incrementar en uno el puntaje de juego. Cada vez que se ingrese a la aplicación mostrar el puntaje actual, es decir recordar el puntaje en un archivo de preferencias.

2.2.2 Almacenamiento de datos en un archivo de texto en la memoria interna

Otra posibilidad de almacenar datos en nuestro dispositivo Android es el empleo de un archivo de texto que se guardará en el almacenamiento interno del equipo (la otro posibilidad es almacenarlo en una tarjeta SD Card)

2.2.2.1 Problema 1:

Confeccionar un programa que permita almacenar notas en un control EditText y cuando se presione un botón almacenar los datos del EditText en un archivo de texto llamado "notas.txt". Cada vez que se ingrese al programa verificar si existe el archivo de textos "notas.txt", proceder a su lectura y almacenamiento de datos en el EditText.

2.2.2.2 Problema 2:

Confeccionar un programa para administrar un calendario de actividades diarias. Los nombres de archivos corresponderán a las fechas que ingresamos. Luego cuando consultamos una fecha verificamos si hay un archivo de texto que coincida con dicha fecha.

2.2.3 Almacenamiento de datos en un archivo de texto localizado en una tarjeta SD

En el concepto anterior vimos como crear y leer un archivo de texto en la memoria interna del equipo Android. En algunas situaciones podría ser útil almacenar los datos en una tarjeta SD (tener en cuenta que no todos los dispositivos Android cuentan con esta característica), esto debido a su mayor capacidad o la facilidad de compartir los archivos con otras personas entregando la tarjeta SD.

2.2.3.1 Problema:

Confeccionar un programa que permita ingresar el nombre de un archivo y el contenido. Permitir grabar los datos ingresados

al presionar un botón. Disponer un segundo botón que permita recuperar los datos del archivo de texto.

Hacer que los archivos se graben en una tarjeta SD.

2.2.4 Almacenamiento en una base de datos SQLite

Hemos visto hasta ahora dos modos de almacenar datos en forma permanente (archivos de texto y la clase SharedPreferences), ahora veremos otra herramienta nativa de Android para almacenar datos en una base de datos llamada SQLite.

Se encuentra invitado para desarrollar un curso completo de SQLite Ya!

SQLite es una base de datos Open Source, es muy popular en muchos dispositivos pequeños, como Android.

Las ventajas que presenta utilizar SQLite es que no requiere configuración, no tiene un servidor de base de datos ejecutándose en un proceso separado y es relativamente simple su empleo.

2.2.4.1 Problema:

Confeccionar un programa que permita almacenar los datos de articulos. Crear la tabla articulos y definir los campos codigo, descripción del articulo y precio. El programa debe permitir:

1–Carga de articulo.

2–Consulta por el codigo.

3–Consulta por la descripción.

4–Borrado de un articulo ingresando su código.

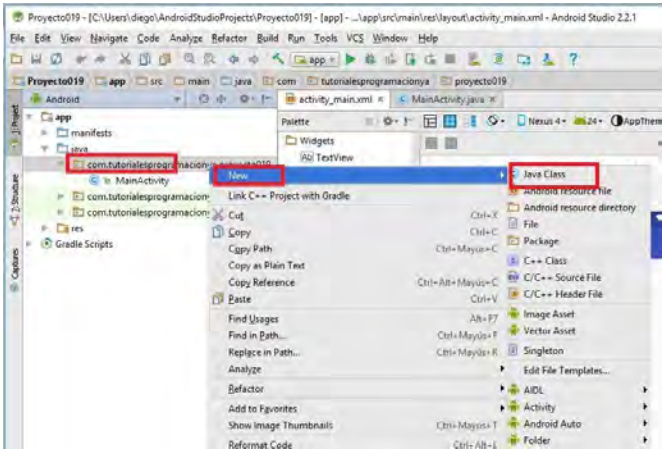
5–Modificación de la descripción y el precio.

Crear un proyecto en Android Studio y definir como nombre: Proyecto019

Lo primero que haremos es crear una clase que herede de SQLiteOpenHelper. Esta clase nos permite crear la base de datos y actualizar la estructura de tablas y datos iniciales.

Para crear una nueva clase desde Android Studio procedemos a presionar el botón derecho del mouse sobre la carpeta que contienen todos los archivo java del proyecto y seleccionamos New–> Java Class:

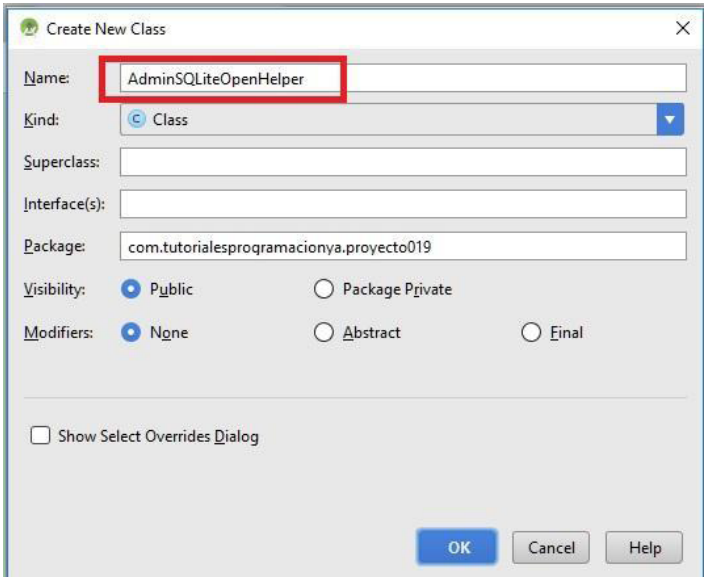
Figura 75. Paso 1: Almacenamiento SQLite



Fuente: Propia

En este diálogo ingresamos el nombre de nuestra clase, en nuestro ejemplo la llamaremos AdminSQLiteOpenHelper:

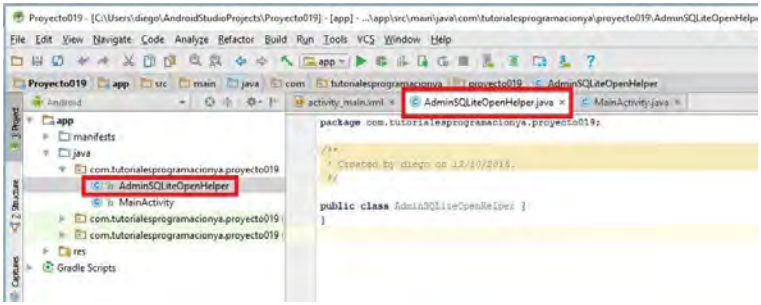
Figura 76. Paso 2: Almacenamiento SQLite



Fuente: Propia

Ya tenemos un nuevo archivo en nuestro proyecto:

Figura 77. Paso 3: Almacenamiento SQLite



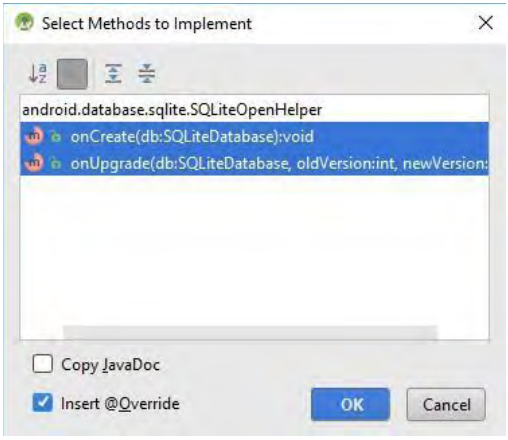
Fuente: Propia

Ahora tenemos que codificar esta clase que tiene por objetivo administrar la base de datos que crearemos. Primero hacemos que nuestra clase herede de la clase SQLiteOpenHelper:

```
package com.tutorialesprogramacionya.proyecto019;
import android.database.sqlite.SQLiteOpenHelper;
/**
* Created by diego on 16/04/2016.
*/
public class AdminSQLiteOpenHelper extends SQLiteOpenHel-
per{
}
```

La clase SQLiteOpenHelper requiere que se implementen dos métodos obligatoriamente onCreate y onUpgrade, podemos hacer que el Android Studio nos ayude en su codificación así no tenemos que tipearlos nosotros: presionamos la teclas "ALT" y "ENTER" teniendo el cursor sobre el nombre de la clase "AdminSQLiteOpenHelper" y nos aparece un menú donde seleccionamos "Implement Methods" y luego un diálogo donde seleccionamos los métodos a implementar:

Figura 78. Paso 4: Almacenamiento SQLite



Fuente: Propia

Ya tenemos la clase con los dos métodos:

```
package com.tutorialesprogramacionya.proyecto019;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;
/**
 * Created by diego on 16/04/2016.
 */
public class AdminSQLiteOpenHelper extends SQLiteOpenHelper{
   @Override
   public void onCreate(SQLiteDatabase db) {
   }
   @Override
   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVer-
sion) {
   }
}
```

Pero todavía nos marca un error ya que la clase padre SQLiteOpenHelper implementa constructores que tienen parámetros, entonces tenemos que definir el constructor en esta clase, para ello estando el cursor sobre el nombre de la clase "AdminSQLiteOpenHelper" presionamos nuevamente las teclas "ALT" y "Enter" y en el menú contectual que aparece seleccionamos "Create constructor matching super". Aparece un diálogo con todos los constructores que implementa la clase padre (seleccionammos el que tiene tres parámetros):

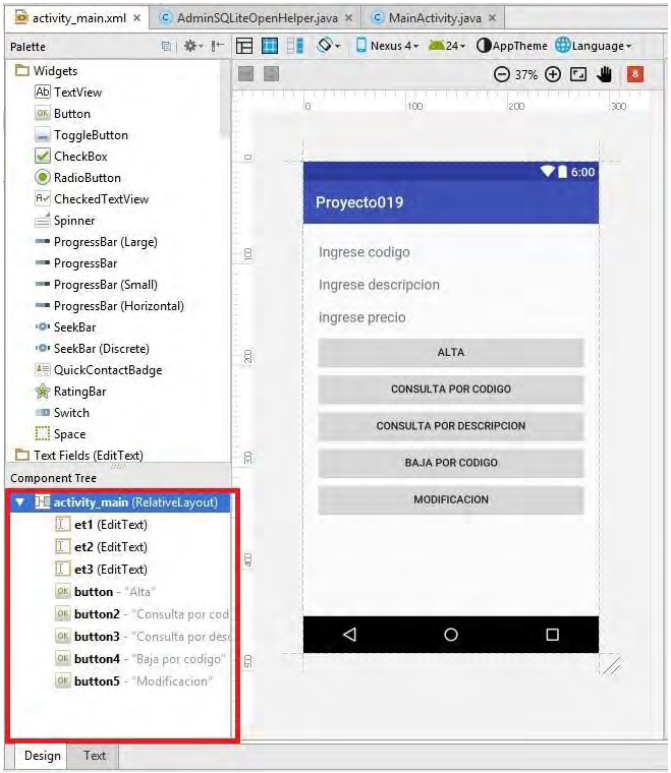
Ahora la clase no muestra ningún error y pasaremos a implementar la creación de la tabla articulos dentro del método onCreate:

```
package com.tutorialesprogramacionya.proyecto019;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

/**
 * Created by diego on 12/10/2016.
 */

public class AdminSQLiteOpenHelper  extends SQLiteOpenHelper {
    public AdminSQLiteOpenHelper(Context context, String name, SQLi-
teDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
      db.execSQL("create table articulos(codigo int primary key,descripcion
text,precio real)");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVer-
sion) {

    }
}
```

Hemos codificado en el método onCreate la creación de la tabla articulos con los campos codigo (que es entero y clave primaria), descripcion que es de tipo texto y precio es un valor real. El método onCreate se ejecutará una única vez (Eventualmente si uno quiere modificar la estructura de la tabla debemos hacerlo en el método onUpgrade).

Ahora implementemos la interfaz visual para resolver nuestro problema. Debemos crear en nuestro archivo activity_main.xml la siguiente interfaz:

Figura 79. Paso 5: Almacenamiento SQLite



Fuente: Propia

Como vemos disponemos tres EditText y cinco Button:

EditText de tipo “Number” (ID=”et1”, hint=”Ingrese codigo”)

EditText de tipo “Plain Text” (ID=”et2”, hint=”Ingrese descripcion”)

EditText de tipo “Number Decimal” (ID=”et3”, hint=”Ingrese precio”)

Button (ID=”button”, text=”Alta”, onClick=”alta”)

Button (ID=”button2”, text=”Consulta por codigo”, onClick=”consultaporcodigo”)

Button (ID=”button3”, text=”Consulta por descripcion”, onClick=”consultapordescripcion”)

Button (ID=”button4”, text=”Baja por codigo”, onClick=”bajaporcodigo”)

Button (ID=”button5”, text=”Modificacion”, onClick=”modificacion”)

El código fuente de nuestro Activity es el siguiente:

```
package com.tutorialesprogramacionya.proyecto019;

import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
```

```
public class MainActivity extends AppCompatActivity {
   private EditText et1,et2,et3;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      et1=(EditText)findViewById(R.id.et1);
      et2=(EditText)findViewById(R.id.et2);
      et3=(EditText)findViewById(R.id.et3);
   }

   public void alta(View v) {
      AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(-
this,
            "administracion", null, 1);
      SQLiteDatabase bd = admin.getWritableDatabase();
      String cod = et1.getText().toString();
      String descri = et2.getText().toString();
      String pre = et3.getText().toString();
      ContentValues registro = new ContentValues();
      registro.put("codigo", cod);
      registro.put("descripcion", descri);
      registro.put("precio", pre);
      bd.insert("articulos", null, registro);
      bd.close();
      et1.setText("");
      et2.setText("");
      et3.setText("");
      Toast.makeText(this, "Se cargaron los datos del artículo",
            Toast.LENGTH_SHORT).show();
```

```
    }

    public void consultaporcodigo(View v) {
        AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(-
this,
                "administracion", null, 1);
        SQLiteDatabase bd = admin.getWritableDatabase();
        String cod = et1.getText().toString();
        Cursor fila = bd.rawQuery(
                "select descripcion,precio from articulos where codigo=" + co|d,
null);
        if (fila.moveToFirst()) {
            et2.setText(fila.getString(0));
            et3.setText(fila.getString(1));
        } else
            Toast.makeText(this, "No existe un artículo con dicho código",
                    Toast.LENGTH_SHORT).show();
        bd.close();
    }

    public void consultapordescripcion(View v) {
        AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(-
this,
                "administracion", null, 1);
        SQLiteDatabase bd = admin.getWritableDatabase();
        String descri = et2.getText().toString();
        Cursor fila = bd.rawQuery(
                "select codigo,precio from articulos where descripcion='" + descri
+"'", null);
        if (fila.moveToFirst()) {
            et1.setText(fila.getString(0));
            et3.setText(fila.getString(1));
        } else
```

```
        Toast.makeText(this, "No existe un artículo con dicha descripción",
            Toast.LENGTH_SHORT).show();
    bd.close();
  }

  public void bajaporcodigo(View v) {
    AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(-
this,
        "administracion", null, 1);
    SQLiteDatabase bd = admin.getWritableDatabase();
    String cod= et1.getText().toString();
    int cant = bd.delete("articulos", "codigo=" + cod, null);
    bd.close();
    et1.setText("");
    et2.setText("");
    et3.setText("");
    if (cant == 1)
      Toast.makeText(this, "Se borró el artículo con dicho código",
          Toast.LENGTH_SHORT).show();
    else
      Toast.makeText(this, "No existe un artículo con dicho código",
          Toast.LENGTH_SHORT).show();
  }

  public void modificacion(View v) {
    AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(-
this,
        "administracion", null, 1);
    SQLiteDatabase bd = admin.getWritableDatabase();
    String cod = et1.getText().toString();
    String descri = et2.getText().toString();
    String pre = et3.getText().toString();
    ContentValues registro = new ContentValues();
```

```
        registro.put("codigo", cod);
        registro.put("descripcion", descri);
        registro.put("precio", pre);
        int cant = bd.update("articulos", registro, "codigo=" + cod, null);
        bd.close();
        if (cant == 1)
                Toast.makeText(this, "se modificaron los datos", Toast.LENG-
TH_SHORT)
                        .show();
        else
          Toast.makeText(this, "no existe un artículo con el código ingresado",
                        Toast.LENGTH_SHORT).show();
    }
}
```

Recordar de inicializar la propiedad onClick de cada botón para enlazarlo con el método respectivo: “alta”,”consultaporcodigo”,”consultapordescripcion”,”bajaporcodigo” y “modificacion”.

1. Alta de datos

Cuando se presiona el botón “ALTA” se ejecuta el método “alta” recordemos inicializar la propiedad “onClick” del botón desde la ventana de visualización del archivo XML. Lo primero que hacemos en este método es crear un objeto de la clase que planteamos anteriormente y le pasamos al constructor this (referencia del Activity actual), “administracion” (es el nombre de la base de datos que crearemos en el caso que no exista) luego pasamos null y un uno indicando que es la primer versión de la base de datos (en caso que cambiemos la estructura o agreguemos tablas por ejemplo podemos pasar un dos en lugar de un

uno para que se ejecute el método onUpgrade donde indicamos la nuestra estructura de la base de datos).

Luego de crear un objeto de la clase AdminSqLiteOpenHelper procedemos a crear un objeto de la clase SQLiteDataBase llamando al método getWritableDatabase (la base de datos se abre en modo lectura y escritura).

Creamos un objeto de la clase ContentValues y mediante el método put inicializamos todos tos campos a cargar.

Seguidamente llamamos al método insert de la clase SQLiteDatabase pasando en el primer parámetro el nombre de la tabla, como segundo parámetro un null y por último el objeto de la clase ContentValues ya inicializado (este método es el que provoca que se inserte una nueva fila en la tabla articulos en la base de datos llamada administracion) Borramos seguidamente los EditText y mostramos un mensaje para que conozca el operador que el alta de datos se efectuó en forma correcta:

```
  public void alta(View v) {
      AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(-
this,
            "administracion", null, 1);
      SQLiteDatabase bd = admin.getWritableDatabase();
      String cod = et1.getText().toString();
      String descri = et2.getText().toString();
      String pre = et3.getText().toString();
      ContentValues registro = new ContentValues();
      registro.put("codigo", cod);
      registro.put("descripcion", descri);
      registro.put("precio", pre);
      bd.insert("articulos", null, registro);
```

```
    bd.close();
    et1.setText("");
    et2.setText("");
    et3.setText("");
    Toast.makeText(this, "Se cargaron los datos del artículo",
        Toast.LENGTH_SHORT).show();
  }
```

2. Consulta de articulo por código.

Cuando se presiona el botón "CONSULTA POR CODIGO" se ejecuta el método consultaporcodigo:

```
  public void consultaporcodigo(View v) {
    AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(-
this,
        "administracion", null, 1);
    SQLiteDatabase bd = admin.getWritableDatabase();
    String cod = et1.getText().toString();
    Cursor fila = bd.rawQuery(
        "select descripcion,precio from articulos where codigo=" + cod,
null);
    if (fila.moveToFirst()) {
      et2.setText(fila.getString(0));
      et3.setText(fila.getString(1));
    } else
      Toast.makeText(this, "No existe un artículo con dicho código",
          Toast.LENGTH_SHORT).show();
    bd.close();
  }
```

En el método consultaporcodigo lo primero que hacemos es crear un objeto de la clase AdminSQLiteOpenHelper y obtener una referencia de la base de datos llamando al método getWritableDatabase.

Seguidamente definimos una variable de la clase Cursor y la inicializamos con el valor devuelto por el método llamado rawQuery.

La clase Cursor almacena en este caso una fila o cero filas (una en caso que hayamos ingresado un codigo existente en la tabla articulos), llamamos al método moveToFirst() de la clase Cursor y retorna true en caso de existir un articulo con el codigo ingresado, en caso contrario retorna cero.

Para recuperar los datos propiamente dichos que queremos consultar llamamos al método getString y le pasamos la posición del campo a recuperar (comienza a numerarse en cero, en este ejemplo la columna cero representa el campo descripcion y la columna 1 representa el campo precio)

3. Consulta de articulo por descripción.

Cuando se presiona el botón "CONSULTA POR DESCRIPCION" se ejecuta el método consultapordescripcion:

```
  public void consultapordescripcion(View v) {
     AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(-
this,
          "administracion", null, 1);
     SQLiteDatabase bd = admin.getWritableDatabase();
     String descri = et2.getText().toString();
     Cursor fila = bd.rawQuery(
         "select codigo,precio from articulos where descripcion='" + descri
+"'", null);
```

```
    if (fila.moveToFirst()) {
        et1.setText(fila.getString(0));
        et3.setText(fila.getString(1));
    } else
        Toast.makeText(this, "No existe un artículo con dicha descripción",
                Toast.LENGTH_SHORT).show();
    bd.close();
}
```

En el método consultapordescripcion lo primero que hacemos es crear un objeto de la clase AdminSQLiteOpenHelper y obtener una referencia de la base de datos llamando al método getWritableDatabase.

Seguidamente definimos una variable de la clase Cursor y la inicializamos con el valor devuelto por el método llamado rawQuery.

Es importante notar en el where de la clausula SQL hemos dispuesto comillas simples entre el contenido de la variable descri:

```
"select codigo,precio from articulos where descripcion='" + descri +"'", null);
```

Esto es obligatorio para los campos de tipo text (en este caso descripcion es de tipo text)

4. Baja o borrado de datos.

Para borrar uno o más registros la clase SQLiteDatabase tiene un método que le pasamos en el primer parámetro el nombre de la tabla y en el segundo la condición que debe cumplirse para

que se borre la fila de la tabla. El método delete retorna un entero que indica la cantidad de registros borrados:

```
public void bajaporcodigo(View v) {
        AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(this,
            "administracion", null, 1);
        SQLiteDatabase bd = admin.getWritableDatabase();
        String cod= et1.getText().toString();
        int cant = bd.delete("articulos", "codigo=" + cod, null);
        bd.close();
        et1.setText("");
        et2.setText("");
        et3.setText("");
        if (cant == 1)
          Toast.makeText(this, "Se borró el artículo con dicho código",
              Toast.LENGTH_SHORT).show();
        else
          Toast.makeText(this, "No existe un artículo con dicho código",
              Toast.LENGTH_SHORT).show();
}
```

5. Modificación de datos.

En la modificación de datos debemos crear un objeto de la clase ContentValues y mediante el método put almacenar los valores para cada campo que será modificado. Luego se llama al método update de la clase SQLiteDatabase pasando el nombre de la tabla, el objeto de la clase ContentValues y la condición del where (el cuanto parámetro en este ejemplo no se lo emplea).

```
  public void modificacion(View v) {
      AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(-
this,
            "administracion", null, 1);
      SQLiteDatabase bd = admin.getWritableDatabase();
      String cod = et1.getText().toString();
      String descri = et2.getText().toString();
      String pre = et3.getText().toString();
      ContentValues registro = new ContentValues();
      registro.put("codigo", cod);
      registro.put("descripcion", descri);
      registro.put("precio", pre);
      int cant = bd.update("articulos", registro, "codigo=" + cod, null);
      bd.close();
      if (cant == 1)
            Toast.makeText(this, "se modificaron los datos", Toast.LENG-
TH_SHORT)
             .show();
      else
        Toast.makeText(this, "no existe un artículo con el código ingresado",
             Toast.LENGTH_SHORT).show();
  }
```

Cuando ejecutamos el programa tenemos la siguiente interfaz:

Figura 80. Paso 6: Almacenamiento SQLite

Fuente: Propia

2.2.5 Acceso a Servicios Web en Android

Hemos visto hasta ahora modos de almacenar datos en forma permanente (archivos de texto, clase SharedPreferences y SQLite), ahora veremos otra manera de Android para almacenar datos en una base de datos en MySQL que estará en en servidor externo

MySQL es una base de datos Open Source, las ventajas que presenta utilizar MySQL tiene un servidor de base de datos ejecutándose en un proceso separado y es relativamente simple su empleo.

2.2.5.1 Problema:

Confeccionar un programa que permita almacenar los datos de articulos. Crear la tabla articulos y definir los campos nombre, apellido, sexo, pais, provincia, y dirección. El programa debe permitir:

1–Carga de persona.

2–Consulta por la cedula.

3–Borrado de una persona ingresando su cédula.

4–Modificación de una persona por medio de su cédula.

Crear un proyecto en Android Studio. Lo primero que haremos es crear el diseño del XML como se muestra en la siguiente interfaz:

Figura 81. Paso 1: Acceso a Servicios Web en Android

Fuente: Propia

Como vemos disponemos tres EditText y cinco Button:

EditText de tipo “Plain Text” (ID=”etNombre”, hint=”Ingrese Nombre”)

EditText de tipo “Plain Text” (ID=”etApellido”,hint=”Ingrese Apellido”)

EditText de tipo “Number” (ID=”etCedula”, hint=”Ingrese cedula”)

RadioButton (ID=”rbtnMasculino”)

RadioButton (ID=”rbtnFemenino”)

Spinner (ID=”spPais”)

EditText de tipo “Plain Text” (ID=”etProvincia”, hint=”Ingrese provincia”)

EditText de tipo “ Plain Text “ (ID=”etDireccion”, hint=”Ingrese direccion”)

Button (ID=”btnIngresar”, text=”Guardar”)

Button (ID=”btnBuscar”, text=”Buscar”)

Button (ID=”btnBorrar”, text=”Borrar”)

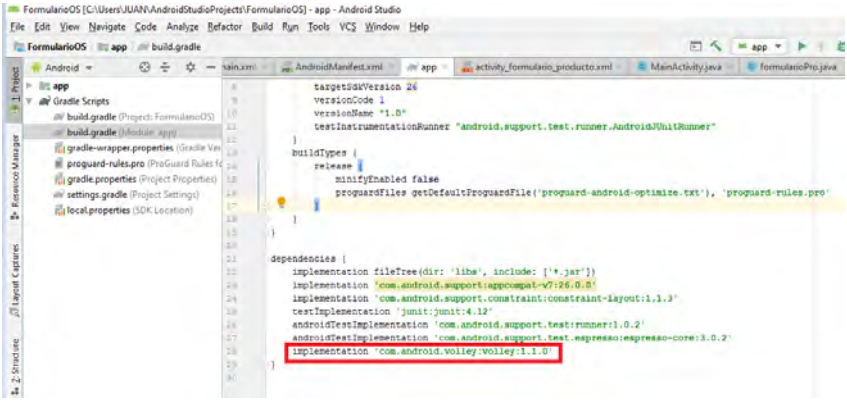
Button (ID=”btnActualizar”, text=”Actualizar”)

Después configuramos los archivos gradle como se muesta a continuación

En el archivo build.gradle(Module:app) implementar lo siguiente:

implementation **‘com.android.volley:volley:1.1.0’**

Figura 82. Paso 2: Acceso a Servicios Web en Android



Fuente: Propia

El código fuente de nuestro Activity es el siguiente:

```
package ec.edu.intsuperior.formularioos;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.RadioButton;
import android.widget.Spinner;
import android.widget.Toast;

import com.android.volley.AuthFailureError;
import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.JsonArrayRequest;
import com.android.volley.toolbox.StringRequest;
import com.android.volley.toolbox.Volley;

import org.json.JSONArray;
import org.json.JSONException;
```

```
import org.json.JSONObject;

import java.util.HashMap;
import java.util.Map;

public class MainActivity extends AppCompatActivity {
 private Spinner spinner1;
 private EditText nombre,apellido,cedula,provincia,direccion;
 private RadioButton masculino,femenino;
 private Button btnAgregar,btnBuscar,btnActualizar,btnBorrar;
 RequestQueue requestQueue;

 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 spinner1 = (Spinner) findViewById(R.id.spPais);
 String []opciones={"Ecuador","Colombia","Venezuela","Peru"};
 ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,an-
droid.R.layout.simple_spinner_item, opciones);
 spinner1.setAdapter(adapter);
 nombre=(EditText)findViewById(R.id.etNombre);
 apellido=(EditText)findViewById(R.id.etApellido);
 cedula=(EditText)findViewById(R.id.etCedula);
 provincia=(EditText)findViewById(R.id.etProvincia);
 direccion=(EditText)findViewById(R.id.etDireccion);
 masculino=(RadioButton)findViewById(R.id.rbtnMasculino);
 femenino=(RadioButton)findViewById(R.id.rbtnFemenino);
 btnAgregar=(Button)findViewById(R.id.btnIngresar);
 btnBuscar=(Button)findViewById(R.id.btnBuscar);
 btnActualizar=(Button)findViewById(R.id.btnActualizar);
 btnBorrar=(Button)findViewById(R.id.btnBorrar);


 btnAgregar.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
  ejecutarServico("http://192.168.1.6/formulario/insertardatos.php");
  }
 });
```

```
btnBuscar.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    buscar("http://192.168.1.6/formulario/buscardatos.php?cedula="+-
cedula.getText()+"");
  }
});
btnActualizar.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    actualizar("http://192.168.1.6/formulario/actualizardatos.php?cedu-
la="+cedula.getText()+"");
   }
});
btnBorrar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
      borrar("http://192.168.1.6/formulario/borrardatos.php?cedu-
la="+  cedula.getText()+"");
    }
  });
}

private void ejecutarServico(String URL){
StringRequest stringRequest=new StringRequest(Request.Method.POST,
URL, new Response.Listener<String>() {
@Override
public void onResponse(String response) {
Toast.makeText(getApplicationContext(), "Operacion exitosa", Toast.
LENGTH_SHORT).show();
}
}, new Response.ErrorListener() {
@Override
public void onErrorResponse(VolleyError error) {
Toast.makeText(getApplicationContext(),error.toString(), Toast.LENG-
TH_SHORT).show();
}
}){
@Override
protected Map<String, String> getParams() throws AuthFailureError {
```

```
    Map<String,String> parametros=new HashMap<String,String>();
    parametros.put("nombres",nombre.getText().toString());
    parametros.put("apellidos",apellido.getText().toString());
    parametros.put("cedula",cedula.getText().toString());
    parametros.put("sexo",sexo());
    parametros.put("pais",String.valueOf(spinner1.getSelectedItemPosition()));
    parametros.put("provincia",provincia.getText().toString());
    parametros.put("direccion",direccion.getText().toString());
    return parametros;
   }
  };
  requestQueue= Volley.newRequestQueue(this);
  requestQueue.add(stringRequest);
 }
  public String sexo() {
 String sexo="";
 if (masculino.isChecked() == true) {
 sexo = "Masculino";
 } else if (femenino.isChecked() == true) {
 sexo = "Femenino";
 }
 return sexo;

 }
 private void buscar(String URL){
 JsonArrayRequest jsonArrayRequest=new JsonArrayRequest(URL, new
Response.Listener<JSONArray>() {
 @Override
 public void onResponse(JSONArray response) {
 JSONObject jsonObject = null;
 for (int i = 0; i < response.length(); i++) {
 try {
 jsonObject = response.getJSONObject(i);
 nombre.setText(jsonObject.getString("nombres"));
 apellido.setText(jsonObject.getString("apellidos"));
 cedula.setText(jsonObject.getString("cedula"));
 provincia.setText(jsonObject.getString("provincia"));
 direccion.setText(jsonObject.getString("direccion"));
 } catch (JSONException e) {
```

```
 Toast.makeText(getApplicationContext(), e.getMessage(), Toast.LENG-
TH_SHORT).show();
 }
 }
 }
 }, new Response.ErrorListener() {
 @Override
 public void onErrorResponse(VolleyError error) {
 Toast.makeText(getApplicationContext(),"Error de conexion",Toast.
LENGTH_SHORT).show();
 }
 }
 );
 requestQueue=Volley.newRequestQueue(this);
 requestQueue.add(jsonArrayRequest);
 }

 private void actualizar(String URL){
 StringRequest stringRequest=new StringRequest(Request.Method.POST,
URL, new Response.Listener<String>() {
 @Override
 public void onResponse(String response) {
 Toast.makeText(getApplicationContext(), "Operacion exitosa", Toast.
LENGTH_SHORT).show();
 }
 }, new Response.ErrorListener() {
 @Override
 public void onErrorResponse(VolleyError error) {
 Toast.makeText(getApplicationContext(),error.toString(), Toast.LENG-
TH_SHORT).show();
 }
 }){
 @Override
 protected Map<String, String> getParams() throws AuthFailureError {
 Map<String,String> parametros=new HashMap<String,String>();
 parametros.put("nombres",nombre.getText().toString());
 parametros.put("apellidos",apellido.getText().toString());
 parametros.put("sexo",sexo());
 parametros.put("pais",String.valueOf(spinner1.getSelectedItemPosi-
tion()));
```

```
 parametros.put("provincia",provincia.getText().toString());
 parametros.put("direccion",direccion.getText().toString());
 return parametros;
 }
 };
 requestQueue= Volley.newRequestQueue(this);
 requestQueue.add(stringRequest);
 }
 private void borrar(String URL){
 StringRequest stringRequest=new StringRequest(Request.Method.POST,
URL, new Response.Listener<String>() {
 @Override
 public void onResponse(String response) {
 Toast.makeText(getApplicationContext(), "Operacion exitosa", Toast.
LENGTH_SHORT).show();
 }
 }, new Response.ErrorListener() {
 @Override
 public void onErrorResponse(VolleyError error) {
 Toast.makeText(getApplicationContext(),error.toString(), Toast.LENG-
TH_SHORT).show();
 }
 }){
 @Override
 protected Map<String, String> getParams() throws AuthFailureError {
 Map<String,String> parametros=new HashMap<String,String>();
 return parametros;
 }
 };
 requestQueue= Volley.newRequestQueue(this);
 requestQueue.add(stringRequest);
 }
}
```

Después creamos la base datos en MySQL y los Web Service implementados en Apache–PHP que permitirán realizar el CRUD, para ello usaremos el servidor local XAMPP, para ello explicaremos que es cada una de estas plataformas:

Figura 83. Logo Software XAMPP

Fuente: Propia

¿QUE ES XAMPP Y COMO PUEDO USARLO?

Como ya lo debes haber deducido, XAMPP es una herramienta de desarrollo que te permite probar tu trabajo (páginas web o programación, por ejemplo) en tu propio ordenador sin necesidad de tener acceso a internet.

Si eres un diseñador web o desarrollador web que recién está comenzando, tampoco debes preocuparte sobre las configuraciones ya que XAMPP te provee de una configuración totalmente funcional desde el momento que lo instalas (básicamente lo extraes). Sin embargo, es bueno acotar que la seguridad de datos no es su punto fuerte, por lo cual no es suficientemente seguro para ambientes grandes o de producción.

¿QUE ES XAMPP?

XAMPP es una distribución de Apache que incluye varios softwares libres. El nombre es un acrónimo compuesto por las iniciales de los programas que lo constituyen: el servidor web Apache, los sistemas relacionales de administración de bases de datos MySQL y MariaDB, así como los lenguajes de programación Perl y PHP. La inicial X se usa para representar a los sistemas operativos Linux, Windows y Mac OS X.

Apache: el servidor web de código abierto es la aplicación más usada globalmente para la entrega de contenidos web. Las aplicaciones del servidor son ofrecidas como software libre por la Apache Software Foundation.

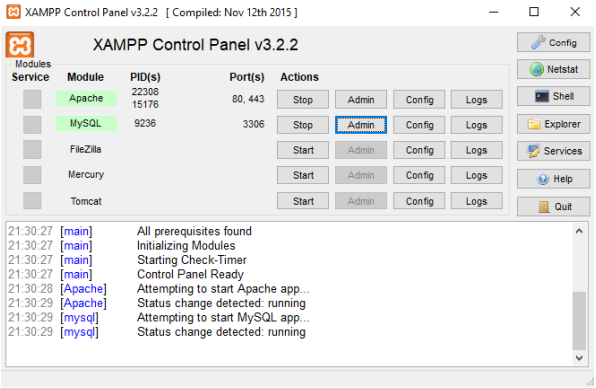
PHP: es un lenguaje de programación de código de lado del servidor que permite crear páginas web o aplicaciones dinámicas. Es independiente de plataforma y soporta varios sistemas de bases de datos.

MySQL/MariaDB: XAMPP cuenta con uno de los sistemas relacionales de gestión de bases de datos más populares del mundo. En combinación con el servidor web Apache y el lenguaje PHP, MySQL sirve para el almacenamiento de datos para servicios web. En las versiones actuales de XAMPP esta base de datos se ha sustituido por MariaDB.

Perl: este lenguaje de programación se usa en la administración del sistema, en el desarrollo web y en la programación de red. También permite programar aplicaciones web dinámicas.

Además de estos componentes principales, esta distribución gratuita también incluye, según el sistema operativo, otras herramientas como el servidor de correo Mercury, el programa de administración de bases de datos phpMyAdmin, el software de analítica web Webalizer, OpenSSL, Apache Tomcat y los servidores FTP FileZilla o ProFTPd.

Figura 84. Entorno de Trabajo XAMPP



Fuente: Propia

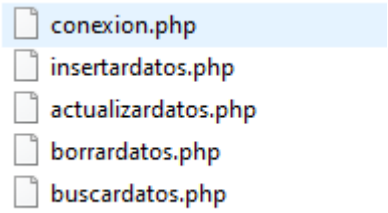
Una vez explicada las herramientas se implementará la Base

de datos en MySQL con la tabla persona y sus atributos nombre, apellido, sexo, pais, provincia, y dirección.

```
— Base de datos: `formulario`
—
— ————————————————————————————————————
—
— Estructura de tabla para la tabla `persona`
—
CREATE TABLE `persona` (
    `nombres` varchar(45) NOT NULL,
    `apellidos` varchar(45) NOT NULL,
    `cedula` varchar(10) NOT NULL,
    `sexo` varchar(15) NOT NULL,
    `pais` varchar(45) NOT NULL,
    `provincia` varchar(45) NOT NULL,
    `direccion` varchar(50) NOT NULL
)
```

Los Web Service se implementarán en Apache – PHP en la siguiente localización de fichero C:\xampp\htdocs\formulario y se crea los siguientes archivos con extencion .php como se muestra acontinucación:

Figura 85. Archivos Web Services PHP



Fuente: Propia

1. Archivo: conexión.php

```
<?php
$hostname='localhost';
$database='formulario';
$username='root';
$password='';
$conexion=new mysqli($hostname,$username,$password,$database);
if($conexion->connect_errno){
echo "lo sentimos el sitio web esta experimentando problemas";
}
?>
```

2. Archivo: insertardatos.php

```
<?php
include 'conexion.php';
$nombres=$_POST['nombres'];
$apellidos=$_POST['apellidos'];
$cedula=$_POST['cedula'];
$sexo=$_POST['sexo'];
$pais=$_POST['pais'];
$provincia=$_POST['provincia'];
$direccion=$_POST['direccion'];
$consulta="insert into datos values('".$nombres."','".$apellidos."','".$cedula."','".$sexo."','".$pais."','".$provincia."','".$direccion."')";
mysqli_query($conexion,$consulta) or die(mysqli_error());
mysqli_close($conexion);
?>
```

3. Archivo: buscardatos.php

```
<?php
include 'conexion.php';
```

```
$cedula=$_GET['cedula'];
$consulta="select nombres,apellidos,cedula,provincia,direccion
from datos where cedula ='$cedula'";
$resultado=$conexion->query($consulta);
while($fila=$resultado->fetch_array()){
    $dato[]=array_map('utf8_encode',$fila);
}
echo json_encode($dato);
$resultado->close();
?>
```

4. Archivo: actualizardatos.php

```
<?php
include 'conexion.php';
$cedula=$_GET['cedula'];
$nombres=$_POST['nombres'];
$apellidos=$_POST['apellidos'];
$sexo=$_POST['sexo'];
$pais=$_POST['pais'];
$provincia=$_POST['provincia'];
$direccion=$_POST['direccion'];
$consulta=" update datos
set nombres='".$nombres."',
    apellidos='".$apellidos."',
    sexo='".$sexo."',
    pais='".$pais."',
    provincia='".$provincia."',
    direccion='".$direccion."'
where cedula ='$cedula'";
mysqli_query($conexion,$consulta) or die(mysqli_error());
mysqli_close($conexion);
?>
```

5. Archivo: borrardatos.php

```
<?php
include ‘conexion.php’;
$cedula=$_GET[‘cedula’];
$consulta=”delete from datos
where cedula =’$cedula”’;
mysqli_query($conexion,$consulta) or die(mysqli_error());
mysqli_close($conexion);
?>
```

2.2.6 Evaluación 3

1. ¿Cuál es el método en SQLite que permite que la base de datos se abra en modo lectura y escritura en Android?
2. ¿Cuál componente agrupa componentes en filas y columnas?
3. ¿Cuál componente nos permite disponer una cantidad de elementos visuales que superan la cantidad de espacio del visor del celular o Tablet y luego el usuario puede desplazar con el dedo la interfaz creada?
4. ¿Cuál es el componente que permite realizar diseños más simples ya que se puede establecer los componentes visuales uno junto al otro, ya sea horizontal o verticalmente?
5. Si se usa un objeto de la clase MediaPlayer con cual método hacemos referencia al archivo de audio que copiamos en la carpeta raw
6. ¿Cuál es la clase que permite la captura de audio en Android?

7. ¿Para que se utiliza el archivo strings.xml?
8. ¿Cuál es el componente ActionBar?
9. La plataforma de Android se pueden agregar y eliminar elementos en el ListView, al momento de añadir un elemento al ArrayList , Cuál es el método que debemos llamar del ArrayAdapter para que informe al ListView que actualice los datos en pantalla
10. ¿Cuál método no es un tipo de Alerta o Notificación en Android?
11. ¿Cuál tipo de notificación constan de un icono y un texto mostrado en la barra de estado superior, y adicionalmente un mensaje algo más descriptivo, una marca de fecha/hora que podemos consultar desplegando la bandeja del sistema?
12. ¿Cuál tipo de notificación que debe utilizarse para mostrar feedbacks sobre alguna operación realizada por el usuario ya que aparece en pantalla por un corto periodo de tiempo y después desaparece automáticamente, además puede contener un botón de texto de acción?
13. En Android se puede implementar Web Services para enlazarse a un gestor de base de datos, cuál de las siguientes herramientas me ayuda con este tipo de implementación
14. ¿Cuál es la definición correcta de Apache?

Capítulo 3
Firebase Authentication

3.1 Firebase Authentication

¿Qué es Firebase?

Firebase permite que los equipos de apps para dispositivos móviles y web alcancen el éxito.

Compila apps rápido, sin administrar la infraestructura

Firebase te proporciona funciones como estadísticas, bases de datos, informes de fallas y mensajería, de manera que puedas ser más eficiente y enfocarte en tus usuarios.

Con el respaldo de Google y la confianza de apps reconocidas

Firebase utiliza la infraestructura de Google y se escala automáticamente, incluso para las apps más grandes.

Una plataforma con productos que funcionan mejor en conjunto

Los productos de Firebase funcionan a la perfección por sí solos. Sin embargo, debido a que comparten datos y estadísticas, funcionan aún mejor juntos.

Compila mejores apps

- **Cloud Firestore:** Almacena y sincroniza los datos de tu app a escala global.
- **Firebase ML BETA**: Aprendizaje automático para desarrolladores de apps para dispositivos móviles.
- **Cloud Functions:** Ejecuta código de back-end para dispositivos móviles sin administrar servidores.
- **Authentication:** Autentica usuarios de forma simple y segura.

- **Hosting:** Entrega recursos de aplicaciones web con velocidad y seguridad
- Cloud Storage: Almacena y envía archivos a la escala de Google
- **Realtime Database:** Almacena y sincroniza datos de app en milisegundos

Mejora la calidad de las apps

- **Crashlytics:** Prioriza y soluciona problemas con informes de fallas potentes y en tiempo real.
- **Performance Monitoring:** Obtén estadísticas sobre el rendimiento de tu app.
- **Test Lab:** Prueba la app en dispositivos alojados en Google.
- **App Distribution BETA:** Distribuye versiones preliminares de tu aplicación a tus testers de confianza.

Haz crecer tu negocio

- **In-App Messaging BETA:** Usa mensajes contextuales para interactuar con los usuarios activos de la app.
- **Google Analytics:** Obtén datos de analítica ilimitados sobre tu app sin cargo.
- **Predictions:** Smart user segmentation based on predicted behavior.
- **A/B Testing BETA:** Optimiza la experiencia que ofrece tu app a través de experimentos.
- **Cloud Messaging:** Envía notificaciones y mensajes orientados.

- **Remote Config:** Modifica tu app sin implementar una versión nueva
- **Dynamic Links:** Usa vínculos directos con atribución para impulsar el crecimiento

3.1.1 Firebase Authentication (I)

Acceso fácil con cualquier plataforma

Firebase Authentication busca facilitar la creación de sistemas de autenticación seguros, a la vez que mejora la experiencia de integración y acceso para los usuarios finales. Proporciona una solución de identidad de extremo a extremo, compatible con cuentas de correo electrónico y contraseñas, autenticación telefónica, acceso mediante Google, Twitter, Facebook y GitHub, y mucho más.

IU flexible y directa

FirebaseUI proporciona una solución de autenticación directa, personalizable y de código abierto que administra los flujos de IU para el acceso de los usuarios. El componente de FirebaseUI Auth implementa recomendaciones para la autenticación en sitios web y dispositivos móviles, lo que puede maximizar la conversión de acceso y registro de tu app.

Seguridad integral

La seguridad de Firebase, creada por el mismo equipo que desarrolló Acceso con Google, Smart Lock y el Administrador de contraseñas de Chrome, aplica la experiencia interna de Google en la administración de una de las bases de datos de cuentas más grandes del mundo.

Implementación rápida

Puede llevar meses configurar su propio sistema de autenticación y se requiere un equipo de ingenieros para mantener ese sistema en el futuro. Configure todo el sistema de autenticación de su aplicación en menos de 10 líneas de código, incluso manejando casos complejos como la fusión de cuentas.

3.1.2 Correo electrónico y Contraseña

Autentica con Firebase mediante cuentas con contraseña en Android.

Puedes usar Firebase Authentication para permitir que los usuarios se autentiquen con Firebase mediante su dirección de correo electrónico y contraseña, y para administrar las cuentas basadas en contraseña de tu app.

Antes de comenzar

1. Si aún no lo has hecho, agrega Firebase a tu proyecto de Android.
2. En tu archivo build.gradle de nivel de proyecto, asegúrate de incluir el repositorio Maven de Google en las secciones buildscript y allprojects.
3. Agrega la dependencia de la biblioteca de Android de Firebase Authentication al archivo Gradle (generalmente app/build.gradle) de tu módulo (a nivel de la app):

```
implementation 'com.google.firebase:firebase-auth:19.3.2'
```

4. Si aún no conectaste la app al proyecto de Firebase, puedes hacerlo desde Firebase console.

5. Habilita el acceso con correo electrónico y contraseña:

- En Firebase console, abre la sección **Auth**.
- En la pestaña **Método de acceso**, habilita el método de acceso **Correo electrónico/contraseña** y haz clic en **Guardar**.

Crea una cuenta basada en contraseña

Para crear una cuenta de usuario nueva con una contraseña, sigue estos pasos en la actividad de acceso de la app:

1. En el método onCreate de tu actividad de registro, obtén la instancia compartida del objeto FirebaseAuth:

```
private FirebaseAuth mAuth;
// ...
// Initialize Firebase Auth
mAuth = FirebaseAuth.getInstance();
```

2. Cuando inicialices tu actividad, verifica que el usuario haya accedido:

```
@Override
public void onStart() {
  super.onStart();
  // Check if user is signed in (non-null) and update UI accordingly.
  FirebaseUser currentUser = mAuth.getCurrentUser();
  updateUI(currentUser);
}
```

3. Cuando un usuario nuevo se registre mediante el formulario de registro de la app, realiza los pasos de validación de la cuenta nueva necesarios, como verificar que se haya escrito correcta-

mente la contraseña y que cumpla con los requisitos de complejidad.

4. Para crear una cuenta nueva, pasa la dirección de correo electrónico y la contraseña del usuario nuevo a createUserWithEmailAndPassword:

```
mAuth.createUserWithEmailAndPassword(email, password)
      .addOnCompleteListener(this, new OnCompleteListener<AuthRe-
sult>() {
          @Override
          public void onComplete(@NonNull Task<AuthResult> task) {
              if (task.isSuccessful()) {
                  // Sign in success, update UI with the signed-in user's infor-
mation
                  Log.d(TAG, "createUserWithEmail:success");
                  FirebaseUser user = mAuth.getCurrentUser();
                  updateUI(user);
              } else {
                  // If sign in fails, display a message to the user.
                  Log.w(TAG, "createUserWithEmail:failure", task.getExcep-
tion());
                  Toast.makeText(EmailPasswordActivity.this, "Authentication
failed.",
                          Toast.LENGTH_SHORT).show();
                  updateUI(null);
              }

              // ...
          }
      });
```

EmailPasswordActivity.java

Si se creó la cuenta nueva, el usuario también accede. En la devolución de llamada, puedes usar el método getCurrentUser para obtener los datos de la cuenta del usuario.

Para proteger tu proyecto de abusos, Firebase limita la cantidad de registros de usuarios nuevos (ya sean anónimos o con correo electrónico y contraseña) que tu aplicación puede recibir desde una misma dirección IP en un período breve. Puedes solicitar y programar cambios temporales para esta cuota desde Firebase console

Haz que un usuario acceda con una dirección de correo electrónico y una contraseña

Los pasos para que un usuario acceda con una contraseña son similares a los pasos para crear una cuenta nueva. En la actividad de acceso de tu app, haz lo siguiente:

1. En el método onCreate de tu actividad de acceso, obtén la instancia compartida del objeto FirebaseAuth:

```
private FirebaseAuth mAuth;
// ...
// Initialize Firebase Auth
mAuth = FirebaseAuth.getInstance();
```

2. Cuando inicialices tu actividad, verifica que el usuario haya accedido:

```
@Override
public void onStart() {
    super.onStart();
    // Check if user is signed in (non-null) and update UI accordingly.
    FirebaseUser currentUser = mAuth.getCurrentUser();
    updateUI(currentUser);
}
```

3. Cuando un usuario acceda a tu app, pasa la dirección de correo electrónico y la contraseña a signInWithEmailAndPassword:

```
mAuth.signInWithEmailAndPassword(email, password)
      .addOnCompleteListener(this, new OnCompleteListener<AuthRe-
sult>() {
          @Override
          public void onComplete(@NonNull Task<AuthResult> task) {
              if (task.isSuccessful()) {
                  // Sign in success, update UI with the signed-in user's infor-
mation
                  Log.d(TAG, "signInWithEmail:success");
                  FirebaseUser user = mAuth.getCurrentUser();
                  updateUI(user);
              } else {
                  // If sign in fails, display a message to the user.
                  Log.w(TAG, "signInWithEmail:failure", task.getException());
                  Toast.makeText(EmailPasswordActivity.this, "Authentication
failed.",
                          Toast.LENGTH_SHORT).show();
                  updateUI(null);
                  // ...
              }

              // ...
          }
      });
```

Si se accede correctamente, puedes usar el valor FirebaseUser que se muestra para continuar.

Próximos pasos

Cuando un usuario accede por primera vez, se crea una cuenta de usuario nueva y se la vincula con las credenciales (el nombre de usuario y la contraseña, el número de teléfono o la información del proveedor de autenticación) que el usuario utilizó para acceder. Esta cuenta se almacena como parte de tu proyecto de Firebase y se puede usar para identificar a un usuario en todas las apps del proyecto, sin importar cómo acceda.

- En tus apps, puedes obtener la información básica de perfil del usuario a partir del objeto FirebaseUser. Consulta Administra usuarios en Firebase.
- En tus reglas de seguridad de Firebase Realtime Database y Cloud Storage, puedes obtener el ID del usuario único que accedió a partir de la variable auth y usarlo para controlar a qué datos podrá acceder.

Para permitir que los usuarios accedan a la app con varios proveedores de autenticación, puedes vincular las credenciales de estos proveedores con una cuenta de usuario existente.

Para salir de la sesión de un usuario, llama a signOut:

```
FirebaseAuth.getInstance().signOut();
```

3.1.3 Google Sign-In en Android

Aunque muchas aplicaciones siguen utilizando su propio sistema de autenticación de usuarios, son cada vez más las que proporcionan al usuario la posibilidad de loguearse utilizando una cuenta ya existente en algún otro servicio, como por ejemplo Facebook, Twitter o Google. Esto tiene varias ventajas, desde el punto de vista del desarrollador permite aliviar un poco el trabajo al no tener que desarrollar todo un nuevo sistema de creación de cuentas de usuario y autenticación para la aplicación, y de cara al usuario también alivia el trabajo de tener que recordar un nuevo usuario y contraseña :). En este artículo veremos cómo hacer uso del servicio de autenticación/login de Google (Google Sign-

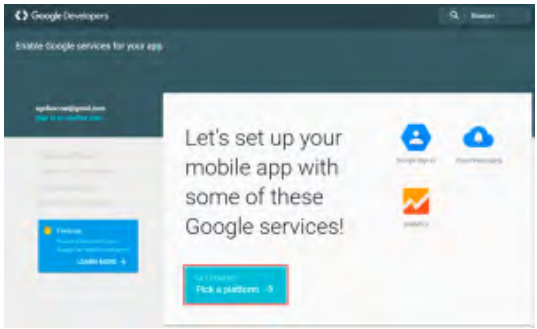
In) en nuestras aplicaciones Android, de forma que el usuario pueda acceder a ella usando cualquier cuenta de Google existente en el dispositivo (o bien crear una nueva).

Al igual que ocurría con la API de Google Maps para Android, para hacer uso del servicio de autenticación de Google es necesario crear previamente un proyecto en la consola de desarrolladores. Aunque podríamos crearlo manualmente como en el caso anterior, en esta ocasión vamos a utilizar otra herramienta que proporciona Google que se denomina **Firebase** (https://firebase.google.com/) para alguno de sus servicios, que nos ayudará tanto en la creación del proyecto como en la configuración posterior de la aplicación Android para asociarla a éste.

3.1.3.1 Problema:

Vamos a acceder para ello a este enlace, desde donde podemos agregar algunos servicios (como el de *login* o el de *analytics*) a un proyecto existente o bien, si aún no existe, hacer que la herramienta lo cree automáticamente por nosotros.

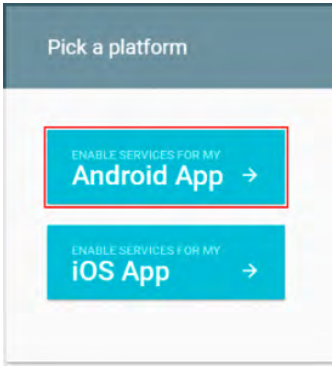
Figura 86. Paso 1: Entorno Firebase



Fuente: Propia

Empezaremos por pulsar el botón «Pick a platform» para indicar qué tipo de aplicación vamos a crear. Indicaremos por supuesto «Android App».

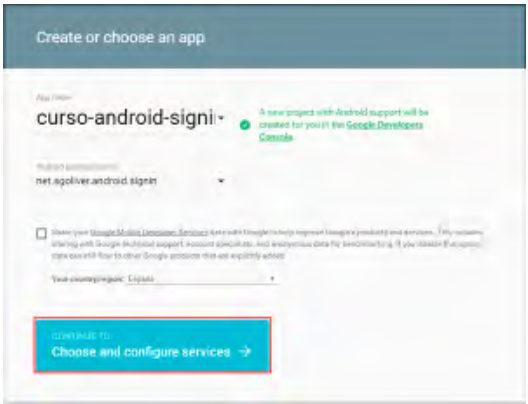
Figura 87. Paso 2: Entorno Firebase



Fuente: Propia

A continuación el asistente nos pregunta el nombre de la aplicación y el paquete java principal. Si no existe ningún proyecto con dicho nombre en la consola de desarrolladores nos indicará que el proyecto se va a crear automáticamente. Si lo deseamos, desmarcamos la opción de compartir datos con Google y seleccionamos nuestro país o región. Finalmente pulsamos «Choose and configure services» para continuar.

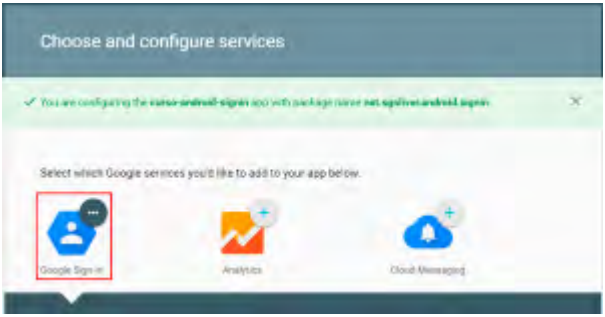
Figura 88. Paso 3: Entorno Firebase



Fuente: Propia

En el siguiente paso tendremos que elegir el servicio que queremos añadir al proyecto. En nuestro caso vamos a seleccionar Google Sign-In.

Figura 89. Paso 4: Entorno Firebase



Fuente: Propia

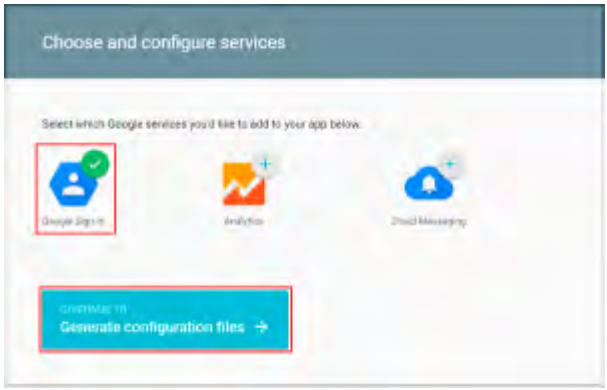
Como ya ocurría con la API de Google Maps, vamos a necesitar indicar la huella SHA-1 del certificado con el que firmamos la aplicación. En el primer capítulo sobre mapas ya vimos cómo obtener este dato. Introducimos la clave SHA-1 y pulsamos «Enable Google Sign-In».

Figura 90. Paso 5: Entorno Firebase

Fuente: Propia

Tras esto ya deberíamos tener correctamente añadido nuestro servicio al proyecto (aparecerá una check verde sobre el icono de Google Sign-In, por lo que podemos finalizar pulsando sobre «Generate configuration files».

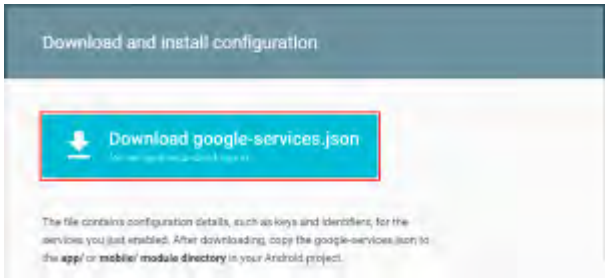
Figura 91. Paso 6: Entorno Firebase



Fuente: Propia

Y ésta es precisamente la principal diferencia entre lo que nos proporciona este asistente y el que ya vimos al crear el proyecto para Google Maps. Ahora tenemos la posibilidad de descargar un fichero de configuración (llamado *google-services.json*) que contiene todo lo necesario para configurar nuestra aplicación Android para asociarla con este proyecto que acabamos de crear. En breve veremos cómo, pero por el momento descarguemos el fichero que se nos ofrece.

Figura 92. Paso 7: Entorno Firebase

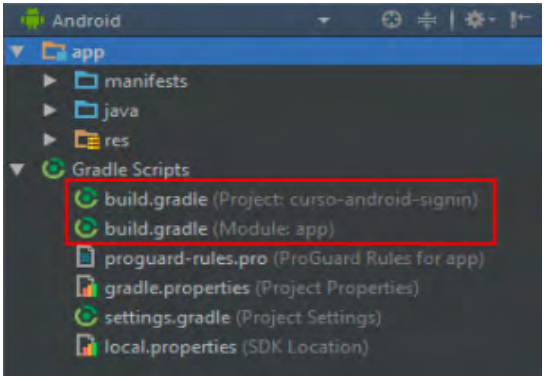


Fuente: Propia

Hecho esto ya podemos ir por fin a nuestra herramienta de desarrollo principal, Android Studio. Vamos a crear un proyecto normal, utilizando la plantilla «*Empty Activity*» y todas las demás opciones por defecto. Y lo primero que vamos a hacer una vez creado es colocar el fichero de configuración «*google-services.json*» que hemos descargado anteriormente dentro de la carpeta «/app» de nuestro proyecto. Para ello puede resultarnos cómodo pulsar botón derecho sobre la carpeta «app» del proyecto en Android Studio y usar la opción «*Show in Explorer*» para ir directamente a dicha carpeta en el explorador de archivos de Windows.

Bien, ya tenemos el fichero de configuración colocado en la carpeta correcta, ¿pero cómo vamos a conseguir que Android Studio lo «consulte» para configurar automáticamente nuestra aplicación? Para esto vamos a añadir el plugin de Google Services para Gradle en nuestro proyecto. Esto se consigue añadiendo un par de lineas a los ficheros *build.gradle* de nuestro proyecto, tanto el situado a nivel de proyecto como el del módulo principal.

Figura 93. Paso 8: Entorno Firebase



Fuente: Propia

En el primero de ellos (nivel de proyecto) añadiremos la siguiente cláusula classpath al final de la sección de dependencias:

```
dependencies {
 //...
 classpath 'com.google.gms:google-services:3.0.0'
}
```

En el segundo (nivel de módulo) añadiremos al final del fichero la siguiente instrucción apply plugin:

```
apply plugin: 'com.google.gms.google-services'
```

Aprovecharemos que estamos en este fichero para añadir también, como ya es habitual, la referencia a la librería específica de autenticación de Google Play Services a la sección de dependencias:

```
dependencies {
 //...
 compile 'com.google.android.gms:play-services-auth:9.4.0'
}
```

Con esto ya habríamos terminado con los preparativos, por lo que podemos empezar a escribir nuestra aplicación. En este caso, la aplicación de ejemplo que vamos a crear será muy sencilla. Constará únicamente de tres botones, el primero de ellos será el de login con Google, y los otros dos nos servirán para hacer logout y para revocar o desconectar completamente la cuenta utilizada de la aplicación. Adicionalmente añadiremos un par de campos de texto para mostrar el nombre e email del usuario logueado cuando aplique. Veamos cómo quedaría el layout:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="net.sgoliver.android.signin.MainActivity">

   <LinearLayout android:id="@+id/lytBotones"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:orientation="horizontal">

      <com.google.android.gms.common.SignInButton
         android:id="@+id/sign_in_button"
         android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/sign_out_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/sign_out" />

    <Button
        android:id="@+id/revoke_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/desconectar" />

  </LinearLayout>

  <TextView android:id="@+id/txtNombre"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/lytBotones" />

  <TextView android:id="@+id/txtEmail"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/txtNombre" />

</RelativeLayout>
```

Nos dirigiremos ahora a nuestra clase java principal. Lo primero que tendremos que hacer será crear un objeto de tipo GoogleApiClient, de forma similar a como ya vimos en el apartado sobre localización, al que añadiremos la API de Google Sign-In. Pero en este caso vamos a dar un paso más antes de eso. Vamos

a construir un objeto de tipo GoogleSignInOptions, con el que vamos a configurar la información que queremos recuperar del usuario que se identifique. Lo construiremos a través de su Builder pasándole como parámetro la opción GoogleSignInOptions.DEFAULT_SIGN_IN que nos asegura la recuperación de la información básica del usuario. Sobre éste podremos llamar a otros métodos requestXYZ() para solicitar que se recupere información adicional. En nuestro caso solicitaremos también el email. El objeto GoogleSignInOptions construido lo pasaremos como parámetro en el método addApi() al construir nuestro *api client*. Todo esto podemos hacerlo dentro del método onCreate() de la actividad principal.

```
GoogleSignInOptions gso =
   new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_
SIGN_IN)
      .requestEmail()
      .build();


apiClient = new GoogleApiClient.Builder(this)
   .enableAutoManage(this, this)
   .addApi(Auth.GOOGLE_SIGN_IN_API, gso)
   .build();
```

Una vez más, como ya hicimos en el caso de la API de localización, indicamos mediante enableAutoManage() que queremos que se gestione automáticamente la conexión con los Google Play Services. Tendremos que implementar por tanto en nuestra clase la interfaz OnConnectionFailedListener y definir el método onConnectionFailed().

```
@Override
public void onConnectionFailed(ConnectionResult connectionResult) {
```

```
    Toast.makeText(this, "Error de conexion!", Toast.LENGTH_SHORT).
show();
    Log.e("GoogleSignIn", "OnConnectionFailed: " + connectionResult);
}
```

Esta vez, por simplicidad, no vamos a añadir los *callback* de conexión mediante addConnectionCallbacks(), pero podríamos hacerlo exactamente igual que para la API de localización.

A continuación podemos personalizar un poco la apariencia del botón de login. En concreto podremos cambiar su tamaño y su esquema de colores.

Para cambiar su tamaño podemos llamar a su método setSize() pasándole como parámetro alguno de los valores siguientes:

- SignInButton.SIZE_STANDARD. Tamaño normal (por defecto).
- SignInButton.SIZE_WIDE. Tamaño grande.
- SignInButton.SIZE_ICON_ONLY. Tamaño pequeño (solo el icono).

Para cambiar su esquema de colores podemos llamar a setColorScheme() pasándole como parámetro uno de los siguientes valores:

- SignInButton.COLOR_LIGHT. Colores claros (por defecto).
- SignInButton.COLOR_DARK. Colores oscuros.

También es posible cambiar la apariencia del botón dependiendo de la información que hemos solicitado recuperar del usuario logueado. Así, por ejemplo, si en el objeto GoogleSignInOptions añadiéramos el *scope* de Google+ (mediante requestScopes())para recuperar información de los círculos del usuario, el

botón se mostraría con el diseño de Google+ y no con el genérico de Google. Esto lo conseguimos llamando al método setScopes() del botón, pasándole el alcance de información solicitado en nuestro objeto GoogleSignInOptions.

```
btnSignIn.setSize(SignInButton.SIZE_STANDARD);
btnSignIn.setColorScheme(SignInButton.COLOR_LIGHT);
btnSignIn.setScopes(gso.getScopeArray());
```

Y vamos por fin por la parte más importante, ¿qué hacer cuando el usuario pulse el botón de login de Google? Pues realmente será muy sencillo al estar todo el proceso bastante automatizado. En primer lugar llamaremos al método getSignInIntent() de la API de autenticación pasándole nuestro cliente API creado anteriormente. El *intent* obtenido lo usaremos para iniciar una nueva actividad (a través de startActivityForResult()), que será la encargada de solicitar al usuario la cuenta con la que quiere identificarse, si ya está registrada en el sistema, o bien añadir una nueva cuenta para la ocasión. Una vez seleccionada una cuenta realizará las comprobaciones de seguridad correspondientes y nos devolverá el resultado del proceso, que podremos gestionar en el método onActivityResult() de la actividad principal, filtrando como siempre por la misma constante arbitraria que hayamos utilizado en la llamada a startActivityForResult().

```
btnSignIn = (SignInButton)findViewById(R.id.sign_in_button);
btnSignIn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent signInIntent = Auth.GoogleSignInApi.getSignInIntent(apiClient);
        startActivityForResult(signInIntent, RC_SIGN_IN);
```

```
    }
});

//...

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == RC_SIGN_IN) {
        GoogleSignInResult result =
            Auth.GoogleSignInApi.getSignInResultFromIntent(data);

        handleSignInResult(result);
    }
}
```

Como podéis observar en el código anterior, el resultado del proceso lo obtenemos llamando a getSignInResultFromInten() a partir del intent recibido como parámetro. Y en nuestro caso, gestionaremos los posibles resultados llamando a un método auxiliar handleSignInResult(). En este método auxiliar revisaremos si el resultado del login ha sido correcto llamando a isSuccess(), en cuyo caso ya podremos obtener algunos datos del usuario logueado, o si bien no se ha finalizado el proceso de login por cualquier motivo. En cualquiera de los dos casos actualizaremos la interfaz de la aplicación según corresponda (por ejemplo ocultando el botón de login y mostrando los de logout si la identificación ha sido correcta) para lo que en nuestro caso usaremos otro método auxiliar updateUI().

```
private void handleSignInResult(GoogleSignInResult result) {
    if (result.isSuccess()) {
        //Usuario logueado --> Mostramos sus datos
        GoogleSignInAccount acct = result.getSignInAccount();
        txtNombre.setText(acct.getDisplayName());
        txtEmail.setText(acct.getEmail());
        updateUI(true);
    } else {
        //Usuario no logueado --> Lo mostramos como "Desconectado"
        updateUI(false);
    }
}

private void updateUI(boolean signedIn) {
    if (signedIn) {
        btnSignIn.setVisibility(View.GONE);
        btnSignOut.setVisibility(View.VISIBLE);
        btnRevoke.setVisibility(View.VISIBLE);
    } else {
        txtNombre.setText("Desconectado");
        txtEmail.setText("Desconectado");

        btnSignIn.setVisibility(View.VISIBLE);
        btnSignOut.setVisibility(View.GONE);
        btnRevoke.setVisibility(View.GONE);
    }
}
```

Y sólo nos queda comentar cómo acceder a los datos del usuario una vez que éste se ha logueado con éxito. Como vemos en el fragmento de código anterior también es muy sencillo, basta con obtener la cuenta activa mediante getSignInAccount() y obte-

ner los datos que necesitemos mediante su método *getXYZ()* correspondiente, por ejemplo getDisplayName() para obtener el nombre público, o getEmail() para el correo electrónico.

Vamos a ver por último cómo implementar los botones de *Sign Out* y *Revocar*. La diferencia entre ambas opciones radica en que «sign out» simplemente desloguea al usuario de la aplicación, y «revocar» no solo nos deslogueará sino que además eliminará cualquier asociación de la cuenta del usuario con nuestra aplicación.

Para ambos el proceso será análogo, diferenciándose tan sólo en el método que llamaremos inicialmente. En el caso de *Sign Out* llamaremos al método signOut() de la API de autenticación, lo que iniciará un proceso asíncrono cuyo resultado obtendremos asignando un *callback* al objeto PendingResult que nos devuelve. En este callback definiremos el método onResult() que se llamará cuando el proceso haya finalizado. En nuestro caso simplemente actualizaremos los botones de la interfaz en consecuencia llamando de nuevo a updateUI(). Por su parte, en el caso de revocar el acceso, el método al que llamaremos será revokeAccess().

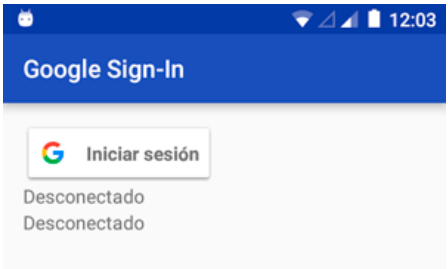
```
btnSignOut.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    Auth.GoogleSignInApi.signOut(apiClient).setResultCallback(
      new ResultCallback<Status>() {
        @Override
        public void onResult(Status status) {
          updateUI(false);
        }
      });
  }
```

```
});

btnRevoke.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
     Auth.GoogleSignInApi.revokeAccess(apiClient).setResultCallback(
        new ResultCallback<Status>() {
          @Override
          public void onResult(Status status) {
             updateUI(false);
          }
        });
  }

});
```

En principio, con esto habríamos finalizado nuestra aplicación. Mi recomendación en este caso sería ejecutarla en un dispositivo real para poder verificar toda la funcionalidad, ya que es posible que en el emular no tengáis configuradas varias cuentas de usuario o el proceso de creación de una sea más laborioso. En cualquier caso no habría problema en hacerlo a través del emulador.

Al ejecutar la aplicación, la pantalla inicial nos mostrará tan sólo el botón de login de Google, y los dos campos de texto de información del usuario indicando que aún no se encuentra ninguno conectado.

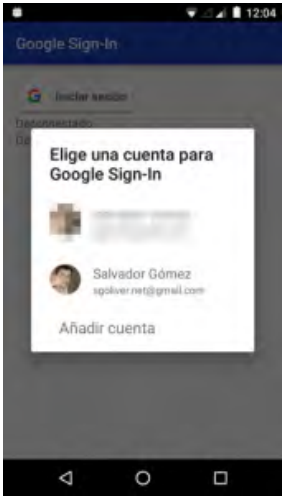
Figura 94. Parte 1: Funcionamiento de Firebase Authentication



Fuente: Propia

Al pulsar sobre el botón de login la aplicación nos mostrará las cuentas de Google registradas ya en el sistema y la opción de crear una nueva si se necesita. En mi caso seleccionaré una de las cuentas existentes.

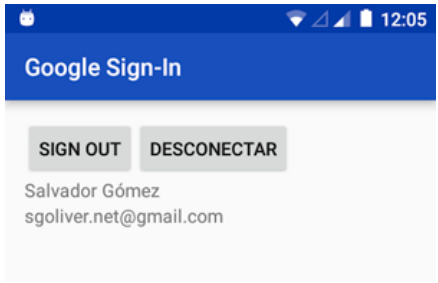
Figura 95. Parte 2: Funcionamiento de Firebase Authentication



Fuente: Propia

Al hacerlo, el sistema hará las comprobaciones necesarias y en caso de no encontrar ningún problema se mostrará la información del usuario logueado y los botones de sign out y revocar acceso (desconectar):

Figura 96. Parte 3: Funcionamiento de Firebase Authentication



Fuente: Propia

Sin embargo hay un problema. Si cerramos la aplicación y volvemos a abrir comprobaremos que tendremos que pulsar de nuevo el botón de *Iniciar Sesión*, aunque ya no nos pedirá seleccionar la cuenta de usuario, sino que se logueará directamente con el que ya elegimos.

Si queremos evitar este problema, es decir, si queremos que cuando el usuario vuelva a la aplicación no tenga que pulsar de nuevo sobre el botón de iniciar sesión sino que se conecte automáticamente con el usuario que ya lo había hecho anteriormente (por supuesto salvo que cerrara sesión la última vez) tendremos que implementar esta funcionalidad de forma explícita en nuestra aplicación.

Este proceso de login automático recibe el nombre de *Silent Sign-In*, y para implementarlo aprovecharemos el evento onStart() de nuestra actividad principal. En este evento, lanzado cuando se inicia la actividad, tendremos que llamar al método silentSignIn() de la API de autenticación pasándole como siempre

una referencia a nuestro cliente API. Esta llamada nos devuelve un objeto OptionalPendingResult que utilizaremos de la siguiente forma: llamaremos a su método isDone() que nos indicará si ya existen credenciales del usuario «cacheadas» y éstas son válidas. En este caso podremos obtener el resultado de forma inmediata y lo gestionaremos llamando a nuestro método auxiliar handleSignInResult(), igual que en el proceso de login normal ya comentado. En caso contrario (es decir, si el usuario no ha iniciado sesión anteriormente en el dispositivo o si la sesión ha expirado o se ha cerrado) se iniciará un proceso asíncrono en el que se intentará loguear al usuario de forma silenciosa por otros medios (por ejemplo, si ha iniciado sesión en la misma aplicación pero en otro dispositivo u otra plataforma, aunque esto se sale del alcance de este artículo). Al tratarse de un proceso asíncrono que puede llevar algún tiempo es recomendable mostrar un indicador de progreso mientras se obtiene el resultado, lo que conseguiremos mostrando un ProgressDialog de tipo indeterminado. El resultado del proceso lo obtendremos una vez más definiendo el método onResult() del ResultCallback que asignaremos al objeto OptionalPendingResult. En caso de no conseguirse iniciar sesión de forma automática, se acabará mostrando el botón de login al usuario para que lo haga de forma manual. Veamos cómo quedaría el código:

```
@Override
protected void onStart() {
   super.onStart();


    OptionalPendingResult<GoogleSignInResult> opr = Auth.GoogleSignInApi.
silentSignIn(apiClient);
   if (opr.isDone()) {
      GoogleSignInResult result = opr.get();
      handleSignInResult(result);
```

```
    } else {
        showProgressDialog();
        opr.setResultCallback(new ResultCallback<GoogleSignInResult>() {
            @Override
            public void onResult(GoogleSignInResult googleSignInResult) {
                hideProgressDialog();
                handleSignInResult(googleSignInResult);
            }
        });
    }
}

private void showProgressDialog() {
    if (progressDialog == null) {
        progressDialog = new ProgressDialog(this);
        progressDialog.setMessage("Silent SignI-In");
        progressDialog.setIndeterminate(true);
    }

    progressDialog.show();
}
```

Podemos probar ahora de nuevo la aplicación para comprobar que mientras no cerremos sesión el usuario conseguirá loguearse de forma automática cuando volvamos a la aplicación tras haberla cerrado.

Y con esto sí habríamos terminado con la configuración básica del proceso de login en nuestras aplicaciones mediante el uso de una cuenta de Google.

Firebase Realtime Database

3.2 Firebase Realtime Database

Firebase es una plataforma de desarrollo web y móvil que nos proporciona gran parte de la infraestructura «de servidor» o de *backend* que podamos necesitar. Nos ofrece una serie de servicios clave para nuestras aplicaciones, como pueden ser:

- Base de datos en tiempo real (*Realtime database*)
- Sistema de autenticación (*Authentication*)
- Mensajería y Notificaciones (*Cloud Messaging*)
- Almacenamiento (*Storage*)
- Estadísticas de uso (*Analytics*)
- Reporte de errores (*Crash Reporting*)
- Publicidad (*AdMob*)

Adicionalmente, también proporciona otros servicios más «modestos» pero no menos interesantes como por ejemplo el Hospedaje web (*Hosting*), la Configuración remota (*Remote Config*), la Indexación de aplicaciones (*App Indexing*)… en la web oficial de Firebase puedes comprobar el listado completo de servicios disponibles.

Muchos de estos servicios se ofrecen de forma gratuita, al menos dentro de unos determinados límites de uso, como parte de su modalidad de suscripción llamada «SPARK». Si nuestros requerimientos superan dichos límites gratuitos es posible adherirse a otros modelos de suscripción de pago, «FLAME» y «BLAZE», el primero de ellos con unos límites más altos que el modelo «SPARK», y el segundo sin límites (pago por uso).

En los próximos artículos del curso vamos a ver cómo utilizar algunos de estos servicios desde nuestras aplicaciones Android. Iré actualizando el siguiente listado a modo de índice temático para todos los capítulos del curso relacionados con Firebase.

 Firebase para Android: Base de Datos en Tiempo Real (I)

La base de datos en tiempo real de Firebase (Firebase Realtime Database) es sin duda uno de los servicios más populares de la plataforma. Contar con la capacidad de almacenar datos «en la nube» es uno de los requerimientos de los que pocas aplicaciones actuales pueden escapar, y poder hacerlo sin necesidad de preocuparnos por toda la infraestructura de servidor necesaria es toda una ventaja.

Firebase nos proporciona un servicio de base de datos con la particularidad de ser *en tiempo real*. ¿Pero qué significa esto? La sincronización en tiempo real implica que cualquier cambio realizado en los datos por cualquier cliente (usuario, aplicación, dispositivo…) se sincronizarán automáticamente y de forma inmediata (siempre que la conexión lo permita) en el resto de clientes, sin necesidad de que éstos vuelvan a consultar los datos. ¿Y si se pierde temporalmente la conexión? No hay problema, Firebase también está preparado para permitir interactuar con la base de datos cuando el dispositivo no tiene conexión (siempre dentro de unos límites) mediante un sistema de cachés y colas de escritura locales. Cuando el dispositivo vuelve a tener conexión, los cambios locales serán sincronizados automáticamente con la base de datos y, si aplica, con el resto de clientes conectados a ella.

En Firebase, los datos estarán estructurados de forma similar a cómo se organiza la información en un fichero JSON, es de-

cir, en forma de árbol donde cada nodo puede contener un valor o bien contener nodos hijo, que a su vez podrán contener valores o tener nuevos nodos hijo… (hasta un máximo de 32 niveles de anidación). Éste es un aspecto importante a tener en cuenta, Firebase no nos ofrece una base de datos SQL tradicional, con sus tablas y sus registros perfectamente estructurados, sino un modelo de datos jerarquizado, tipo JSON, al que podremos acceder de forma remota. Si acabas de llevarte una decepción, no temas, este modelo es más que suficiente para la mayoría de las aplicaciones. Te recomiendo que sigas adelante con el curso.

En los próximos artículos vamos a describir cómo preparar y configurar un proyecto Android para hacer uso de una base de datos de Firebase, cómo realizar las acciones más habituales desde nuestra aplicación, y cómo administrar algunos aspectos importantes de la base de datos. Empecemos.

Al igual que ocurría con muchos de los servicios de Google Play Services, antes de ponernos a trabajar con Firebase desde un proyecto de Android Studio tendremos dirigirnos a la consola de desarrolladores, en esta ocasión la Consola de Firebase, para crear y configurar el nuevo proyecto y asociar a él nuestra aplicación.

Si es la primera vez que accedemos a la consola de Firebase, nos aparecerá un mensaje de bienvenida que directamente nos invita a empezar a crear un nuevo proyecto.

Si pulsamos la opción de "CREAR NUEVO PROYECTO" el sistema nos solicitará un nombre identificativo y la región a la que perteneces. Con estos simples datos quedaría creado el nuevo proyecto de Firebase. El siguiente paso será asociar una aplicación a dicho proyecto, en nuestro caso una aplicación Android, para lo que pulsaremos en la opción correspondiente a dicho sistema.

Esto iniciará un pequeño asistente donde se nos solicitarán algunos datos que ya deberían sonarnos de artículos anteriores. Tendremos que indicar el paquete java principal de nuestra aplicación Android, yo usaré net.sgoliver.android.fcm, y la huella digital SHA-1 del certificado con el que se firmará la aplicación, en principio usaremos como siempre la correspondiente al certificado de pruebas (tienes más información sobre cómo obtener este dato por ejemplo en el primer artículo sobre mapas), pero recuerda cambiarlo por el certificado de producción si subes tu aplicación a Google Play.

En el segundo paso podrás descargar un fichero de configuración, en formato JSON, que tendremos que añadir a la aplicación Android una vez creemos el proyecto en Android Studio. Al pulsar el botón CONTINUAR se descargará el fichero, más adelante veremos qué hacer con él (aunque ya hicimos algo muy similar cuando tratamos el tema de autenticación mediante Google Sign-In).

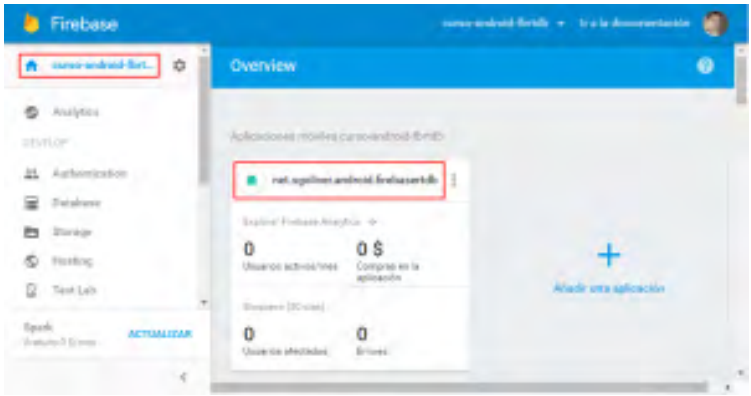
En el tercer y último paso se ofrecen una serie de indicaciones sobre cómo configurar todas las dependencias necesarias en el proyecto de Android Studio. Esto lo veremos más adelante con más detalle, por lo que ya podemos FINALIZAR el asistente.

Figura 97. Paso 1: Funcionamiento de Firebase Realtime Database

Fuente: Propia

Con esto ya tendríamos configurado todo lo necesario en la consola de Firebase. En este momento deberíamos estar viendo tanto nuestro proyecto (en la parte superior izquierda) como nuestra aplicación Android asociada (en la parte central). No debemos confundir una entidad con otra. Un mismo proyecto de Firebase puede tener asociadas varias aplicaciones, que además pueden ser de Sistemas distintos (Android, IOS o Web).

Figura 98. Paso 2: Funcionamiento de Firebase Realtime Database



Fuente: Propia

Finalizados todos los preparativos en la consola de Firebase, podemos disponernos ya a crear nuestro proyecto en Android Studio. Crearemos un proyecto estándar, con API mínima 15 y utilizando la plantilla *Empty Activity*.

Creado el proyecto lo primero que vamos a hacer es colocar el fichero de configuración "*google-services.json*" que hemos descargado antes en su ubicación correcta. Debemos copiarlo a la carpeta "/app" situada dentro de la carpeta de nuestro proyecto. En mi caso particular he llamado al proyecto "android-firebase-rtdb-1", por lo que el fichero de configuración debe ir colocado en la carpeta "\android-firebase-rtdb-1\app".

El siguiente paso será añadir todas las dependencias necesarias a nuestros ficheros *build.gradle*. Tendremos que modificar tanto el fichero *build.gradle* situado a nivel de proyecto, como el de nuestro módulo principal (¿no sabes cuáles son estos ficheros? Consúltalo aquí).

Empezaremos por el *build.gradle* de proyecto. Aquí tendremos que añadir a la sección de dependencias la referencia al plugin de Google Play Services para Gradle.

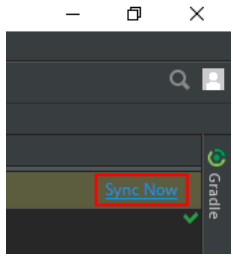
```
buildscript {
   repositories {
      jcenter()
   }
   dependencies {
      //...
      classpath 'com.google.gms:google-services:3.0.0'
   }
}
```

Por su parte, en el *build.gradle* del módulo principal, tendremos por un lado que aplicar dicho plugin al final del fichero, y por otro añadir la referencia a los servicios de Firebase que vayamos a utilizar. En nuestro caso utilizaremos la funcionalidad de base de datos en tiempo real, por lo que añadiremos tan solo su librería correspondiente (puedes consultar el listado completo de librerías disponibles en la documentación oficial).

```
//...
 dependencies {
   //...
   //compile 'com.google.firebase:firebase-messaging:9.6.1'
   compile 'com.google.firebase:firebase-database:9.6.1'
}
 apply plugin: 'com.google.gms.google-services'
```

Después de modificar cada fichero de Gradle recordad que es necesario sincronizar los cambios con el proyecto. Para ello, podemos utilizar la opción "*Sync Now*" que aparece en la zona superior derecha del editor cuando modificamos cualquier fichero de configuración de Gradle.

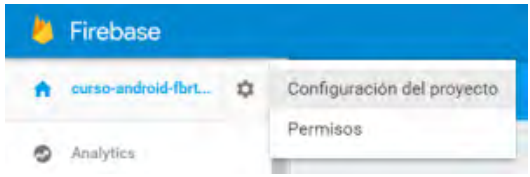
Figura 99. Paso 3: Funcionamiento de Firebase Realtime Database



Fuente: Propia

En alguna ocasión me he encontrado con la sorpresa de recibir errores tras finalizar la configuración del proyecto de Android Studio tal como lo hemos hecho hasta ahora. El motivo era que el fichero «*google-services.json*» descargado desde la consola de Firebase durante el asistente de configuración no era correcto o estaba incompleto. Si sospechas que algún error mostrado por Android Studio en este momento podría ser debido a este motivo, siempre puedes volver a descargar el fichero de configuración desde la consola de Firebase, accediendo a la configuración del proyecto.

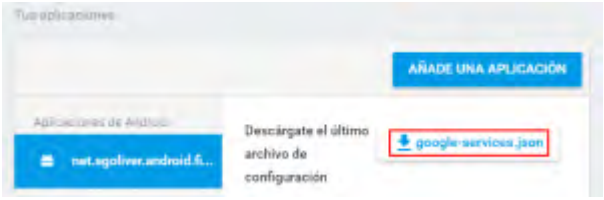
Figura 100. Paso 4: Funcionamiento de Firebase Realtime Database



Fuente: Propia

Seleccionando la aplicación Android asociada al proyecto dentro del apartado «*Tus aplicaciones*» tendremos la posibilidad de descargar de nuevo su fichero de configuración, lo que suele solucionar el problema mencionado:

Figura 101. Paso 5: Funcionamiento de Firebase Realtime Database



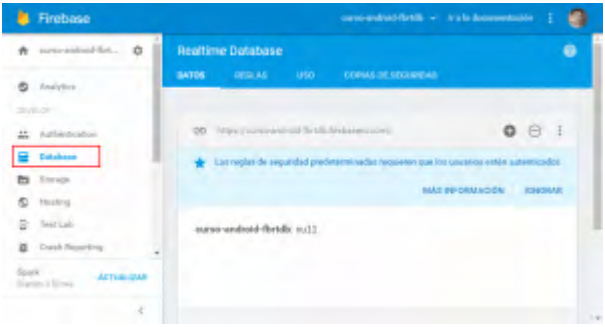
Fuente: Propia

Con esto ya tendríamos finalizada la configuración básica de nuestro proyecto en la consola de Firebase y en Android Studio, por lo que podríamos empezar a hacer uso de la base de datos.

En este primer artículo veremos las opciones que nos proporciona la propia consola de Firebase para interactuar directamente con la base de datos y en los próximos artículos nos centraremos en cómo consultar y manipular los datos desde la aplicación Android.

Si nos dirigimos a la consola y pulsamos sobre la opción Database del menú lateral izquierdo accederemos al «panel de administración» de nuestra base de datos. A la derecha veremos la información distribuida en 4 pestañas: Datos, Reglas, Uso y Copias de Seguridad. Por el momento tan sólo nos preocuparemos de los dos primeros.

Figura 102. Paso 6: Funcionamiento de Firebase Realtime Database



Fuente: Propia

La primera pestaña nos da acceso a la base de datos como tal, donde podremos añadir, modificar, y eliminar datos, editando directamente el contenido desde la consola.

Por defecto, nuestra base de datos contendrá tan sólo un nodo vacío con el nombre de nuestro proyecto de Firebase. Si colocamos el ratón sobre él tendremos la posibilidad de añadir un nuevo nodo hijo pulsando sobre el símbolo «+«.

Figura 103. Paso 7: Funcionamiento de Firebase Realtime Database

Fuente: Propia

Cada nodo de la base de datos tendrá un *nombre* (o *clave*) y un *valor*, o bien solo el *nombre* en caso de que vaya a servir como nodo padre a otros elementos.

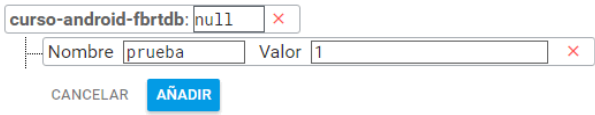
El *valor* de un nodo puede ser de alguno de los siguientes tipos:

- **Numérico**. Ejemplos: 1 (entero), 3.14 (real)
- **Alfanumérico**, entre comillas dobles. Ejemplo: «abc».
- **Booleano**. Valores posibles: true y false.
- **Objeto**, entre llaves. Es una secuencia de pares clave-valor. Ejemplo: {«nombre»:»pepe», «edad»:25}. En la práctica esto sería equivalente a añadir al nodo dos nodos hijo con los nombres y valores indicados en el objeto.
- **Array**, entre corchetes. Ejemplo: [«a», «b», «c»]. Realmente Firebase no soporta de forma nativa los arrays, pero si los utilizamos en el campo *valor* de un nodo los convertirá automáticamente a un objeto, cuyas claves

serán los índices del array, y como valores los indicados entre los corchetes.

Como primer ejemplo vamos a crear un nodo con clave «prueba» y valor numérico «1»:

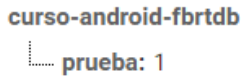
Figura 104. Paso 8: Funcionamiento de Firebase Realtime Database



Fuente: Propia

Pulsando el botón «Añadir» se aplicará el cambio de forma definitiva, añadiéndose el nuevo nodo a la base de datos.

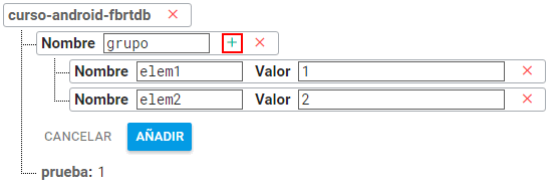
Figura 105. Paso 9: Funcionamiento de Firebase Realtime Database



Fuente: Propia

Añadamos ahora un nuevo nodo que tendrá varios elementos hijo. Para ello volvemos a colocarnos encima del nodo raíz y pulsamos el símbolo «+» para añadir un nuevo elemento hijo. Le ponemos un nombre, por ejemplo «grupo», y sin indicar ningún valor pulsamos sobre su botón «+» para añadir sus elementos hijos, a los que en mi caso de ejemplo llamaré «elem1» y «elem2» y les asignaré valores arbitrarios «1» y «2».

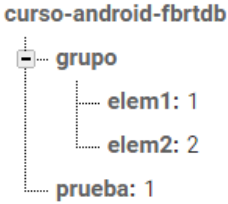
Figura 106. Paso 10: Funcionamiento de Firebase Realtime Database



Fuente: Propia

Pulsando nuevamente el botón «Añadir» quedarían aplicados los cambios realizados:

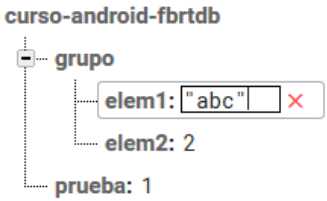
Figura 107. Paso 11: Funcionamiento de Firebase Realtime Database



Fuente: Propia

Para editar cualquier nodo no tenemos más que colocarnos sobre él, editar su valor (en caso de ser un nodo con valor asignado), y pulsar la tecla Intro en nuestro teclado.

Figura 108. Paso 12: Funcionamiento de Firebase Realtime Database



Fuente: Propia

Si no es un nodo final, es decir, si contiene subnodos, también podemos añadirle nuevos nodos hijo colocándonos sobre él y pulsando de nuevo el botón «+«.

Figura 109. Paso 13: Funcionamiento de Firebase Realtime Database

Fuente: Propia

Por último, para eliminar cualquier nodo tan sólo tendremos que pulsar el botón «**x**» que aparece a la derecha al colocarnos sobre él. Debemos tener en cuenta que al eliminar un nodo también se eliminarán todos los elementos que cuelguen de él. De cualquier forma, recibiremos una advertencia para recordárnoslo cada vez que eliminemos un nodo.

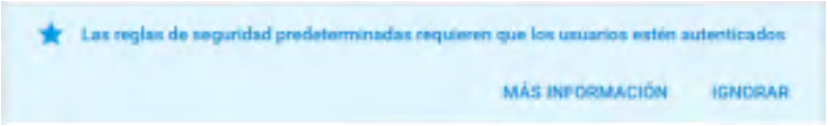
Figura 110. Paso 14: Funcionamiento de Firebase Realtime Database

Fuente: Propia

Y no hay más, así de sencillo sería manipular la estructura y los datos de nuestra base de datos Firebase.

¿Pero habéis notado el mensaje de advertencia mostrado sobre el editor de datos? Nos indica: «*Las reglas de seguridad predeterminadas requieren que los usuarios estén autenticados*«. ¿Qué significa esto?

Figura 111. Paso 15: Funcionamiento de Firebase Realtime Database



Fuente: Propia

Firebase permite definir una serie de reglas de seguridad para determinar quiénes y de qué forma podrán acceder a los datos de nuestra base de datos. Aquí es donde entra en juego la segunda pestaña («Reglas») de la sección de base de datos de nuestro proyecto. Si accedemos a ella podremos ver qué reglas están definidas actualmente.

Figura 112. Paso 16: Funcionamiento de Firebase Realtime Database

Fuente: Propia

Por defecto, Firebase establece como regla de seguridad que tan sólo aquellos usuarios que estén autenticados en la aplicación podrán leer y escribir en nuestra base de datos.

```
{
 "rules": {
  ".read": "auth != null",
  ".write": "auth != null"
 }
}
```

No es necesario que entendamos por ahora la forma de definir estas reglas, ya volveremos a este tema más adelante, tan solo pretendía que conociérais la existencia de las mismas y por qué vamos a cambiarlas de forma temporal.

De cara a no complicar en exceso estos artículos iniciales sobre la base de datos de Firebase entrando en temas de auten-

tication, vamos a modificar temporalmente las reglas de acceso para permitir la lectura y escritura de datos a cualquier usuario. Para ello editaremos el apartado de reglas para que quede de la siguiente forma:

```
{
 "rules": {
   ".read": true,
   ".write": true
 }
}
```

Tras editar las reglas tendremos que aplicar los cambios pulsando sobre el botón «PUBLICAR».

Figura 113. Paso 17: Funcionamiento de Firebase Realtime Database

Fuente: Propia

A partir de ahora nos aparecerá una nueva advertencia indicando que las reglas establecidas permiten el acceso a los datos a cualquier usuario.

Figura 114. Paso 18: Funcionamiento de Firebase Realtime Database

Fuente: Propia

Supongo que por el momento no vais a guardar secretos de estado en la base de datos, por lo que entiendo que éste es un riesgo asumible, y además nos facilitará un poco las cosas mientras damos nuestros primeros pasos con Firebase. Más adelante veremos como restringir el acceso a los datos de forma más adecuada.

Y por ahora vamos a dejarlo aquí. Hoy hemos descrito los preparativos necesarios para crear un proyecto Android con Firebase y hemos repasado las opciones básicas que nos proporciona la propia consola de Firebase en lo que al servicio de base de datos se refiere. En el siguiente artículo empezaremos ya a ver en detalle la forma de interacturar con la base de datos desde nuestro proyecto Android.

Guardad el proyecto de Android Studio que hemos creado porque lo utilizaremos de base para los próximos artículos.

3.2.1 Firebase para Android: Base de Datos en Tiempo Real (II)

En el artículo anterior del curso repasamos todos los preparativos necesarios para empezar a trabajar en un proyecto Android con soporte para los servicios proporcionados por Firebase, centrándonos en esta ocasión en la base de datos en tiempo real (realtime database). Por un lado vimos cómo crear el proyecto

de Firebase desde su consola de administración, y por otro cómo crear y configurar el proyecto Android en Android Studio. Por último explicamos algunas particularidades de la base de datos de Firebase, como por ejemplo su organización de los datos en forma de árbol JSON, y cómo manipular dichos datos desde la propia consola. En este nuevo artículo nos centraremos ya en el proyecto Android y veremos las distintas formas que tenemos disponibles para leer los datos de la base de datos y mostrarlos en nuestra aplicación.

Cuando hablamos de las características de la base de datos de Firebase hubo un punto que destacamos sobre el resto, y era su sincronización en tiempo real. Aunque ya explicamos brevemente qué significaba esto voy a intentar explicarlo de nuevo comparando la base de datos de Firebase con una base de datos tradicional.

Cuando utilizamos una base de datos tradicional como backend de una aplicación Android es muy probable que nos encontremos con un problema similar a éste: imaginemos que todos los clientes conectados a la base de datos leen de ella una determinada información y la muestran al usuario. En algún momento del tiempo uno de los clientes modifica dicha información, la envía al servidor, y se actualiza en la base de datos. Tras esto, ¿cómo se enteran de los cambios el resto de clientes que usan la base de datos? Algunas de las soluciones típicas ante estas situaciones son las siguientes:

- Dotar a las aplicaciones de alguna opción de refresco para que, bien sea de forma manual por el usuario o automáticamente de forma periódica, la aplicación vuelva a consultar los datos al servidor y obtenga la nueva información si ésta ha cambiado.

- Implementar algún sistema de *notificaciones push*, de forma que cada vez que un cliente actualiza información relevante en la base de datos, el resto de clientes conectados sean notificados automáticamente para que vuelvan a recuperar dicha información del servidor.

Cualquiera de estas soluciones es válida y pueden ser suficiente en muchos casos, pero implican que en un momento u otro todos los clientes deben volver a consultar de forma explícita al servidor para ver si la información ha cambiado, con el trabajo de codificación que ello conlleva, más el trabajo adicional de tener que utilizar otras tecnologías en paralelo como las notificaciones push de la segunda opción.

Cuando utilizamos la base de datos en tiempo real de Firebase la estrategia de obtención de información será distinta. Ya no necesitaremos que la aplicación consulte en distintos momentos un determinado dato por si éste ha cambiado, sino que directamente nos *suscribiremos* a dicho dato de forma que seamos notificados y recibamos su nuevo valor automáticamente cada vez que éste cambie. Podríamos decir de alguna forma que con las bases de datos tradicionales la iniciativa la debe llevar siempre la aplicación, y con Firebase gran parte de esa iniciativa pasa al lado de la base de datos. De esta forma, cualquier cambio que se produzca en los datos (por ejemplo cuando un cliente actualiza parte de la información) se trasladará de forma automática e inmediata a todos los clientes interesados en dicho dato, sin necesidad de implementar este mecanismo de notificación y refresco por nuestra parte.

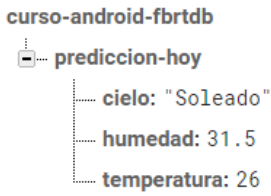
¿Pero cómo hacemos esto? ¿Realmente es tan sencillo? Sí que lo es, veamos cómo conseguirlo. Vamos a empezar escribiendo desde la consola de Firebase una serie de datos en la base de datos

que podamos leer después desde nuestra aplicación Android. Ya vimos cómo hacer esto en el artículo anterior por lo que no entraré en mucho detalle.

3.2.1.1 Problema

Como ejemplo, imaginemos que estamos construyendo una aplicación para mostrar las condiciones meteorológicas en nuestra ciudad y queremos mostrar el estado del cielo, la temperatura, y la humedad en el día de hoy. Para almacenar esta información podemos crear un nuevo nodo en la base de datos llamado «*prediccion-hoy*» y como subelementos de éste tres nuevos nodos «cielo», «temperatura» y «humedad» con sus valores correspondientes (ficticios, por supuesto):

Figura 115. Parte 1: Problema 3.2.1.1 Realtime Database



Fuente: Propia

Hecho esto ya podemos dirigirnos a nuestro proyecto Android en Android Studio (en el artículo anterior vimos cómo crearlo y configurarlo). Lo primero que haremos será crear una layout muy sencillo para nuestra pantalla principal, donde mostremos los tres datos de la predicción meteorológica. Simplemente añadiré tres campos de texto a los que llamaré lblCielo, lblTemperatura y lblHumedad. Si necesitas consultar el código del layout encontrarás al final del artículo el enlace al código completo del ejemplo en github.

Y con esto terminado vamos ya con Firebase. Lo primero que tendremos que hacer será crear una *referencia* a nuestra base de datos. Esta referencia podría entenderse como un indicador que apunta a un lugar concreto de nuestra base de datos, aquel en el que estemos interesados. Aunque es posible definir una referencia al elemento raíz de la base de datos, casi nunca necesitaremos acceder a la totalidad de los datos, sino que nos bastará con un fragmento o sección concreta, o dicho de otra forma, solo necesitaremos los datos que cuelguen de un determinado nodo de nuestro árbol JSON.

Por empezar por el caso más sencillo, vamos a crear por ejemplo una referencia al nodo «cielo» de forma que podamos suscribirnos a él para conocer su valor actual y estar informados de sus posibles cambios. Para ello crearemos en primer lugar un objeto de tipo DatabaseReference que almacenará la referencia a la base de datos. Para obtener ésta obtendremos primero una instancia a la base de datos de Firebase mediante getInstance() y después una referencia al nodo raíz de los datos con getReference() sin parámetros. A partir de esta primera referencia podemos crear cualquier otra más específica «navegando» entre los nodos hijo mediante el método child(), que recibe como parámetro el nombre del subnodo al que queremos bajar en el árbol. En nuestro caso tendremos que acceder primero al subnodo «prediccion-hoy» y posteriormente a su subnodo «cielo«:

```
DatabaseReference dbCielo =
  FirebaseDatabase.getInstance().getReference()
    .child("prediccion-hoy")
      .child("cielo");
```

Obtenida la referencia ya podríamos suscribirnos a ella asignándole un listener de tipo ValueEventListener. Este listener tendrá dos métodos:

- onDataChange(). Se llamará automáticamente cada vez que se actualice la información del nodo actual o se produzca cualquier cambio en cualquiera de sus nodos descendientes. Se llamará por primera vez en el momento de suscribirnos, de forma que recibamos el valor actual del nodo y todos sus descendientes (si los tiene).
- onCancelled(). Se llamará cuando la lectura de los datos sea cancelada por cualquier motivo, por ejemplo porque el usuario no tiene permiso para acceder a los datos.

El primero de los métodos es por supuesto el más interesante, ya que es el que nos permitirá estar al tanto de cualquier cambio que se produzca en la información contenida a partir de la localización a la que apunta nuestra referencia. Como parámetro de este método recibiremos siempre un objeto de tipo DataSnapshot que contendrá toda la información del nodo. Un objeto DataSnapshot representa básicamente una rama del árbol JSON, es decir, contendrá toda la información de un nodo determinado, con su clave, su valor, y su listado de nodos hijos (que a su vez pueden tener otros descendientes), por el que podremos navegar libremente. Podremos obtener la clave y el valor mediante los métodos getKey() y getValue() respectivamente. Los subnodos los podremos obtener en su totalidad mediante getChildren() (los recibiremos en forma de listado de objetos DataSnapshot) o bien podremos navegar a subnodos concretos mediante child(«nombre-subnodo»).

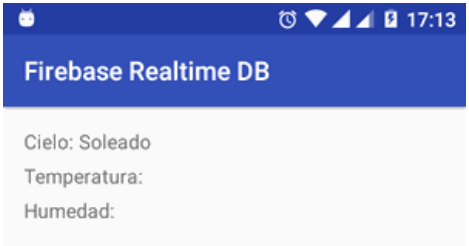
Entendiendo esto, nuestro primer caso de ejemplo sería muy sencillo. Cada vez que se llame al método onDataChange() simplemente recuperaremos el valor del nodo con getValue() y ac-

tualizaremos el campo correspondiente de la interfaz de la aplicación.

```
dbCielo.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        String valor = dataSnapshot.getValue();
        lblCielo.setText(valor);
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
        Log.e(TAGLOG, "Error!", databaseError.toException());
    }
});
```

Si ejecutamos en este momento la aplicación (recomiendo hacerlo en un dispositivo real), podremos comprobar como nuestro campo «Cielo» de la aplicación se informa correctamente con el valor actual almacenado la base de datos (ya que onDataChange() se ejecuta una primera vez al suscribirnos a la referencia definida).

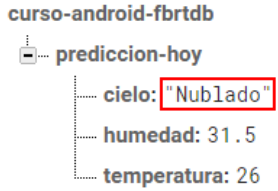
Figura 116. Parte 2: Problema 3.2.1.1 Realtime Database



Fuente: Propia

Si ahora modificamos dicho valor desde la consola de Firebase:

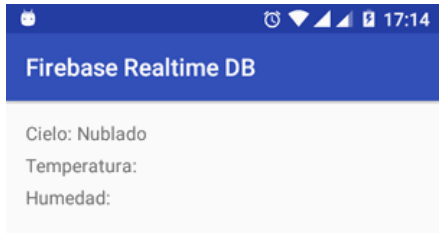
Figura 117. Parte 3: Problema 3.2.1.1 Realtime Database



Fuente: Propia

Nuestra aplicación debería reflejar inmediatamente el cambio:

Figura 118. Parte 4: Problema 3.2.1.1 Realtime Database



Fuente: Propia

Sencillo, ¿verdad? Pues podríamos hacerlo de forma análoga para los otros dos campos de nuestra predicción, temperatura y humedad, y ya tendríamos la aplicación funcionando como pretendíamos. Sin embargo vamos a hacerlo de forma distinta para ver otras alternativas de acceso a los datos de Firebase.

No siempre tenemos que definir referencias a nodos finales del árbol (nodos sin hijos) como hemos hecho con el nodo «cielo». También podemos por supuesto definir referencias a nodos

intermedios del árbol de forma que podamos detectar cambios en cualquiera de sus descendientes asignando un sólo listener ValueEventListener. En nuestro caso particular la solución obvia será crear una referencia al nodo «prediccion-hoy» y suscribirnos a sus cambios. De esta forma, cada vez que cualquiera de los nodos «cielo», «temperatura» o «humedad» sea actualizado recibiremos en el método onDataChange() la información del nodo «prediccion-hoy» completo. Para obtener toda la información de este nodo tendremos dos posibilidades: navegar de forma explícita por sus hijos mediante child() y obteniendo su valor con getValue(), o bien crear un objeto Java con la misma estructura (los mismos campos) y dejar que getValue() trabaje por nosotros convirtiendo la información recibida en un objeto de ese tipo.

La primera opción quedaría más o menos de la siguiente forma:

```
private DatabaseReference dbPrediccion;
private ValueEventListener eventListener;

//...

dbPrediccion =
  FirebaseDatabase.getInstance().getReference()
    .child("prediccion-hoy");

eventListener = new ValueEventListener() {
  @Override
  public void onDataChange(DataSnapshot dataSnapshot) {

    lblCielo.setText(dataSnapshot.child("cielo").getValue().toString());
      lblTemperatura.setText(dataSnapshot.child("temperatura").getVa-
lue().toString());
    lblHumedad.setText(dataSnapshot.child("humedad").getValue().toS-
tring());
  }
```

```
    @Override
    public void onCancelled(DatabaseError databaseError) {
        Log.e(TAGLOG, "Error!", databaseError.toException());
    }
};

dbPrediccion.addValueEventListener(eventListener);
```

Para la segunda tendríamos que crear primero una clase con los mismos campos que el nodo al que nos hemos suscrito. En mi caso crearé una clase llamada Prediccion, con los tres campos indicados, y sus *getters* y *setters*. Es obligatorio incluir además un constructor por defecto para que todo funcione correctamente:

```
public class Prediccion {
    private String cielo;
    private long temperatura;
    private double humedad;

    public Prediccion() {
        //Es obligatorio incluir constructor por defecto
    }

    public Prediccion(String cielo, long temperatura, double humedad)
    {
        this.cielo = cielo;
        this.temperatura = temperatura;
        this.humedad = humedad;
    }

    public String getCielo() {
        return cielo;
```

```
    }

    public void setCielo(String cielo) {
        this.cielo = cielo;
    }

    public long getTemperatura() {
        return temperatura;
    }

    public void setTemperatura(long temperatura) {
        this.temperatura = temperatura;
    }

    public double getHumedad() {
        return humedad;
    }

    public void setHumedad(double humedad) {
        this.humedad = humedad;
    }

    @Override
    public String toString() {
        return "Prediccion{" +
                "cielo='" + cielo + '\'' +
                ", temperatura=" + temperatura +
                ", humedad=" + humedad +
                '}';
    }
}
```

Una vez creada, podríamos hacer que el método getValue() del DataSnapshot nos devuelva directamente un objeto de

este tipo pasándole como parámetro la clase que debe devolver, de la siguiente forma:

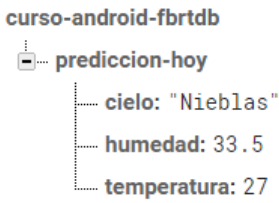
```
private DatabaseReference dbPrediccion;
private ValueEventListener eventListener;

//...

dbPrediccion =
   FirebaseDatabase.getInstance().getReference()
      .child("prediccion-hoy");

eventListener = new ValueEventListener() {
   @Override
   public void onDataChange(DataSnapshot dataSnapshot) {

      Prediccion pred = dataSnapshot.getValue(Prediccion.class);
      lblCielo.setText(pred.getCielo());
      lblTemperatura.setText(pred.getTemperatura() + "ºC");
      lblHumedad.setText(pred.getHumedad() + "%");
   }

   @Override
   public void onCancelled(DatabaseError databaseError) {
      Log.e(TAGLOG, "Error!", databaseError.toException());
   }
};

dbPrediccion.addValueEventListener(eventListener);
```

Por su parte, y por comentarlo brevemente, el método onCancelled() recibe como parámetro un objeto de tipo DatabaseError, que contiene toda la información del error que se haya producido. En nuestro caso de ejemplo me estoy limitando a añadir esta información de error, en forma de excepción llamando

al método toException(), a un mensaje en el log de Android. En la práctica deberíamos mostrar el mensaje de error al usuario y adaptar nuestra interfaz a la posible ausencia de datos, si procede.

Si ejecutamos de nuevo la aplicación y modificamos algunos datos desde la consola de Firebase:

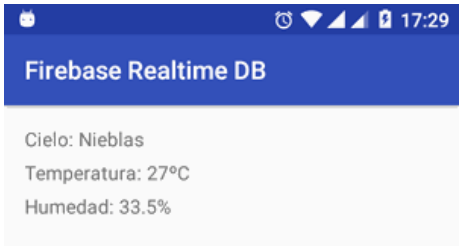
Figura 119. Parte 5: Problema 3.2.1.1 Realtime Database



Fuente: Propia

Deberíamos ver cómo se actualizan convenientemente en la aplicación Android todos los datos de nuestra predicción meteorológica:

Figura 120. Parte 6: Problema 3.2.1.1 Realtime Database



Fuente: Propia

Otro tema importante a tener en cuenta es que la suscripción a una referencia de una base de datos de Firebase, es decir, el hecho de asignar un listener a una ubicación del árbol JSON para estar al tanto de sus cambios no es algo «gratuito» desde el punto de vista de consumo de recursos. Por tanto, es recomendable

eliminar esa suscripción cuando ya no la necesitamos. Para hacer esto basta con llamar al método removeEventListener() sobre cada referencia a la base de datos que ya no sea necesaria y pasarle como parámetro el listener que deseamos eliminar. Para el ejemplo, añadiré un botón que realice esta desvinculación. Tras pulsarlo, si hacemos cualquier cambio en los datos desde la consola de Firebase éste ya no se sincronizará con la aplicación Android.

```
btnEliminarListener.setOnClickListener(new  View.OnClickListener()
{
   @Override
   public void onClick(View view) {
      dbPrediccion.removeEventListener(eventListener);
   }
});
```

Con esto en cuenta, y considerando que no siempre es necesario el mecanismo de sincronización en tiempo real, por ejemplo para datos que sabemos que no van a cambiar o que no lo van a hacer frecuentemente, la API de Firebase nos ofrece una forma alternativa de asociar el listener a una referencia de la base de datos de forma que éste sólo se ejecutará una vez, es decir, recibiremos el valor inicial del nodo en el momento de la suscripción a su referencia y ya no volveremos a recibir más actualizaciones de ese nodo. Esto nos evitará, en el caso de datos que no cambian, el suscribirnos a una referencia y tener que eliminar el listener inmediatamente después de recuperar su valor inicial. Para conseguir esto, el procedimiento sería análogo al ya mencionado con la diferencia de que utilizaríamos el método addListenerForSingleValueEvent(), en vez del ya mencionado addValueEventListener().

```
    eventListener = new ValueEventListener() {
      //...
    };

    dbPrediccion.addListenerForSingleValueEvent(eventListener);
```

Y con esto finalizaríamos este segundo artículo sobre la base de datos en tiempo real de Firebase. En la siguiente entrega seguiremos viendo nuevas formas de leer y mostrar datos desde la base de datos que pueden sernos de utilidad en muchas ocasiones.

3.2.2 Firebase para Android: Base de Datos en Tiempo Real (III)

En el último artículo sobre la base de datos en tiempo real de Firebase ya aprendimos a suscribirnos a un nodo de nuestra base de datos para conocer su valor y ser notificado de sus cambios. Sin embargo nos dejamos en el tintero algo muy importante. ¿Qué ocurre cuando el contenido de un nodo de la base de datos no almacena datos independientes (como el ejemplo que utilizamos de cielo, temperatura y humedad) sino una lista de elementos (normalmente del mismo tipo)?

Con la técnica que ya aprendimos podríamos suscribirnos al nodo padre de la lista de elementos, pero cada vez que se añadiera, modificara o eliminara un elemento de la lista recibiríamos en el evento onDataChange() la lista completa de datos, lo que nos dificultaría saber el cambio concreto que se ha producido, además de incrementar enormemente el tráfico de información necesaria entre Firebase y nuestra aplicación.

Para solucionar esto, Firebase nos ofrece un método alternativo de suscribirnos a un nodo de la base de datos cuando

queremos tratar el contenido de dicho nodo como una lista de elementos. Para ello utilizaremos un listener de tipo ChildEventListener (en vez del ValueEventListener que ya conocíamos). Con un listener de este tipo seremos notificados de 5 posibles eventos:

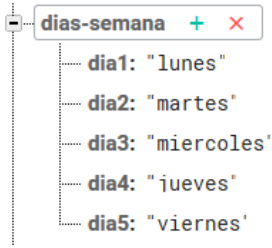
- onChildAdded(). Se lanzará cada vez que se añada un nuevo elemento a la lista, y recibe como parámetro únicamente la información del nuevo elemento añadido. Adicionalmente, también se lanzará en el momento de suscribirnos a la lista, una vez por cada elemento que contenga la lista, de forma que podamos conocer el estado inicial de nuestra lista.
- onChildChanged(). Se lanzará cada vez que un elemento de la lista (incluidos sus subelementos) sea modificado. Recibe como parámetro únicamente la información del elemento que ha sido modificado.
- onChildRemoved(). Se lanzará cada vez que se elimine un elemento de la lista. Recibe como parámetro la información del elemento eliminado.
- onChildMoved(). Se lanzará cuando un elemento de una lista ordenada cambie de posición. No nos preocuparemos por ahora de este evento, volveremos a él más adelante.
- onCancelled(). Se lanzará cuando se produzca cualquier error en la lectura de los datos.

Con estos eventos nos aseguramos de minimizar la información recibida desde Firebase, que se limitará a únicamente los nodos de la lista que cambien, y no la lista completa cada vez que se produce alguna actualización. Aclarar además, que la información recibida en estos eventos nos llegará como siempre en forma de objeto DataSnapshot, que ya aprendimos a gestionar en el artículo pasado.

3.2.2.1 Problema:

Veamos un ejemplo sencillo. Usaré como base los proyectos de Firebase y Android que ya creamos en el primer artículo sobre Firebase. Vamos a crear desde la consola de Firebase una lista muy sencilla de elementos:

Figura 121. Parte 1: Problema 3.2.2.1 Realtime Database



Fuente: Propia

Para acceder a esta lista desde nuestra aplicación Android el procedimiento sería muy similar al ya visto en el artículo anterior, con la única salvedad de que utilizaremos un listener de tipo ChildEventListener. Crearemos primero la referencia a la base de datos y posteriormente asignaremos el listener a la referencia implementando la lógica necesaria dentro de cada evento:

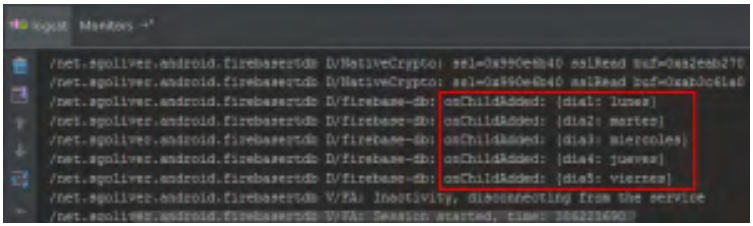
```
DatabaseReference dbDiasSemana =
   FirebaseDatabase.getInstance().getReference()
      .child("dias-semana");


ChildEventListener childEventListener = new ChildEventListener() {
   @Override
   public void onChildAdded(DataSnapshot dataSnapshot, String pre-
viousChildName) {
      Log.d(TAGLOG, "onChildAdded: {" + dataSnapshot.getKey() + ": "
```

```
+ dataSnapshot.getValue() + "}");
    }

    @Override
    public void onChildChanged(DataSnapshot dataSnapshot, String
previousChildName) {
        Log.d(TAGLOG, "onChildChanged: {" + dataSnapshot.getKey() + ":
" + dataSnapshot.getValue() + "}");
    }

    @Override
    public void onChildRemoved(DataSnapshot dataSnapshot) {
        Log.d(TAGLOG, "onChildRemoved: {" + dataSnapshot.getKey() + ":
" + dataSnapshot.getValue() + "}");
    }

    @Override
    public void onChildMoved(DataSnapshot dataSnapshot, String pre-
viousChildName) {
        Log.d(TAGLOG, "onChildMoved: " + dataSnapshot.getKey());
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
        Log.e(TAGLOG, "Error!", databaseError.toException());
    }
};

dbDiasSemana.addChildEventListener(childEventListener);
```

Para este primer ejemplo utilizaré simplemente mensajes de log para mostrar los resultados.

Si ejecutamos la aplicación y revisamos el log desde Android Studio podremos comprobar cómo tras suscribirnos por primera vez a la lista recibiremos un evento onChildAdded() por cada elemento que contiene inicialmente la lista:

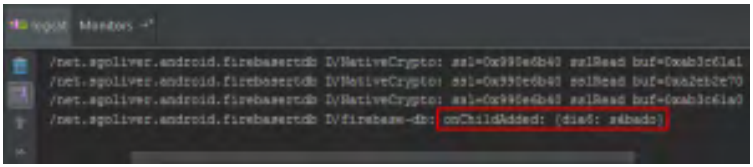
Figura 122. Parte 2: Problema 3.2.2.1 Realtime Database



Fuente: Propia

Si añadimos ahora un nuevo elemento desde la consola de Firebase, por ejemplo {dia6: «Sábado»}, veremos cómo inmediatamente recibimos un nuevo evento onChildAdded() en nuestra aplicación:

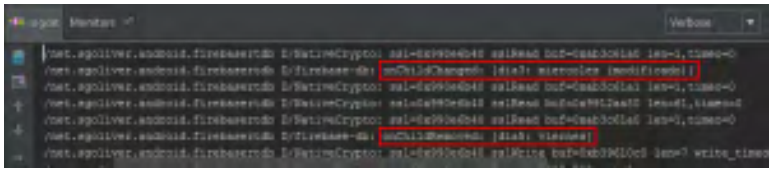
Figura 123. Parte 3: Problema 3.2.2.1 Realtime Database



Fuente: Propia

Igualmente, al modificar o eliminar cualquier dato de la lista veremos los cambios reflejados en el log:

Figura 124. Parte 4: Problema 3.2.2.1 Realtime Database



Fuente: Propia

Vemos por tanto lo sencillo que resulta gestionar desde nuestras aplicaciones Android listas de elementos almacenadas en la base de datos de Firebase. Con la información recibida en estos eventos podríamos realizar cualquier tarea relacionada con los elementos de la lista. Lo más habitual será por supuesto mostrar esta información al usuario, normalmente en controles de tipo ListView o RecyclerView. Para ello tendríamos que mantener actualizado el contenido de la lista en alguna estructura de datos interna (tipo array, lista, …) a partir de la información recibida en los eventos del ChildEventListener, construir los adaptadores necesarios, y asignarlos al ListView/RecyclerView encargado de mostrar los datos al usuario. Como parte del curso tenéis disponibles varios artículos sobre cómo utilizar los controles ListView y RecyclerView. Aunque no sería excesivamente complicado hacerlo de esta forma, afortunadamente Firebase nos proporciona una librería auxiliar, llamada FirebaseUI, con la que (entre otras cosas) podremos ahorrarnos gran parte del código necesario para hacer esto.

FirebaseUI nos permite asociar fácilmente a un control ListView o RecyclerView una referencia a una lista de elementos de una base de datos Firebase. De esta forma, el control se actualizará automáticamente cada vez que se produzca cualquier cambio en la lista, sin tener que gestionar manualmente por nuestra parte los eventos de la lista, ni tener que almacenar en una estructura

paralela la información, ni tener que construir gran parte de los adaptadores necesarios… en resumen, ahorrando muchas líneas de código y evitando muchos posibles errores.

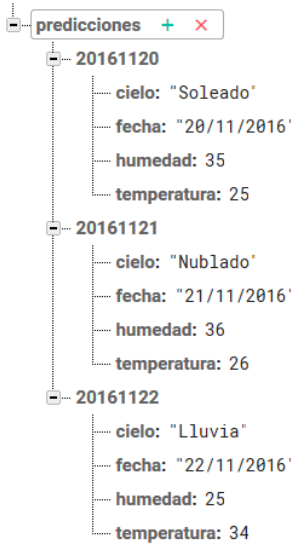
Para utilizar FirebaseUI lo primero que tendremos que hacer será añadir la referencia a la librería en el fichero *build.gradle* de nuestro módulo principal. Hay que tener en cuenta que cada versión de FirebaseUI es compatible únicamente con una versión concreta de Firebase, por lo que debemos asegurar que las versiones utilizadas de ambas librerías son coherentes. En la página de FirebaseUI tenéis disponible la tabla de compatibilidades entre versiones. En mi caso particular utilizaré la versión 1.0.0 de FirebaseUI, por lo que utilizaré la versión 9.8.0 de las librerías de Firebase:

```
dependencies {

   //...

   //Librería de Firebase (para Base de Datos)
   compile 'com.google.firebase:firebase-database:9.8.0'

   //Librería de FirebaseUI (para Base de Datos)
   compile 'com.firebaseui:firebase-ui-database:1.0.0'
}
```

3.2.2.2 Problema:

Hecho esto, ya podríamos empezar a utilizar las distintas funcionalidades ofrecidas por la librería. Pero para no utilizar un ejemplo tan sencillo como el de los días de la semana, vamos a continuar con nuestra fantástica aplicación de predicciones meteorológicas que ya comenzamos en el artículo anterior. Vamos a reestructurar nuestros datos desde la consola de Firebase para almacenar la información meteorológica de varios días, en vez de solo el día actual como teníamos antes. Para ello incluiremos la fecha como parte de cada predicción:

Figura 125. Parte 1: Problema 3.2.2.2 Realtime Database



Fuente: Propia

Algo importante y que antes no comentamos es que cada elemento de la lista debe tener una clave única. En la lista anterior utilizamos claves del tipo «dia1«, «dia2«, … y en este nuevo

ejemplo estoy utilizando como clave de cada elemento la fecha del día codificada en formato «yyyymmdd«. Por ahora no nos preocuparemos mucho de esto ya que nos estamos limitando a *leer* la información de la base de datos, pero cuando veamos cómo *escribir* tendremos que tener en cuenta este tema. Pero no adelantemos acontecimientos, por ahora nos conformamos con saber que debe tratarse de una clave única.

Ya tenemos los datos, ahora debemos decidir cómo queremos mostrarlos al usuario. Para el ejemplo no voy a complicar mucho este tema. Mostraremos los datos en una lista donde cada elemento estará formado por dos líneas de texto: la superior mostrará la fecha y la inferior los datos de la predicción (cielo, temperatura, humedad) uno tras otro. Empezaremos creando el layout XML para mostrar cada elemento de la lista de la forma indicada:

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="wrap_content">

   <TextView android:id="@+id/lblFecha"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textAppearance="@style/TextAppearance.AppCompat.
Medium"/>

   <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="horizontal" >
```

```
    <TextView android:id="@+id/lblCielo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="5dp" />

    <TextView android:id="@+id/lblTemperatura"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="5dp" />

    <TextView android:id="@+id/lblHumedad"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

  </LinearLayout>

</LinearLayout>
```

Lo siguiente será decidir qué tipo de control vamos a utilizar para mostrar los datos. FirebaseUI ofrece soporte tanto para el control ListView como para el más actual RecyclerView. En mi caso de ejemplo utilizaré RecyclerView, aunque en la web de FirebaseUI podéis encontrar el procedimiento para utilizar un ListView, que será muy similar al que veremos ahora.

Para utilizar un RecyclerView vamos a empezar creando un ViewHolder personalizado que gestione los datos de nuestras predicciones. Si no has oido hablar antes del término *ViewHolder* o quieres repasar cómo funciona en general un control RecyclerView te recomiendo consultar, antes de seguir con este tema, el capítulo del curso dedicado al control RecyclerView.

```
public class PrediccionHolder extends RecyclerView.ViewHolder {
   private View mView;

   public PrediccionHolder(View itemView) {
      super(itemView);
      mView = itemView;
   }

   public void setFecha(String fecha) {
      TextView field = (TextView) mView.findViewById(R.id.lblFecha);
      field.setText(fecha);
   }

   public void setCielo(String cielo) {
      TextView field = (TextView) mView.findViewById(R.id.lblCielo);
      field.setText(cielo);
   }

   public void setTemperatura(String temp) {
       TextView field = (TextView) mView.findViewById(R.id.lblTempera-
tura);
      field.setText(temp);
   }

   public void setHumedad(String hum) {
     TextView field = (TextView) mView.findViewById(R.id.lblHumedad);
      field.setText(hum);
   }
}
```

En este ViewHolder, además de su constructor, implementaremos un método *set* para cada uno de nuestros datos de la predicción. Cada uno de estos métodos recuperarán una referencia

a su control correspondiente del layout que hemos creado antes para los elementos de la lista y asignarán su contenido a partir del dato recibido como parámetro.

El siguiente paso será crear un adaptador para nuestro RecyclerView. FirebaseUI nos facilita este paso proporcionándonos una clase genérica, llamada FirebaseRecyclerAdapter, que podemos utilizar con nuestro ViewHolder personalizado que acabamos de crear y la clase java que utilicemos para encapsular la información de cada elemento de la lista. Recordemos que en el artículo anterior ya creamos esta clase, llamada Prediccion, con los tres datos necesarios. Para este ejemplo la ampliaremos con el nuevo dato de *fecha*.

```
public class Prediccion {
   private String cielo;
   private long temperatura;
   private double humedad;
   private String fecha;

   public Prediccion() {
      //Es obligatorio incluir constructor por defecto
   }

    public Prediccion(String fecha, String cielo, long temperatura, double
humedad)
   {
      this.fecha = fecha;
      this.cielo = cielo;
      this.temperatura = temperatura;
      this.humedad = humedad;
   }

   public String getFecha() {
```

```
    return fecha;
}

public void setFecha(String fecha) {
    this.fecha = fecha;
}

public String getCielo() {
    return cielo;
}

public void setCielo(String cielo) {
    this.cielo = cielo;
}

public long getTemperatura() {
    return temperatura;
}

public void setTemperatura(long temperatura) {
    this.temperatura = temperatura;
}

public double getHumedad() {
    return humedad;
}

public void setHumedad(double humedad) {
    this.humedad = humedad;
}

@Override
public String toString() {
    return "Prediccion{" +
            "fecha='" + fecha + '\'' +
```

```
            ", cielo='" + cielo + '\'' +
            ", temperatura=" + temperatura +
            ", humedad=" + humedad +
            '}';
    }
}
```

Con esto ya podemos crear un objeto de tipo FirebaseRecyclerAdapter, personalizada para nuestras clases Prediccion y PrediccionHolder, que recibirá como parámetros el objeto clase de nuestra Predicción (Prediccion.class), el ID del layout de los elementos de la lista (en mi caso lo llamé item_lista.xml), el objeto clase de nuestro ViewHolder (PrediccionHolder.class) y la referencia al nodo de la base de datos que contiene la lista de elementos que queremos mostrar en el control. Adicionalmente, sobrescribiremos el método populateViewHolder() para asignar cada uno de los datos de la predicción con su método correspondiente del ViewHolder. Por último asignaremos el adaptador al control RecyclerView.

Veamos el código completo, que es mucho más sencillo de escribir que de explicar:

```
DatabaseReference dbPredicciones =
    FirebaseDatabase.getInstance().getReference()
        .child("predicciones");

RecyclerView recycler = (RecyclerView) findViewById(R.id.lstPredicciones);
recycler.setHasFixedSize(true);
recycler.setLayoutManager(new LinearLayoutManager(this));

FirebaseRecyclerAdapter mAdapter =
```

```
  new FirebaseRecyclerAdapter<Prediccion, PrediccionHolder>(
      Prediccion.class, R.layout.item_lista, PrediccionHolder.class,
dbPredicciones) {

  @Override
      public void populateViewHolder(PrediccionHolder pred-
ViewHolder, Prediccion pred, int position) {
    predViewHolder.setFecha(pred.getFecha());
    predViewHolder.setCielo(pred.getCielo());
        predViewHolder.setTemperatura(pred.getTemperatura() +
"ºC");
    predViewHolder.setHumedad(pred.getHumedad() + "%");
  }
};

recycler.setAdapter(mAdapter);
```

Un último detalle a tener en cuenta es que, antes de finalizar nuestra actividad debemos indicar a Firebase que ya no queremos seguir recibiendo los eventos asociados a nuestra lista. Para ello, utilizaremos el método cleanup() sobre el adaptador creado, por ejemplo dentro del evento onDestroy() de nuestra actividad principal.

```
@Override
protected void onDestroy() {
  super.onDestroy();
  mAdapter.cleanup();
}
```

Y listo, ya habríamos finalizado los pasos necesarios para mostrar y mantener actualizados los datos de la lista de predicciones en nuestra aplicación Android. Si ejecutamos la aplicación debemos ver inmediatamente los elementos actuales de la lista, y si realizamos cualquier cambio en los datos desde la consola de Firebase éstos deberían reflejarse directamente en la aplicación.

Figura 126. Parte 2: Problema 3.2.2.2 Realtime Database

Fuente: Propia

Con esto finalizaríamos este tercer artículo de la serie sobre la base de datos de Firebase, el segundo dedicado a la lectura de datos desde una aplicación Android. En breve seguiremos con más temas interesantes sobre Firebase.

3.2.3 Firebase para Android: Base de Datos en Tiempo Real (IV)

En los dos artículos anteriores (parte 2 y parte 3) de la serie, nos hemos ocupado de repasar las diferentes formas que tenemos disponibles para leer y mostrar los datos de nuestra base de datos de Firebase, ya sean datos concretos o listas de elementos. Sin embargo aún nos quedan un par de temas importantes que tratar sobre el acceso a los datos: el *filtrado* y la *ordenación* de la información. Ambos mecanismos deben ser tareas habituales

para aquellas personas acostumbradas a trabajar con bases de datos SQL tradicionales, y en este artículo veremos cómo aplicarlos sobre los datos almacenados en una base de datos de Firebase.

Lo primero que debemos tener en cuenta es que en Firebase no vamos a tener todas las facilidades de ordenación y filtrado que suelen encontrarse en bases de datos SQL tradicionales. Por ejemplo, sólo podremos ordenar por un solo criterio, siempre ascendente, y que podrá basarse en la *clave* o el *valor* de los elementos de la lista, y adicionalmente el criterio de filtrado (si se utiliza) será dependiente del criterio de ordenación elegido. Así, si por ejemplo ordenamos una lista por el *valor* de sus elementos, si queremos utilizar algún criterio de filtrado éste se tendrá que basar también, necesariamente, en el *valor* de los elementos (no podría por ejemplo filtrar por la *clave*). Lo iremos entendiendo mejor a lo largo del artículo.

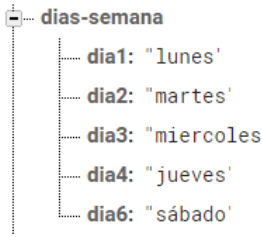
Empecemos por los diferentes métodos de ordenación disponibles. Firebase ofrece tres criterios de ordenación diferentes:

- orderByKey(). Ordena la información por la *clave* de cada elemento de la lista, en orden ascendente.
- orderByValue(). Ordena la información por el *valor* de cada elemento de la lista, en orden ascendente.
- orderByChild(). Ordena la información por el *valor* de una *clave hija* concreta de cada elemento de la lista, en orden ascendente. La ordenación por clave (orderByKey) no necesita más explicación. En cuanto a las dos restantes, la ordenación directa por valor (orderByValue) tendrá más sentido cuando los elementos de la lista tengan un valor simple (numérico, alfanumérico o booleano), y la ordenación por el valor de una clave hija (orderByChild) nos será más útil cuando los elementos de la lista

sean *objetos*, es decir, cuando contengan subelementos, y queramos ordenar por el valor de alguno de estos subelementos.

Veamos un par de ejemplos. Como ejemplo de lista con elementos simples recuperemos nuestra lista de días de la semana:

Figura 127. Nodo dias-semana RealTime DataBase



Fuente: Propia

Si recuperamos esta lista ordenada por clave (orderByKey) obtendremos los elementos exactamente en el mismo orden que vemos en la imagen anterior, ya que en mi caso utilicé claves correlativas y ascendentes (aunque por supuesto esto no tiene por qué ser así necesariamente).

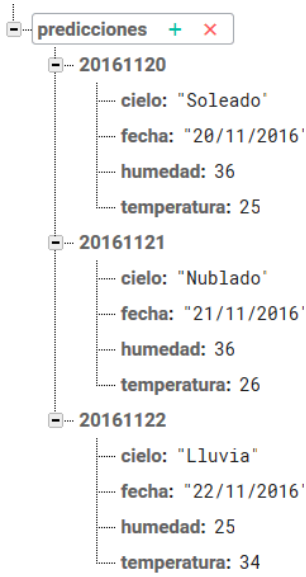
Si ordenamos la lista por valores (orderByValue) obtendríamos los elementos en este orden (por orden alfabético del campo *valor*):

{dia4: «jueves»}
{dia1: «lunes»}
{dia2: «martes»}
{dia3: «miercoles»}
{dia6: «sábado»}

Para este ejemplo no tendría mucho sentido utilizar una ordenación por clave hija ya que los elementos de la lista no con-

tienen subelementos. Pero podemos recuperar el ejemplo de las predicciones meteorológicas:

Figura 128. Nodo Predicciones RealTime DataBase



Fuente: Propia

En este caso, podríamos ordenar la lista por el valor de cualquiera de las claves de los nodos hijo de cada elemento. Por ejemplo, si ordenamos por el campo «cielo» de cada elemento obtendríamos la lista en este orden:

{20161122: {cielo: «Lluvia», …}}
{20161121: {cielo: «Nublado», …}}
{20161120: {cielo: «Soleado», …}}

Habiendo entendido los diferentes métodos de ordenación ya solo nos queda saber cómo aplicarlos al recuperar los datos

de Firebase. Para ello, basta con utilizar el método correspondiente al crear la referencia a la base de datos, teniendo en cuenta además que el resultado ya no será un objeto de tipo DatabaseReference, sino de tipo Query (que realmente es una superclase de DatabaseReference).

Así, por ejemplo, para ordenar la lista de días de la semana por *clave* haríamos lo siguiente:

```
Query diasSemanaPorClave =
  FirebaseDatabase.getInstance().getReference()
    .child("dias-semana")
    .orderByKey();
```

Para obtenerla ordenada por *valor* se haría de forma análoga:

```
Query diasSemanaPorValor =
  FirebaseDatabase.getInstance().getReference()
    .child("dias-semana")
    .orderByValue();
```

Por último, para ordenar la lista de predicciones meteorológicas por la *clave hija* «cielo» de cada elemento, haríamos lo siguiente:

```
Query prediccionesPorClaveHija =
  FirebaseDatabase.getInstance().getReference()
    .child("predicciones")
    .orderByChild("cielo");
```

Como podéis comprobar es tan sencillo como pasar el nombre de la clave hija por la que queremos ordenar al método orderByChild().

Tras obtener una referencia de esta forma, el resto del trabajo con la base de datos se realiza exactamente igual a como lo hemos descrito en artículos anteriores, usando nuestro objeto Query en lugar de una referencia «normal» de tipo DatabaseReference.

Vamos ya con los métodos de filtrado o de consulta (query) en Firebase. Como hemos dicho anteriormente, el método de filtrado que apliquemos debe basarse necesariamente en el mismo campo por el que hayamos realizado la ordenación de la lista. Así, si ordenamos por clave, podremos filtrar por dicha clave. Si ordenamos por valor podremos filtrar por valor, y de forma análoga en el caso de ordenar por una clave hija.

Los distintos métodos de filtrado/consulta que tendremos disponibles serán los siguientes:

- limitToFirst(*N*). La consulta sólo devolverá los primeros *N* elementos de la lista ordenada.
- limitToLast(*N*). La consulta sólo devolverá los últimos *N* elementos de la lista ordenada.
- startAt(…). La consulta sólo devolverá los elementos cuya clave/valor/valor_clave_hija (según el método de ordenación elegido) sea igual o superior al dato pasado como parámetro.
- endAt(…). La consulta sólo devolverá los elementos cuya clave/valor/valor_clave_hija (según el método de ordenación elegido) sea igual o inferior al dato pasado como parámetro.
- equalTo(…). La consulta sólo devolverá los elementos

cuya clave/valor/valor_clave_hija (según el método de ordenación elegido) sea igual al dato pasado como parámetro. Al contrario de lo que ocurre con los métodos de ordenación, en este caso sí podremos utilizar varios criterios de filtrado en la misma consulta, es decir, podremos combinar varios de los métodos anteriores para obtener sólo el rango de elementos necesario. Así, si ordenamos por ejemplo por valor, podríamos recuperar los primeros 50 valores de la lista (limitToFirst(50)) que sean mayores a un determinado valor «X» (startAt(«X»)). Este tipo de filtros podrían servirnos por ejemplo para paginar una lista grande de elementos. La forma de aplicar los criterios de filtrado es análoga a la de los métodos de ordenación, utilizando el/los métodos necesarios al crear la referencia a la base de datos. Así, por ejemplo, si volvemos a utilizar el ejemplo de los días de la semana, podríamos obtener los 2 días siguientes al «miércoles» (inclusive) de la siguiente forma:

```
Query diasSemanaPorValorFiltrado =
  FirebaseDatabase.getInstance().getReference()
    .child("dias-semana")
    .orderByValue()
    .startAt("miercoles")
    .limitToFirst(2);
```

Con este criterio obtendríamos la siguiente lista:

{dia3: «miercoles»}

{dia6: «sábado»}

Y en principio nada más sobre los criterios de ordenación y filtrado de Firebase. Ya solo nos quedaría volver sobre un detalle que dejamos pendiente en el artículo anterior. Cuando comen-

tamos los eventos que podíamos gestionar al suscribirnos a una lista de elementos, dejamos apartado temporalmente el evento llamado onChildMoved(). Entonces no tenía mucho sentido que un elemento de una lista pudiera moverse de sitio, y por eso dejamos su explicación para más adelante. Ahora, una vez introducidos los criterios de ordenación y filtrado de Firebase, este evento cobra sentido de forma inmediata. El evento onChildMoved() se lanzará cada vez que un elemento de una lista ordenada cambie de posición debido a una modificación de su *valor* (lo que incluye a sus subelementos) y por tanto irá acompañado habitualmente del evento onChildChanged(). Todo esto nos permitirá ser notificados de este tipo de cambios en nuestra lista para actuar en consecuencia si fuera necesario, por ejemplo cambiando la posición del elemento en alguna lista que se esté mostrando al usuario en la interfaz de nuestra aplicación (si se usa FirebaseUI, como vimos en el artículo anterior, no será necesario preocuparnos de estos temas).

Con esto terminaríamos por el momento con las alternativas que ofrece Firebase para *leer y consultar* información de su base de datos. En el artículo siguiente comenzaremos ya a hablar de las distintas formas que tenemos disponibles para *escribir y actualizar* los datos desde nuestra aplicación Android.

3.2.4 Firebase para Android: Base de Datos en Tiempo Real (V)

En los artículos anteriores de esta serie nos hemos centrado principalmente en los distintos mecanismos que nos ofrece Firebase para leer o consultar la información almacenada en la base de datos desde nuestras aplicaciones Android. Aprendimos a leer datos individuales y listas de elementos, vimos cómo suscribirnos a una determina ruta de la base de datos para ser notificados

cuando haya cambios en la información, aprendimos a utilizar la librería FirebaseUI para vincular controles Android de tipo lista (ListView/RecyclerView) a la base de datos, y descubrimos cómo filtrar y ordenar la información recuperada.

Sin embargo no solo de *leer* vive el hombre, y habitualmente también nos será necesario *escribir* o actualizar información en la base de datos para ponerla a buen recaudo y/o hacerla disponible a otros clientes implicados (recordemos que con Firebase, cada vez que escribamos algo en la base de datos, dicha información se sincronizará automáticamente y al instante en todos los clientes suscritos). Éste será el tema central de este nuevo artículo de la serie.

3.2.4.1 Problema:

Empezaremos por la operación más básica, la de escribir un dato sencillo *(clave + valor* simple) a nuestra base de datos. Para hacer esto, simplemente tendremos que construir una referencia, como siempre de tipo DatabaseReference, a la *clave* que queremos escribir (ya existente o no) y utilizar sobre ella el método setValue(*valor*) para establecer su valor. Así, siguiendo con el ejemplo de los días de la semana utilizado en artículos anteriores, si quisiéramos añadir un nuevo elemento a nuestra lista de días de la semana podríamos hacer lo siguiente:

```
DatabaseReference dbRef =
   FirebaseDatabase.getInstance().getReference()
      .child("dias-semana");

dbRef.child("dia7").setValue("domingo");
```

Como podéis comprobar, la referencia sobre la que vamos a escribir podemos obtenerla directamente al crear el objeto DatabaseReference, o bien «navegando» con el método child() sobre una referencia obtenida previamente.

Hay que tener muy en cuenta que con setValue() podemos tanto insertar nuevos elementos como actualizar los ya existentes. Así, si la clave sobre la que escribimos no existe estaremos insertando un nuevo nodo, y si ya existía estaremos sobrescribiendo su valor (y ojo, esto incluye la eliminación de todos sus elementos descendientes si los tenía).

Los posibles valores que acepta el método setValue() son los siguientes:

- String
- Long
- Double
- Boolean
- Map<String, Object>
- List<Object>
- Objeto Java personalizado

Los cuatro primeros no necesitan mayor explicación, son tipos sencillos, por lo que nos detendremos tan solo en los tres siguientes.

El método setValue() puede recibir como parámetro un objeto de tipo Map que contenga una serie de pares clave-valor que se insertarán como *hijos* de la clave sobre la que se está escribiendo. Esto nos puede facilitar la vida a la hora de escribir estructuras de árbol complejas, ya que nos evita tener que escribir nodo a nodo. Así, imaginemos por ejemplo que quisiéramos insertar en nuestra lista de días de la semana un nodo de este tipo:

«dia7»:

«periodo-1»: «domingo-mañana»

«periodo-2»: «domingo-tarde»

«periodo-3»: «domingo-noche»

Podríamos crear los tres nodos hijo previamente en forma de objeto Map e insertarlos de una sola vez utilizando el método setValue().

```
DatabaseReference dbRef =
   FirebaseDatabase.getInstance().getReference()
      .child("dias-semana");

Map<String, String> domingo = new HashMap<>();
domingo.put("periodo-1", "domingo-mañana");
domingo.put("periodo-2", "domingo-tarde");
domingo.put("periodo-3", "domingo-noche");

dbRef.child("dia7").setValue(domingo);
```

Si consultamos el nuevo nodo creado en la consola de Firebase veremos que efectivamente el resultado ha sido el esperado:

Figura 129. Insertar datos en RealTime Database con HashMap

Fuente: Propia

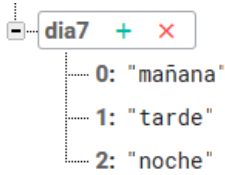
El siguiente tipo admitido es una lista de objetos de tipo List. Si pasamos a setValue() una lista de objetos de este tipo, Firebase creará por nosotros una lista de elementos donde cada uno de

ellos tendrá como *clave* su posición en la lista y como *valor* el dato de la lista situado en dicha posición. Veamos un ejemplo:

```
DatabaseReference dbRef =
   FirebaseDatabase.getInstance().getReference()
      .child("dias-semana");

List<String> domingo = new LinkedList<>();
domingo.add("mañana");
domingo.add("tarde");
domingo.add("noche");

dbRef.child("dia7").setValue(domingo);
```

Consultando esta información desde la consola de Firebase veremos nuestra nueva lista en el formato indicado:

Figura 130. Insertar datos en RealTime Database con LinkedList



Fuente: Propia

La última opción disponible es utilizar objetos Java personalizados. En un artículo pasado vimos como Firebase nos permite recuperar información de la base de datos y mapearla directamente sobre un objeto Java propio que tenga la misma estructura. Pues bien, a la hora de escribir información sobre la base de datos vamos a disponer de las mismas facilidades. El único requisito será como siempre que nuestra clase Java personalizada tenga un constructor por defecto y disponga de *getters* y *setters* públicos para los datos que queremos escribir a la base de datos. Siguiendo

con el ejemplo de las predicciones meteorológicas, recordemos que ya construimos la siguiente clase para encapsular nuestros datos:

```
public class Prediccion {
   private String cielo;
   private long temperatura;
   private double humedad;
   private String fecha;

   public Prediccion() {
      //Es obligatorio incluir constructor por defecto
   }

   public Prediccion(String fecha, String cielo, long temperatura, double
humedad)
   {
      this.fecha = fecha;
      this.cielo = cielo;
      this.temperatura = temperatura;
      this.humedad = humedad;
   }

   public String getFecha() {
      return fecha;
   }

   public void setFecha(String fecha) {
      this.fecha = fecha;
   }

   public String getCielo() {
      return cielo;
   }
```

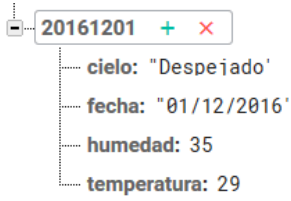
```
    public void setCielo(String cielo) {
        this.cielo = cielo;
    }

    public long getTemperatura() {
        return temperatura;
    }

    public void setTemperatura(long temperatura) {
        this.temperatura = temperatura;
    }

    public double getHumedad() {
        return humedad;
    }

    public void setHumedad(double humedad) {
        this.humedad = humedad;
    }

    @Override
    public String toString() {
        return "Prediccion{" +
                "fecha='" + fecha + '\'' +
                ", cielo='" + cielo + '\'' +
                ", temperatura=" + temperatura +
                ", humedad=" + humedad +
                '}';
    }
}
```

Firebase nos va a permitir construir un objeto de este tipo y pasarlo directamente al método setValue() para utilizarlo como valor de la clave a escribir.

```
DatabaseReference dbRef =
   FirebaseDatabase.getInstance().getReference()
      .child("predicciones");

Prediccion pred =
   new Prediccion("01/12/2016", "Despejado", 29, 35);

dbRef.child("20161201").setValue(pred);
```

Podemos comprobar en la consola de Firebase que el nodo insertado es el que esperábamos.

Figura 131. Insertar datos en RealTime Database con Objetos y la Class



Fuente: Propia

El método setValue() es la forma más sencilla de insertar o actualizar información de la base de datos, pero nos permite actualizar sólo una clave por operación. Firebase nos ofrece un método alternativo que nos permite actualizar varias claves al mismo tiempo, y además de forma *atómica*, es decir, que nos garantizará que se realizan correctamente todas las actualizaciones o bien no se realizará ninguna. Este método se llama updateChildren() y recibe como parámetro un objeto de tipo Map con una serie de pares clave-valor donde cada clave es la ruta a una de las referencias que queremos actualizar y el valor será el *valor* (valga la redundancia) que queremos asignar a dicha referencia.

Así, si por ejemplo queremos actualizar de forma simultánea la temperatura y humedad de una predicción de nuestra lista, podríamos hacer lo siguiente:

```
DatabaseReference dbRef =
   FirebaseDatabase.getInstance().getReference()
      .child("predicciones");

Map<String, Object> actualizacion = new HashMap<>();
actualizacion.put("/temperatura", 29);
actualizacion.put("/humedad", 34);

dbRef.child("20161120")
   .updateChildren(actualizacion);
```

Del ejemplo anterior hacer notar que las diferentes rutas indicadas deben comenzar por el caracter «/». También importante conocer que las rutas incluidas no tienen por qué pertenecer al mismo nodo, sino que pueden indicarse claves de diferentes nodos de nuestra base de datos. Por ejemplo podríamos actualizar de forma simultánea la temperatura de un día y la humedad de otro distinto:

```
DatabaseReference dbRef =
   FirebaseDatabase.getInstance().getReference();

Map<String, Object> actualizacion2 = new HashMap<>();
actualizacion2.put("/20161120/temperatura", 25);
actualizacion2.put("/20161121/humedad", 31);

dbRef.child("predicciones")
   .updateChildren(actualizacion2);
```

Tanto el método setValue() como updateChildren() pueden recibir de forma opcional un segundo parámetro con un listener de tipo CompletionListener, cuyo método onComplete() se llamará automáticamente cuando la operación se dé por finalizada (correctamente o con algún error). Este método recibe como primer parámetro un objeto de tipo DatabaseError que contendrá el mensaje de error en caso de que la operación no haya finalizado correctamente, o será null en caso de haberse finalizado sin errores. Veamos un ejemplo:

```
Prediccion pred =
   new Prediccion("20161201", "Despejado", 29, 35);

dbRef.child("predicciones")
   .setValue(pred, new DatabaseReference.CompletionListener(){
       public void onComplete(DatabaseError error, DatabaseReference
ref) {
          if(error == null)
             Log.i(LOGTAG, "Operación OK");
          else
             Log.e(LOGTAG, "Error: " + error.getMessage());
       }
});
```

Vamos ahora como otro tema importante que habíamos dejado pendiente en artículos anteriores. Hasta ahora, cada vez que hemos creado o insertado nuevos datos en una lista de elementos en nuestra base de datos Firebase hemos utilizado claves arbitrarias elegidas o construidas por nosotros, por ejemplo: «dia1», «dia2», … en nuestro ejemplo de los días de la semana, o «20161122», «20161123», … en el ejemplo de las predicciones. Aunque esto es perfectamente válido, en la práctica nos obliga a mantener cierto control sobre las claves ya utilizadas para poder

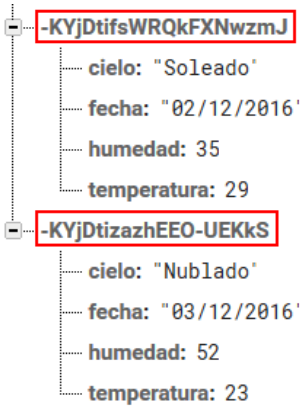
asegurar que no utilizamos claves duplicadas o incorrectas en los nuevos elementos que añadamos a la lista. También nos obliga a implementar la lógica necesaria para generar dichas claves personalizadas.

Firebase nos ofrece un método alternativo que nos facilitará las cosas cuando no necesitemos personalizar a tal nivel las claves de los elementos de nuestras listas. Para ello nos ofrece el método push() que generará por nosotros una *clave* única y nos devolverá una referencia a ella para que podamos establecer su valor. Estas claves automáticas se basan en el *timestamp* actual (fecha-hora) por lo que nos asegurará que los elementos de la lista quedarán ordenados por clave de forma cronológica. Veamos un ejemplo para ver que aspecto tienen las claves generadas, vamos a insertar nuevas predicciones a nuestra lista utilizando el método push().

```
DatabaseReference dbPredicciones =
   FirebaseDatabase.getInstance().getReference()
      .child("predicciones");

Prediccion p1 = new Prediccion("02/12/2016", "Soleado", 29, 35);
Prediccion p2 = new Prediccion("03/12/2016", "Nublado", 23, 52);

dbPredicciones.push().setValue(p1);
dbPredicciones.push().setValue(p2);
```

Si observamos los nuevos datos insertados desde la consola de Firebase veremos lo siguiente:

Figura 132. Insertar datos en RealTime Database con Objetos y Push



Fuente: Propia

Como podéis comprobar, las claves generadas no nos dicen nada a simple vista, pero nos aseguran que si ordenamos la lista por clave (consulta los métodos de ordenación de Firebase) los elementos quedarán ordenados en orden cronológico de forma ascendente.

En ocasiones puede ser útil conocer la clave que ha asignado Firebase al nuevo elemento insertado, por ejemplo para insertar referencias a dicho elemento en otros lugares de la base de datos. Para ello podemos aprovechar que el método push() devuelve una referencia al nuevo elemento creado, por lo que podemos utilizar su método getKey() para conocer la clave que le ha asignado. Por ejemplo:

```
DatabaseReference dbPredicciones =
   FirebaseDatabase.getInstance().getReference()
      .child("predicciones");

Prediccion p3 = new Prediccion("02/12/2016", "Tormenta", 20, 54);
```

```
String claveP3 = dbPredicciones.push(p3).getKey();
```

Y nos queda un último tema pendiente en relación a la actualización de información sobre la base de datos, la eliminación de contenido. Para eliminar un nodo de nuestra base de datos tenemos dos posibilidades:

- Utilizar los métodos setValue() o updateChildren() para asignarle el valor null.
- Utilizar el método removeValue() sobre la referencia que queremos eliminar.

De esta forma, eliminar por ejemplo uno de nuestros días de la semana sería tan sencillo como:

```
DatabaseReference dbRef =
   FirebaseDatabase.getInstance().getReference();

//Alternativa 1
dbRef.child("dias-semana").child("dia6").setValue(null);

//Alternativa 2
dbRef.child("dias-semana").child("dia7").removeValue();
```

3.2.5 Evaluación 4

1. ¿Cuál es la definición de Firebase?
2. Firebase permite compilar mejores apps. ¿Cuál herramienta de Firebase me permite autenticar usuarios de forma simple y segura?
3. Con cuál de las siguientes herramientas no es compatible Firebase Authentication
4. ¿Cuál es la dependencia de la biblioteca de Android de Firebase Authentication?
5. ¿Cuál es el método de Firebase Authentication, para crear una cuenta nueva, pasa la dirección de correo electrónico y la contraseña del usuario nuevo?
6. ¿Cuál es el método cuando un usuario acceda a tu app, pasa la dirección de correo electrónico y la contraseña?
7. ¿Cuál es el fichero de configuración por defecto de Firebase que debemos colocar dentro de la carpeta «/app» de nuestro proyecto?
8. Con respecto a la confiabilidad y rendimiento cuál de las siguientes características corresponde al Realtime Database
9. Con respecto a la escalabilidad cuál de las siguientes características corresponde al Realtime Database
10. Con respecto a la seguridad a cuál corresponde la siguiente característica: Las reglas no se aplican en cascada, a menos que uses un comodín.
11. Con respecto al precio cuál de las siguientes características corresponde al Cloud Firestore

12. ¿Cuál base de datos cuenta con la capacidad de almacenar datos en la nube sin la necesidad de preocuparnos por toda la infraestructura de servidor?

13. Realtime Database de Firebase nos ofrece una base de datos SQL tradicional, con sus tablas y sus registros perfectamente estructurados.

14. Cada nodo de la base de datos de Firebase tendrá un nombre o clave y un valor, o bien solo el nombre en caso de que vaya a servir como nodo padre a otros elementos. ¿A qué tipo de dato corresponde el siguiente ejemplo {«nombre»:»pepe», «edad»:25}?

15. ¿Cuál es el objeto que representa básicamente una rama del árbol JSON del Realtime Database y contendrá toda la información de un nodo determinado, con su clave, su valor, y su listado de nodos hijos?

16. ¿Cuál es el método del Realtime Database de Firebase que nos permite navegar entre los nodos hijo y que recibe como parámetro el nombre del subnodo al que queremos bajar en el árbol?

17. ¿Cuál es el evento en el Realtime Database que se lanza cada vez que un elemento de la lista (incluidos sus subelementos) sea modificado y recibe como parámetro únicamente la información del elemento que ha sido modificado?

18. Firebase ofrece diferentes métodos de ordenación disponibles. ¿Cuál criterio ordena la información por el valor de cada elemento de la lista de un nodo, en orden ascendente?

19. Hay distintos métodos de filtrado/consulta en el Realtime Database. ¿Cuál método consulta y sólo devuelve los últimos N elementos de la lista ordenada?

20. ¿Cuál método en Realtime Database nos permite actualizar sólo una clave por operación y puede recibir como parámetro un objeto de tipo Map que contenga una serie de pares clave-valor que se insertarán como hijos de la clave sobre la que se está escribiendo?

21. ¿Cuál es el método en Realtime Database nos permite actualizar varias claves al mismo tiempo y recibe como parámetro un objeto de tipo Map con una serie de pares clave-valor donde cada clave es la ruta a una de las referencias que queremos actualizar y el valor será el que queremos asignar a dicha referencia?

Capítulo 4
Localización Geográfica en Android

4.1 Localización Geográfica en Android

La **localización geográfica** o **detección de ubicación** en Android es una funcionalidad habitual en muchas de las aplicaciones actuales. Proporciona una parte importante del *contexto* del usuario y puede utilizarse no solo para su aplicación directa, es decir, para mostrar la ubicación del dispositivo, sino también para dar forma a multitud de otras características de una aplicación, por ejemplo para personalizar la información mostrada al usuario, para etiquetar datos, para activar o desencadenar eventos, o incluso puede convertirse en la base principal sobre la que sustentar una aplicación o un juego (¿os suena de algo *Pokemon GO*?).

En los artículos que componen esta serie aprenderemos a utilizar los servicios de localización o ubicación de Android, integrados desde hace ya algún tiempo en los Google Play Services. Conoceremos sus características principales, las peculiaridades del servicio derivadas de los diferentes métodos de ubicación disponibles (GPS, Wi-Fi, Red Móvil), y por supuesto la preparación y configuración de un proyecto de Android Studio que haga uso de esta funcionalidad.

4.1.1 Localización geográfica en Android (I)

En este capítulo vamos a aprender a acceder a los datos de localización o ubicación actual del dispositivo en el que se está ejecutando una aplicación Android. El contar con datos de ubicación va a multiplicar las posibilidades a la hora de crear nuestras aplicaciones, permitiendo ofrecer al usuario información relativa al *contexto* en el que se encuentra.

Desde el punto de vista del desarrollador, hay que decir que la API de ubicación es una de esas características de Android que quizá no sean tan intuitivas como deberían ser, por lo que es importante estar atento a los detalles, aunque bien es cierto que ha ido mejorando con los años y a día de hoy contamos con muchas «facilidades» que antes no teníamos. Como ejemplo más llamativo, actualmente no tenemos que preocuparnos demasiado, al menos no de forma explícita, de qué proveedor de localización queremos utilizar en cada momento (señal móvil, Wi-Fi, o GPS), ya que tenemos disponible con un nuevo proveedor (*Fused Location Provider*) que se encargará automáticamente de gestionar todas las fuentes de datos disponibles para obtener la información que nuestra aplicación necesita.

4.1.1.1 Problema:

La funcionalidad de localización geográfica fue movida hace ya algún tiempo desde el SDK general de Android a las librerías de los *Google Play Services*. Por lo tanto, lo primero que tendremos que aprender será a conectarnos a dichos servicios.

El primer paso siempre que queramos hacer uso de los Google Play Services será añadir a la sección de dependencias de nuestro fichero *build.gradle* la referencia a las librerías necesarias. En el caso de estos servicios existen dos alternativas, o bien añadimos la referencia a la librería completa de Google Play Services, que nos daría acceso a todos los servicios disponibles, o bien utilizamos las librerías independientes de cada servicio que vayamos a utilizar.

Para la primera opción la referencia a añadir sería la siguiente (en el momento de escribir este artículo la versión más reciente de los servicios es la 9.4.0):

```
dependencies {
    ...
    compile 'com.google.android.gms:play-services:9.4.0'
}
```

Y si sólo queremos añadir la librería de localización (que también incluye las funciones de *Reconocimiento de Actividad* y *Google Places*), utilizaríamos la siguiente:

```
dependencies {
    ...
    compile 'com.google.android.gms:play-services-location:9.4.0'
}
```

Por norma general utilizaremos esta segunda alternativa, añadiendo únicamente las librerías de los servicios que vamos a utilizar. Podéis encontrar el listado completo de librerías para los servicios disponibles en el siguiente enlace.

Hecho esto vamos a empezar a crear nuestra aplicación de ejemplo. No nos complicaremos mucho, incluiremos tan sólo un par de etiquetas de texto para mostrar la latitud y longitud recibidas, y un botón para iniciar o parar las actualizaciones de posición (esto último lo veremos en el próximo artículo).

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:orientation="vertical"
    tools:context="net.sgoliver.android.localizacion.MainActivity">
```

```
  <TextView android:id="@+id/lblLatitud"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/latitud" />

  <TextView android:id="@+id/lblLongitud"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/longitud" />

  <ToggleButton android:id="@+id/btnActualizar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="@string/parar_actualizaciones"
    android:textOff="@string/iniciar_actualizaciones" />

</LinearLayout>
```

Obtenemos como siempre las referencias a estos controles en el método onCreate() de la actividad principal, y crearemos un método auxiliar updateUI() que actualice los campos de latitud y longitud a partir de un objeto Location, que como veremos más adelante será lo que obtengamos del servicio de localización.

```
import android.location.Location;

//...

@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);

  lblLatitud = (TextView) findViewById(R.id.lblLatitud);
```

```
    lblLongitud = (TextView) findViewById(R.id.lblLongitud);
    btnActualizar = (ToggleButton) findViewById(R.id.btnActualizar);

    //...
}

private void updateUI(Location loc) {
    if (loc != null) {
        lblLatitud.setText("Latitud: " + String.valueOf(loc.getLatitude()));
          lblLongitud.setText("Longitud: " + String.valueOf(loc.getLongi-
tude()));
    } else {
        lblLatitud.setText("Latitud: (desconocida)");
        lblLongitud.setText("Longitud: (desconocida)");
    }
}
```

El objeto Location simplemente encapsula los datos principales de una dirección geográfica, entre otros la latitud y longitud, a los que podemos acceder mediante los métodos getLatitude() y getLongitude() respectivamente. En caso de recibirse un objeto nulo mostraremos un literal de dirección «*(desconocida)*«, más adelante veremos por qué puede ocurrir esto.

Hechos los preparativos iniciales, empecemos ya a añadir funcionalidad más específica del tema que nos ocupa. Comenzaremos por crear un objeto de tipo GoogleApiClient, que generalmente será el punto de acceso común a los servicios de Google Play. Para la creación de este objeto deberemos indicar la API a la que queremos acceder y cómo queremos gestionar la conexión con los servicios (de forma manual o automática, utilizaremos la segunda opción siempre que sea posible). Habrá que proporcionar también los *listeners* necesarios para responder a los eventos de conexión/desconexión a los servicios y a posibles errores que pudieran producirse durante la conexión. Veamos en primer lu-

gar el código de creación de este objeto y posteriormente pasaremos a comentar los detalles.

```
GoogleApiClient apiClient = new GoogleApiClient.Builder(this)
   .enableAutoManage(this, this)
   .addConnectionCallbacks(this)
   .addApi(Location.API)
   .build();
```

Como puede comprobarse, la configuración del cliente se realiza utilizando un patrón *builder* a través de la clase GoogleApiClient.Builder.

En primer lugar solicitamos con enableAutoManage() que la gestión de la conexión a los servicios se realice automáticamente, esto nos evitará entre otras cosas tener que conectarnos y desconectarnos manualmente a los servicios en los eventos onStart() y onStop() de la actividad, y además también se gestionarán de forma automática muchos de los posibles errores que puedan producirse durante la conexión (por ejemplo, solicitando al usuario una actualización de los *play services* en el dispositivo, si fuera necesario). Este método recibe dos parámetros, el primero de ellos es una referencia a la actividad en cuyo ciclo de vida debe integrarse la conexión/desconexión a los servicios, en nuestro caso será la propia actividad principal (this). El segundo parámetro es la referencia al listener (de tipo OnConnectionFailedListener) que se llamará en caso de producirse algún error que el sistema no pueda gestionar automáticamente. En nuestro caso haremos que la propia actividad principal implemente la interfaz OnConnectionFailedListener, por lo que también pasaremos this en este segundo parámetro. Para implementar esta interfaz tendremos que definir el método onConnectionFailed(ConnectionResult), que en nuestro caso de ejemplo se limitará a mostrar un mensaje de error en el log:

```
@Override
public void onConnectionFailed(ConnectionResult result) {
   //Se ha producido un error que no se puede resolver automáticamente
    //y la conexión con los Google Play Services no se ha establecido.

   Log.e(LOGTAG, "Error grave al conectar con Google Play Services");
}
```

Lo siguiente que hacemos es indicar el listener que se ocupará de responder a los eventos de conexión y desconexión de los servicios, mediante una llamada a addConnectionCallbacks(). Esto no es obligatorio, aunque en la práctica casi siempre nos interesará conocer cuándo se realiza la conexión y cuándo se pierde, de forma que podamos adaptarnos convenientemente a cada situación (es importante entender que no podemos hacer llamadas a los servicios mientras no estemos conectados a ellos). Una vez más haremos que nuestra actividad implemente la interfaz necesaria, en este caso ConnectionCallbacks, lo que nos obligará a implementar los métodos onConnected(), que se ejecutará cuándo se realice la conexión con los servicios, y onConnectionSuspended(), que se lanzará cuando la conexión se pierda temporalmente (cuando la conexión se recupera volverá a lanzarse el evento onConnected()). En breve veremos el código de estos dos eventos.

Posteriormente indicamos mediante addApi() la API de los servicios a los que vamos a acceder, en este caso Location.API. En caso de querer utilizar diferentes servicios podríamos realizar varias llamadas al método addApi() añadiendo los servicios necesarios.

Por último, construimos el objeto final llamando al método build(). Esto iniciará la conexión con los servicios solicitados, lo que desembocará como hemos indicado en una llamada al

evento onConnectionFailed() en caso de error, o al evento onConnected() en el caso de funcionar todo correctamente.

El evento onConnected() se aprovecha frecuentemente para realizar la interacción deseada con el servicio, una vez que estamos seguros que la conexión se ha realizado correctamente. En nuestro caso utilizaremos este evento para obtener una primera posición inicial para inicializar los datos de latitud y longitud de la interfaz. Para ello obtendremos la *última posición geográfica conocida*, lo que conseguimos llamando al método getLastLocation() del proveedor de datos de localización LocationServices.FusedLocationApi, pasándole como parámetro la referencia al cliente API que hemos creado anteriormente. Por último llamamos a nuestro método auxiliar updateUI() con el valor recibido para mostrar los datos en la actividad principal. Muestro también a continuación el evento onConnectionSuspended(), que para nuestro ejemplo se limitará a mostrar un mensaje en el log.

```
@Override
public void onConnected(@Nullable Bundle bundle) {
   //Conectado correctamente a Google Play Services

   //...

   Location lastLocation =
      LocationServices.FusedLocationApi.getLastLocation(apiClient);

   updateUI(lastLocation);

   //...
}

@Override
public void onConnectionSuspended(int i) {
   //Se ha interrumpido la conexión con Google Play Services
```

```
    Log.e(LOGTAG, "Se ha interrumpido la conexión con Google Play
Services");
}
```

¿Pero por qué hablo de «*última posición conocida*» y no de «*posición actual*«? En Android no existe ningún método del tipo "*obtenerPosiciónActual()*". Obtener la posición a través de un dispositivo de localización como por ejemplo el GPS no es una tarea inmediata, sino que puede requerir de un cierto tiempo de espera y procesamiento, por lo que no tendría demasiado sentido proporcionar un método de ese tipo. Lo más parecido que encontramos es el método indicado getLastLocation() que nos devuelve la posición más reciente obtenida por el dispositivo (no necesariamente solicitada por nuestra aplicación). Y es importante entender esto: este método NO devuelve la posición actual, este método NO solicita una nueva posición al proveedor de localización, este método se limita a devolver la última posición que se obtuvo a través del proveedor de localización. Y esta posición se pudo obtener hace unos pocos segundos, hace días, hace meses, o incluso nunca (si el dispositivo ha estado apagado, si nunca se ha activado el GPS, …). Por tanto, cuidado cuando se haga uso de la posición devuelta por el método getLastLocation(). En los casos raros en que no existe ninguna posición conocida en el dispositivo este método devuelve null, algo para lo que ya hemos preparado nuestro método updateUI().

En el código anterior del evento onConnected() he omitido inicialmente, por claridad, un fragmento de código relacionado con la obtención de los permisos necesarios para que la aplicación pueda acceder a datos de localización. A continuación, comentaré un poco este tema de permisos y completaremos el código anterior.

Para empezar, indicar que los permisos relacionados con la ubicación geográfica en Android son básicamente dos:

- ACCESS_FINE_LOCATION. Permiso para acceder a datos de localización con una precisión alta.
- ACCESS_COARSE_LOCATION. Permiso para acceder a datos de localización con una precisión baja.

Dependiendo de qué permiso/s solicita nuestra aplicación, los datos de localización obtenidos posteriormente podrán tener una mayor o menor precisión.

Para versiones de Android hasta la 5.0 los permisos necesarios para la aplicación se deben declarar en el fichero AndroidManifest.xml, y deben ser aceptados por el usuario (todos o ninguno) en el momento de instalar la aplicación. En nuestro caso solicitaremos permisos para obtener ubicaciones con la máxima precisión disponible, añadiendo la cláusula correspondiente <uses-permission> a nuestro fichero *AndroidManifest.xml*

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="net.sgoliver.android.localizacion">

      <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

   ....

</manifest>
```

Sin embargo, a partir de Android 6.0 algunos permisos se deben consultar en tiempo de ejecución, con lo que el usuario decidirá si conceder o no, cada uno de ellos, en el momento en

que se haga uso de cada funcionalidad que los necesite (y no en la instalación como ocurría en versiones anteriores). Nuestra aplicación debería estar preparada por tanto para todas las situaciones posibles, es decir, debería funcionar sin errores tanto si se le conceden los permisos solicitados como si no (deshabilitando por supuesto en este caso la funcionalidad asociada).

No entraré en mucho detalle dado que no es el tema principal de este artículo, para más información sobre permisos puede consultarse la documentación oficial de Android, pero veamos rápidamente cómo chequear y solicitar permisos en tiempo de ejecución para versiones de Android >= 6.0. Para chequear si nuestra aplicación tiene concedido un determinado permiso utilizamos el método *checkSelfPermission()*. En el caso de no tenerlo aún concedido, lo solicitaremos al usuario llamado a *requestPermissions()*, pasándole como parámetro el identificador del permiso deseado y una constante arbitraria definida por nuestra aplicación (PETICION_PERMISO_LOCALIZACION en mi ejemplo) que nos permita identificar posteriormente dicha petición. Para conocer el resultado de la petición sobrescribiremos el evento onRequestPermissionResult(), utilizando la constante indicada para reconocer a qué petición corresponde el resultado recibido.

A continuación completo el código del evento onConnected() con el chequeo/petición de permisos y añado la implementación de onRequestPermissionResult() para actuar según la respuesta del usuario a la petición de permisos en caso de haberse realizado.

```
@Override
public void onConnected(@Nullable Bundle bundle) {
  //Conectado correctamente a Google Play Services

  if (ActivityCompat.checkSelfPermission(this,
```

```
        Manifest.permission.ACCESS_FINE_LOCATION) != Package-
Manager.PERMISSION_GRANTED) {

      ActivityCompat.requestPermissions(this,
       new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
            PETICION_PERMISO_LOCALIZACION);
   } else {

      Location lastLocation =
        LocationServices.FusedLocationApi.getLastLocation(apiClient);

      updateUI(lastLocation);
   }
}

@Override
public void onRequestPermissionsResult(int requestCode, String[]
permissions, int[] grantResults) {
   if (requestCode == PETICION_PERMISO_LOCALIZACION) {
      if (grantResults.length == 1
        && grantResults[0] == PackageManager.PERMISSION_GRAN-
TED) {

         //Permiso concedido

         @SuppressWarnings("MissingPermission")
         Location lastLocation =
                LocationServices.FusedLocationApi.getLastLocation(api-
Client);

         updateUI(lastLocation);

      } else {
         //Permiso denegado:
           //Deberíamos deshabilitar toda la funcionalidad relativa a la
localización.
```

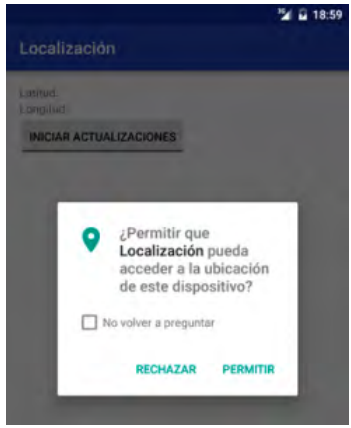
```
            Log.e(LOGTAG, "Permiso denegado");
        }
    }
}
```

Para nuestro ejemplo, en el evento onRequestPermission-Result(), en el caso de obtener los permisos solicitados hacemos exactamente lo mismo que en el evento onConnected(), es decir, obtener la última posición conocida y actualizar la interfaz. En caso de no recibir los permisos por parte del usuario mostramos simplemente un mensaje de error en el log.

Con esto, ya tendríamos una versión inicial, muy básica, de la aplicación de ejemplo, que al iniciarse obtendría y mostraría la última posición recibida en el dispositivo.

Si ejecutamos en este momento la aplicación en el emulador, lo primero que deberíamos ver es la petición del permiso para acceder a la ubicación:

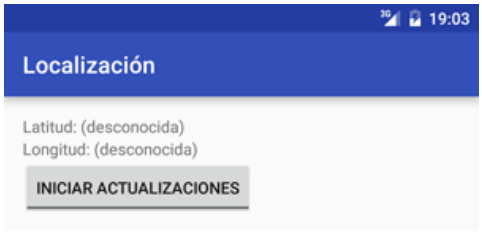
Figura 133. Parte 1: Problema 4.1.1.1 Localization



Fuente: Propia

Si pulsamos en el botón *PERMITIR* del diálogo para aceptar el permiso solicitado por la aplicación pueden pasar dos cosas. Si la aplicación se está ejecutando en un emulador recién creado, o donde nunca se ha ejecutado ninguna aplicación que acceda a ubicaciones es muy posible que obtengamos una localización nula, por lo que se mostraría el literal «*(desconocida)*» en los valores de latitud y longitud.

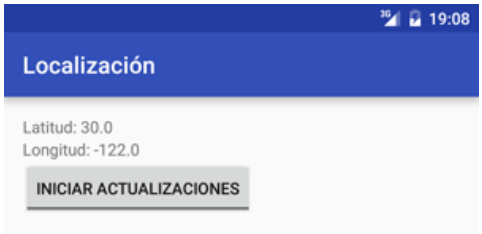
Figura 134. Parte 2: Problema 4.1.1.1 Localization



Fuente: Propia

Si por el contrario ya se había obtenido alguna ubicación en el emulador, o bien si estamos ejecutando la aplicación sobre un dispositivo físico, deberían mostrarse sin problemas los valores correspondientes de latitud y longitud de la ubicación más reciente recibida.

Figura 135. Parte 3: Problema 4.1.1.1 Localization



Fuente: Propia

Si estás en el primer caso, no te preocupes porque pronto lo vamos a arreglar. En el siguiente artículo veremos como activar la recepción periódica de ubicaciones (mediante el botón «Iniciar Actualizaciones» que hemos añadido a la interfaz) de forma que nuestra aplicación esté periódicamente recibiendo, esta vez sí, la posición real y exacta del dispositivo.

4.1.2 Localización geográfica en Android (II)

En el artículo anterior del curso vimos cómo configurar todo lo necesario para acceder a los servicios de localización o ubicación geográfica de Google Play Services, y como primera opción describimos cómo obtener la última localización conocida del dispositivo. Esta opción es una buena forma de conseguir rápidamente una primera ubicación, que en un dispositivo real suele ser bastante aproximada a menos que llevemos siempre desactivadas todas las opciones de ubicación, pero como ya advertimos no siempre puede corresponderse con la ubicación real actual.

En esta entrega vamos a describir cómo solicitar al sistema datos actualizados, esta vez sí, de la posición actual del dispositivo, por supuesto siempre cumpliendo con el nivel de permisos y precisión que hayamos solicitado.

Como ya dijimos, en Android no tenemos ningún método que nos devuelva directamente la posición actual, entre otras cosas porque es impredecible el tiempo que podemos tardar en obtenerla. Seguiremos por tanto otra estrategia, que consistirá en primer lugar en indicar al sistema nuestros *requerimientos*, entre ellos la precisión y periodicidad con que nos gustaría recibir actualizaciones de la posición actual, y en segundo lugar definiremos un método encargado de procesar los nuevos datos a medida que se vayan recibiendo.

Pero antes de esto otro tema importante. En la precisión de los datos obtenidos no solo interviene lo que nuestra aplicación solicite, sino también la configuración del dispositivo que el usuario tenga establecida. Por ejemplo, nuestra aplicación no podría obtener, aunque así lo solicite, una ubicación con máxima precisión si el usuario lleva deshabilitada la *Ubicación* en el dispositivo, o si el modo que tiene seleccionado en las opciones de ubicación de Android no es el de «*Alta precisión*«. Por tanto, un primer paso importante será chequear de alguna forma si las necesidades de nuestra aplicación son coherentes con la configuración actual establecida en el dispositivo, y en caso contrario solicitar al usuario que la modifique siempre que sea posible.

4.1.2.1 Problema:

Vayamos paso a paso. La forma en que nuestra aplicación puede definir sus requerimientos en cuanto a opciones de ubicación será a través de un objeto de tipo LocationRequest. Este objeto almacenará las opciones de ubicación que nuestra aplicación necesita, entre las que destacan:

- Periodicidad de actualizaciones. Se establece mediante el método setInterval() y define cada cuanto tiempo (en milisegundos) *nos gustaría* recibir datos actualizados de la posición. De esta forma, si queremos recibir la nueva posición cada 2 segundos utilizaremos setInterval(2000). ¿Y por qué digo «nos gustaría»? Con este método lo único que damos es nuestra preferencia, pero la periodicidad real podría ser mayor o menor dependiendo de muchas circunstancias (conectividad GPS limitada o intermitente, otras aplicaciones han solicitado periodicidades más altas, …).

- Periodicidad máxima de actualizaciones. El proveedor de localización de Android (*Fused Location Provider*) proporciona actualizaciones de la ubicación con la periodicidad más alta que haya solicitado cualquier aplicación ejecutándose en el dispositivo (éste es uno de los motivos por los que en el apartado anterior indicábamos que es posible recibir actualizaciones a mayor velocidad de la solicitada). Por este motivo, es importante indicar al sistema a qué periodicidad máxima (también en milisegundos) nuestra aplicación es capaz de procesar nuevos datos de ubicación de forma que no nos provoque problemas de rendimiento o sobrecarga. Este dato lo proporcionaremos mediante el método setFastestInterval().
- Precisión. La precisión de los datos que queremos recibir se establecerá mediante el método setPriority(). Existen varios valores posibles para definir esta información:
- PRIORITY_BALANCED_POWER_ACCURACY. Los datos recibidos tendrán una precisión de unos 100 metros. En este modo el dispositivo tendrá un consumo de energía comedido al utilizar normalmente la señal WIFI y de datos móviles para determinar la ubicación.
- PRIORITY_HIGH_ACCURACY. Es el modo más preciso para obtener la ubicación, por lo que utilizará normalmente la señal GPS.
- PRIORITY_LOW_POWER. Los datos recibidos tendrán una precisión de unos 10 kilómetros, pero se utilizará muy poca energía para obtener la ubicación.
- PRIORITY_NO_POWER. En este modo nuestra aplicación solo recibirá datos si éstos están disponibles por-

que *alguna otra aplicación* los haya solicitado. Es decir, nuestra aplicación no tendrá un impacto directo en el consumo de energía solicitando nuevas ubicaciones, pero si éstas están disponibles las utilizará.

Con esta información vamos a definir ya el LocationRequest para nuestro ejemplo. Crearemos un método auxiliar enableLocationUpdates() al que llamaremos desde nuestro botón de iniciar/detener las actualizaciones. Para el ejemplo utilizaremos una periodicidad de 2 segundos, una periodicidad máxima de 1 segundo, y una precisión alta (PRIORITY_HIGH_ACCURACY).

```
private LocationRequest locRequest;

//...

protected void onCreate(Bundle savedInstanceState) {

   //...

   btnActualizar = (ToggleButton) findViewById(R.id.btnActualizar);
   btnActualizar.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
         toggleLocationUpdates(btnActualizar.isChecked());
      }
   });

   //...
}

private void toggleLocationUpdates(boolean enable) {
   if (enable) {
      enableLocationUpdates();
   } else {
```

```
        disableLocationUpdates();
    }
}

private void enableLocationUpdates() {

    locRequest = new LocationRequest();
    locRequest.setInterval(2000);
    locRequest.setFastestInterval(1000);
    locRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCU-
RACY);

    //...
}
```

Definidos nuestros requisitos vamos ahora a comprobar si la configuración actual del dispositivo es coherente con ellos. Para ello construiremos, a continuación dentro de enableLocationUpdates(), un objeto LocationSettingsRequest mediante su *builder*, al que pasaremos el LocationRequest definido en el paso anterior.

```
LocationSettingsRequest locSettingsRequest =
    new LocationSettingsRequest.Builder()
        .addLocationRequest(locRequest)
        .build();
```

Con esto queremos que de alguna forma el sistema compare los requisitos de nuestra aplicación con la configuración actual. ¿Pero cómo ejecutamos y conocemos el resultado de dicha comparación? Para esto llamaremos al método checkLocationSettings() de la API de localización, al que pasaremos la instancia de nuestro cliente API y del LocationSettingsRequest que acaba-

mos de construir. El resultado vendrá dado en forma de objeto PendingResult, del que tendremos de definir su evento onResult() para conocer el resultado de la comparación una vez esté disponible. Este evento recibe como parámetro un objeto LocationSettingsResult, cuyo método getStatus() contiene el resultado de la comparación.

Contemplaremos tres posibles resultados:

- SUCCESS. Significará que la configuración del dispositivo es válida para nuestros requisitos de información.
- RESOLUTION_REQUIRED. Indica que la configuración actual del dispositivo no es suficiente para nuestra aplicación, pero existe una posible solución por parte del usuario (por ejemplo: solicitarle que active la ubicación en el dispositivo o que cambie su modalidad).
- SETTINGS_CHANGE_UNAVAILABLE. Indica que la configuración del dispositivo no es suficiente y además no existe ninguna acción del usuario que pueda solucionarlo.

En el primer caso ya podríamos solicitar el inicio de las actualizaciones de localización, ya que sabemos que la configuración del dispositivo es correcta. En el tercer caso, no nos quedaría más opción que mostrar algún mensaje al usuario indicando que no es posible obtener la ubicación, o bien deshabilitar la funcionalidad relacionada.

Y el caso más interesante, el segundo, necesitamos solicitar al usuario que cambie la configuración del sistema. Por suerte, esta solicitud está ya implementada en la api, por lo que tan sólo tendremos que llamar al método startResolutionForResult() sobre el estado recibido en el evento onResult(). Este método recibe una referencia a la actividad principal y una constante arbitraria

(que podemos definir con cualquier valor único) que después nos servirá para obtener el resultado de la operación.

Veamos todo lo anterior sobre el código.

```
private void enableLocationUpdates() {

   locRequest = new LocationRequest();
   locRequest.setInterval(2000);
   locRequest.setFastestInterval(1000);
   locRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCU-
RACY);

   LocationSettingsRequest locSettingsRequest =
      new LocationSettingsRequest.Builder()
         .addLocationRequest(locRequest)
         .build();

   PendingResult<LocationSettingsResult> result =
      LocationServices.SettingsApi.checkLocationSettings(
            apiClient, locSettingsRequest);

     result.setResultCallback(new  ResultCallback<LocationSettingsRe-
sult>() {
      @Override
       public void onResult(LocationSettingsResult locationSettingsRe-
sult) {
         final Status status = locationSettingsResult.getStatus();
         switch (status.getStatusCode()) {
            case LocationSettingsStatusCodes.SUCCESS:

               Log.i(LOGTAG, "Configuración correcta");
               startLocationUpdates();
               break;

              case LocationSettingsStatusCodes.RESOLUTION_REQUI-
RED:
```

```
                try {
                    Log.i(LOGTAG, "Se requiere actuación del usuario");
                   status.startResolutionForResult(MainActivity.this, PETI-
CION_CONFIG_UBICACION);
                } catch (IntentSender.SendIntentException e) {
                    btnActualizar.setChecked(false);
                Log.i(LOGTAG, "Error al intentar solucionar configuración
de ubicación");
                }
                break;

                case LocationSettingsStatusCodes.SETTINGS_CHANGE_
UNAVAILABLE:
                 Log.i(LOGTAG, "No se puede cumplir la configuración de
ubicación necesaria");
                btnActualizar.setChecked(false);
                break;
            }
        }
    });
}
```

Nos faltaría saber el resultado de la solicitud realizada al usuario para cambiar la configuración en el caso de RESOLUTION_REQUIRED. Para ello sobrescribiremos el método onActivityResult() de la actividad principal, y atenderemos el caso en el que el *requestCode* recibido sea igual a la constante que utilizamos en el método startResolutionForResult(). Existen dos posibles resultados:

- RESULT_OK. Indica que el usuario ha realizado el cambio solicitado. En este caso ya podremos solicitar el inicio de las actualizaciones de ubicación.
- RESULT_CANCELED. Indica que el usuario no ha realizado ningún cambio. En nuestro caso de ejemplo mos-

traremos un error en el log y desactivaremos el botón de inicio de las actualizaciones.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
   switch (requestCode) {
      case PETICION_CONFIG_UBICACION:
         switch (resultCode) {
            case Activity.RESULT_OK:
               startLocationUpdates();
               break;
            case Activity.RESULT_CANCELED:
               Log.i(LOGTAG, "El usuario no ha realizado los cambios de configuración necesarios");
               btnActualizar.setChecked(false);
               break;
         }
         break;
   }
}
```

Pues bien, después de todo esto, ya nos quedaría únicamente saber como solicitar el inicio de las actualizaciones de localización del dispositivo. Esto lo haremos en un método auxiliar startLocationUpdates(). Esta acción es muy sencilla, basta con llamar al método requestLocationUpdates() de la API de localización, pasándole como parámetros nuestro cliente API, el objeto LocationRequest construido al inicio y una referencia al objeto que implementará la interfaz LocationListener, cuyo método onLocationChanged() recibirá los datos de ubicación actualizados. En nuestro caso, haremos que sea nuestra actividad principal la que implemente esta interfaz. Definiremos por tanto el evento indicado en nuestra actividad, que se limitará a llamar a nuestro método auxiliar updateUI() con los nuevos datos recibidos.

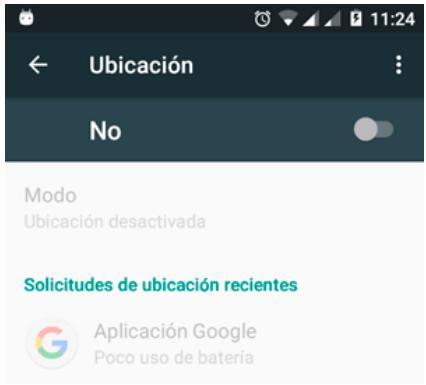
```
public class MainActivity extends AppCompatActivity
      implements GoogleApiClient.OnConnectionFailedListener,
      GoogleApiClient.ConnectionCallbacks,
      LocationListener {

   //...

   private void startLocationUpdates() {
      if (ActivityCompat.checkSelfPermission(MainActivity.this,
            Manifest.permission.ACCESS_FINE_LOCATION) == Packa-
geManager.PERMISSION_GRANTED) {

         //Ojo: estamos suponiendo que ya tenemos concedido el permiso.
         //Sería recomendable implementar la posible petición en caso de
no tenerlo.

         Log.i(LOGTAG, "Inicio de recepción de ubicaciones");

         LocationServices.FusedLocationApi.requestLocationUpdates(
               apiClient, locRequest, MainActivity.this);
      }
   }

   @Override
   public void onLocationChanged(Location location) {

      Log.i(LOGTAG, "Recibida nueva ubicación!");

      //Mostramos la nueva ubicación recibida
      updateUI(location);
   }

   //...
}
```

Por último, para detener la actualización de ubicaciones, tan sólo tendremos que llamar al método removeLocationUpdates() de la API de localización, lo que haremos en un método auxiliar disableLocationUpdates() que a su vez llamaremos cuando corresponda al pulsar el botón de iniciar/detener actualizaciones.

```
private void disableLocationUpdates() {

LocationServices.FusedLocationApi.removeLocationUpdates(

apiClient, this);

}
```

Y con esto habríamos terminado. Ya estaríamos listos para ejecutar la aplicación de ejemplo en el emulador o en un dispositivo real. Para ello, configuraremos primero el dispositivo para desactivar las opciones de ubicación y poder comprobar si nuestra aplicación detecta correctamente esta situación.

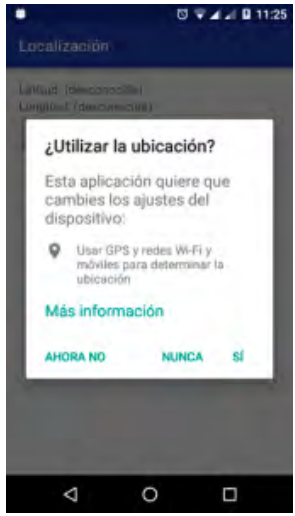
Figura 136. Parte 1: Problema 4.1.2.1 Localization



Fuente: Propia

Ejecutamos ahora la aplicación y aceptamos los permisos de localización si se nos solicita (en Android 6 o superior). Pulsamos el botón de «*INICIAR ACTUALIZACIONES*» y deberíamos ver un diálogo como el siguiente, donde se nos indica que debemos habilitar las opciones de Ubicación para usar la señal móvil, Wi-Fi y GPS (recordemos que hemos solicitado ubicaciones con la máxima precisión).

Figura 137. Parte 2: Problema 4.1.2.1 Localization



Fuente: Propia

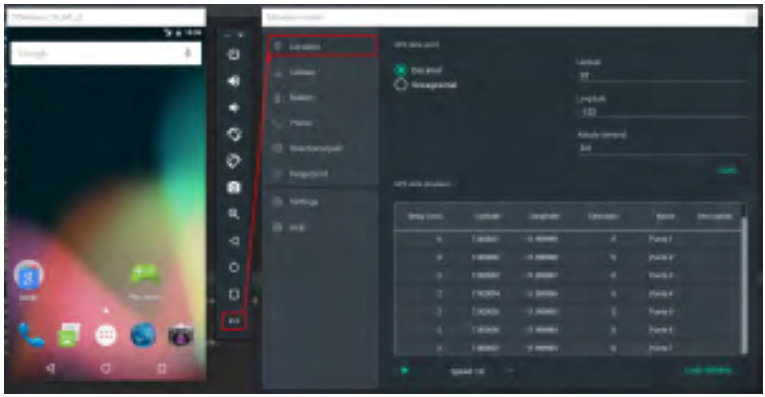
Si pulsamos «SÍ» se habilitarán automáticamente estas opciones (Ubicación activada y modo de «*Alta precisión*«) y nuestra aplicación debería comenzar a recibir actualizaciones de ubicación tal y como habíamos previsto.

Sin embargo, si estamos ejecutando la aplicación en un emulador no veremos ningún cambio en la ubicación. Latitud y Longitud quedarán con valor fijo (última posición conocida) o bien con valor «(desconocido)». ¿Por qué ocurre esto? Muy sencillo, el

emulador, al no ser un dispositivo real, no recibe señal móvil ni GPS, por lo que es incapaz de obtener la ubicación actual a partir de dicha información.

Por suerte, el emulador ofrece un método alternativo para simular que el dispositivo recibe actualizaciones de ubicación. Estas opciones se pueden encontrar accediendo a los controles extendidos del emulador, en la sección «*Location*«.

Figura 138. Parte 3: Problema 4.1.2.1 Localization



Fuente: Propia

Accediendo a estos controles tendremos dos alternativas para enviar ubicaciones a nuestro emulador, una manual y otra automática. La manual, situada en la parte superior, nos permite introducir un valor de latitud-longitud y enviarlo al emulador mediante el botón «SEND», de uno en uno. Si lo hacemos mientras nuestra aplicación se está ejecutando y nuestro botón de actualizaciones está activado, veremos cómo el dato de latitud-longitud introducido en las opciones del emulador aparece en nuestra aplicación (es posible que tarde un poco en aparecer, en general los controles extendidos del emulador van relativamente lentos dependiendo de los recursos del equipo de trabajo).

Figura 139. Parte 4: Problema 4.1.2.1 Localization

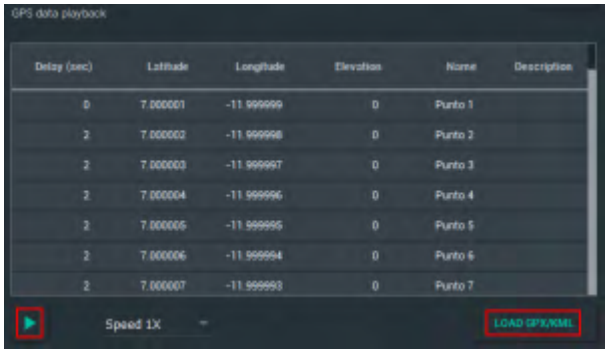
Fuente: Propia

Este método, aunque efectivo, es algo laborioso si queremos probar que nuestra aplicación recibe actualizaciones de la ubicación con cierta frecuencia. Para solucionar esto podemos utilizar la segunda de las opciones, que nos permite automatizar el envío al emulador de un listado de ubicaciones a una cierta velocidad.

El listado de ubicaciones debe estar en formato GPX o KML. En mi caso particular, he elegido KML por su simplicidad. Desde este enlace podéis descargar un fichero KML de ejemplo, que podéis abrir/editar con cualquier editor de texto para adaptarlo a vuestras necesidades. Como podéis comprobar no es más que un listado de pares de latitud-longitud con una estructura muy sencilla de etiquetas tipo XML.

Para cargar este fichero pulsaremos sobre el botón inferior «LOAD GPX/KML» y seleccionaremos nuestro fichero de prueba. Inmediatamente (o como digo, no tan inmediato) aparecerán en la lista superior nuestro listado de ubicaciones. A continuación seleccionaremos la velocidad a la que queremos enviar estos valores al emulador con el desplegable de la parte inferior izquierda (Speed 1X – 5X) y por último pulsamos el botón de «Play» de la izquierda.

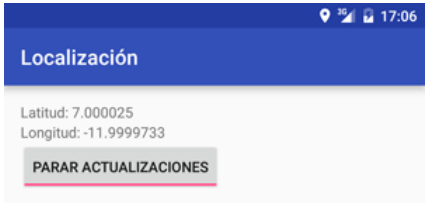
Figura 140. Parte 5: Problema 4.1.2.1 Localization



Fuente: Propia

Hecho esto, ya deberíamos empezar a ver en nuestra aplicación cómo se reciben periódicamente los valores de ubicación de nuestro listado de prueba.

Figura 141. Parte 6: Problema 4.1.2.1 Localization



Fuente: Propia

Y con esto finalizaríamos con los servicios básicos de ubicación de Google Play Services. Hemos aprendido cómo hacer que nuestras aplicaciones obtengan la localización actual del dispositivo, pasando para ello por conocer cómo conectarnos a los servicios correspondientes de Google Play, y cómo lidiar con los permisos y configuraciones requeridas para el acceso a este tipo de información.

4.2 Mapas en Android – Google Maps Android API

No hay duda de que uno de los servicios más populares de Google es, desde hace muchos años, Google Maps. Hoy día recurrimos a él para multitud de tareas, no solo para consultar un mapa, sino también para trazar o calcular rutas, buscar servicios cercanos, escribir o consultar opiniones sobre establecimientos, pasear a pie de calle con Street View, navegar por GPS… En definitiva, este servicio se ha convertido en uno de los fundamentales y está presente en casi cualquier dispositivo Android.

Pero Google Maps no se limita a su propia aplicación, sino que son infinidad las aplicaciones de terceros que integran y utilizan sus servicios con objetivos de todo tipo y color. Por tanto, como desarrolladores Android es importante conocer bien las posibilidades que nos ofrecen los servicios de Google Maps y saber integrarlas en nuestras propias aplicaciones según nuestras necesidades. Adicionalmente, si unimos estos servicios con los de localización geográfica que ya describimos en artículos pasados las posibilidades son infinitas.

4.2.1. Mapas en Android – Google Maps Android API (I)

La API de Google Maps se integró con los *Google Play Services* allá por finales de 2012. Este cambio trajo consigo importantes mejoras, como la utilización de mapas vectoriales y mejoras en el sistema de caché, lo que proporcionaba mayor rendimiento, mayor velocidad de carga, y menor consumo de datos.

También llegó con un cambio en la forma en que los desarrolladores interactuaríamos con los mapas, pasando de los antiguos MapActivity y MapView a un nuevo tipo de *fragment* llamado MapFragment, con las ventajas que conlleva el uso de este tipo de componentes.

Antes de empezar a utilizar esta API en nuestras aplicaciones será necesario realizar algunos preparativos, y es que para hacer uso de los servicios de Google Maps es necesario que previamente generemos una *Clave de API* (o *API key*) asociada a nuestra aplicación. Éste es un proceso sencillo y se realiza accediendo a la Consola de Desarrolladores de Google.

Una vez hemos accedido, tendremos que crear un nuevo proyecto desde la lista desplegable que aparece en la parte superior derecha y seleccionando la opción «Crear proyecto…».

Figura 142. Paso 1: Configurar la API de Google Maps

Fuente: Propia

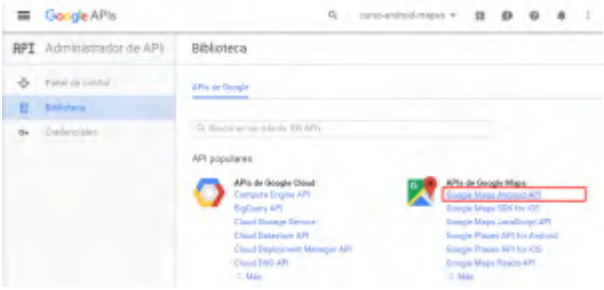
Aparecerá entonces una ventana que nos solicitará el nombre del proyecto. Introducimos algún nombre descriptivo, se generará automáticamente un ID único (que podemos editar aunque no es necesario), y aceptamos pulsando "*Crear*".

Figura 143. Paso 2: Configurar la API de Google Maps

Fuente: Propia

Una vez creado el proyecto llegamos a una página donde se nos permite seleccionar las APIs de Google que vamos a utilizar (como podéis ver la lista es bastante extensa). En nuestro caso particular vamos a seleccionar «*Google Maps Android API*«.

Figura 144. Paso 3: Configurar la API de Google Maps



Fuente: Propia

Aparecerá entonces una ventana informativa con una breve descripción de la API y una advertencia indicando que se necesitará una clave de API para poder utilizarla. Por el momento vamos a activar la API haciendo click sobre la opción «*HABILITAR*» que aparece en la parte superior.

Figura 145. Paso 4: Configurar la API de Google Maps

Fuente: Propia

Una vez activada nos vuelve a aparecer la advertencia sobre la necesidad de obtener credenciales para el uso de la API por lo

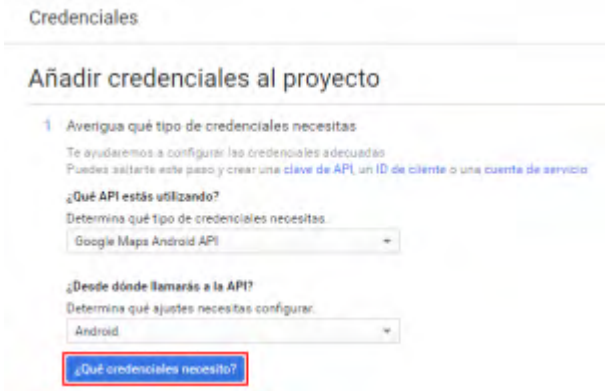
que, ahora sí, tendremos que iniciar el proceso de obtención de la clave. Pulsaremos para ello sobre el botón «*Ir a las credenciales*«.

Figura 146. Paso 5: Configurar la API de Google Maps

Fuente: Propia

Esto iniciará un pequeño asistente. En el primer paso nos volverán a preguntar qué API vamos a utilizar (aunque debería aparecer seleccionada por defecto la de Google Maps para Android), y desde dónde vamos a utilizarla. Indicamos «Android» y pulsamos el botón «*¿Qué tipo de credenciales necesito?*«.

Figura 147. Paso 6: Configurar la API de Google Maps



Fuente: Propia

En el segundo paso tendremos que poner un nombre descriptivo a la clave de API que se va a generar, no es demasiado relevante, por lo que podemos dejar el que nos proponen por defecto. También en este paso se nos da la posibilidad de poder

restringir el uso de este proyecto a determinadas aplicaciones concretas. Como es una práctica baste recomendable vamos a ver cómo hacerlo. Pulsaremos sobre el botón «+ *Añadir nombre de paquete y huella digital*» y veremos que se solicitan dos datos:

- Nombre de paquete
- Huella digital de certificado SHA-1

El primero de ellos es simplemente el paquete java principal que utilizaremos en nuestra aplicación. Lo indicaremos al crear nuestro proyecto en Android Studio (o si ya tenemos una aplicación creada podemos encontrarlo en el fichero *AndroidManifest.xml*, en el atributo package del elemento principal). En mi caso de ejemplo utilizaré el paquete «net.sgoliver.android.mapas».

El segundo de los datos requiere más explicación. Toda aplicación Android debe ir firmada para poder ejecutarse en un dispositivo, tanto físico como emulado. Este proceso de firma es uno de los pasos que tenemos que hacer siempre antes de distribuir públicamente una aplicación. Adicionalmente, durante el desarrollo de la misma, para realizar las pruebas y la depuración del código, aunque no seamos conscientes de ello también estamos firmado la aplicación con un "certificado de pruebas". Pues bien, para obtener la huella digital del certificado con el que estamos firmando la aplicación podemos utilizar la utilidad *keytool* que se proporciona en el SDK de Java. **Importante**: en este ejemplo vamos a obtener el SHA1 del certificado de pruebas, que es el que se utilizará para probar en el emulador, pero en caso de que nuestra aplicación se firmara de nuevo para subirla a la tienda de Google tendríamos que obtener de nuevo el SHA1 del nuevo certificado, o de lo contrario los mapas no se visualizarán.

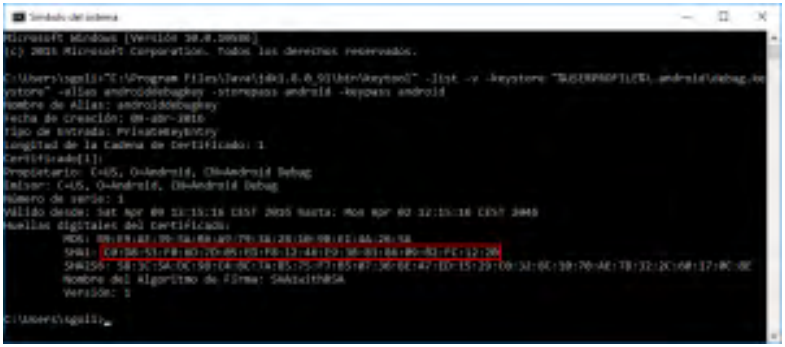
El certificado de pruebas debería encontrarse (en el caso de Windows) en la ruta «<carpeta-usuario-logueado>\.android\de-

bug.keystore». Utilizaremos por tanto el siguiente comando para obtener nuestra huella digital SHA-1:

```
c:\>"C:\Program Files\Java\jdk1.8.0_91\bin\keytool" -list -v -keystore "%USERPROFILE%\.android\debug.keystore" -alias androiddebugkey -storepass android -keypass android
```

Por supuesto que tu instalación de Java está en la ruta "C:\Program Files\Java\jdk1.8.0_91". Si no es así sólo debes sustituir ésta por la correcta.

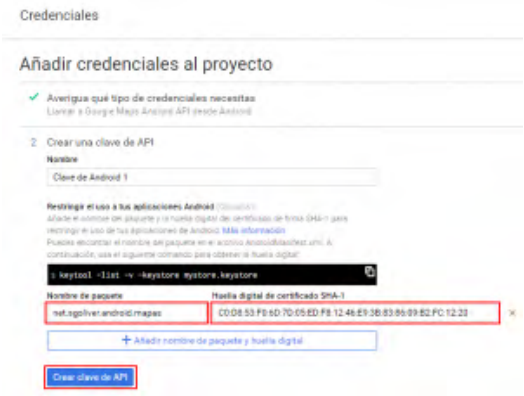
Figura 148. Paso 7: Configurar la API de Google Maps



Fuente: Propia

Una vez tenemos ya los dos datos necesarios (paquete y SHA-1) los introducimos en el asistente de obtención de credenciales y pulsamos el botón «*Crear clave de API*«.

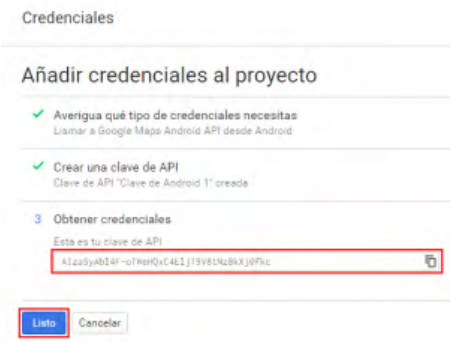
Figura 149. Paso 8: Configurar la API de Google Maps



Fuente: Propia

Hecho esto, obtendremos por fin nuestra clave de API. La copiamos en lugar seguro (más tarde nos hará falta para nuestra aplicación Android), y pulsamos el botón «*Listo*» para finalizar.

Figura 150. Paso 9: Configurar la API de Google Maps



Fuente: Propia

Con esto ya habríamos concluido los preparativos iniciales necesarios para utilizar el servicio de mapas de Android en nuestras aplicaciones, por lo que empecemos a crear un proyecto de ejemplo en Android Studio.

Abriremos Android Studio y crearemos un nuevo proyecto de Android, en mi caso lo he llamado "android-mapas-1", utilizando la plantilla «Empty Activity» y dejando todas las demás opciones por defecto. Recordemos utilizar para el proyecto el mismo paquete java que hemos indicado durante la obtención de la clave de API.

Creado el proyecto, lo primero que haremos será dirigirnos al fichero *build.gradle* de nuestro módulo principal para añadir la referencia a la librería específica de mapas de Google Play Services. En la última actualización de este capítulo la versión más reciente disponible es la 9.4.0, pero es posible que en tu caso puedas ya utilizar alguna versión posterior.

```
dependencies {
   //...
   compile 'com.google.android.gms:play-services-maps:9.4.0'
}
```

A continuación modificaremos el fichero AndroidManifest.xml para añadir tres elementos nuevos. En primer lugar, dentro del elemento <application> tendremos que añadir un nuevo elemento <meta-data> en el que se especificará la versión de los Google Play Services utilizada para compilar la aplicación.

```
<meta-data
   android:name="com.google.android.gms.version"
   android:value="@integer/google_play_services_version" />
```

Inmediatamente después añadiremos otro nuevo <meta-data> donde indicaremos la clave de API para Google Maps que hemos obtenido a través de la consola de desarrolladores.

```
<meta-data
   android:name="com.google.android.geo.API_KEY"
   android:value="AIzaSyAbl4F-oTWeHQxC4EljT9V8tNz8kXj0fkc"/>
```

Por último, también dentro del elemento <application> añadiremos un nuevo elemento <uses-feature> para indicar que nuestra aplicación, o más concretamente Google Maps, hará uso de OpenGL ES versión 2.

```
<uses-feature
  android:glEsVersion="0x00020000"
  android:required="true"/>
```

Y con esto hemos terminado de configurar todo lo necesario. Ya podemos empezar a escribir nuestro código. Para este primer artículo sobre el tema nos vamos a limitar a mostrar un mapa en la pantalla principal de la aplicación. En artículos posteriores veremos como añadir otras opciones o elementos al mapa.

Para esto tendremos simplemente que añadir el control correspondiente al layout de nuestra actividad principal. Lo añadiremos en forma de fragment de tipo com.google.android.gms.maps.MapFragment, quedando de la siguiente forma (no preocuparos si veis algún error de renderizado en el editor visual de layouts de Android Studio):

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/map"
  android:name="com.google.android.gms.maps.MapFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent" />
```

Con esto ya podríamos ejecutar nuestra aplicación y deberíamos ver sin problemas un mapa a pantalla completa en su posición y zoom por defecto. Pero antes vamos a avanzar sólo un poco más para dejar nuestro proyecto preparado para los próximos artículos. Además de visualizar el mapa vamos a obtener

una referencia a él desde nuestro código, referencia a partir de la cual podremos más adelante modificar sus propiedades y realizar determinadas acciones sobre el mapa.

Para esto, vamos a crear en primer lugar un atributo privado de tipo GoogleMap en nuestra actividad principal (que debe heredar de AppCompatActivity), siendo original yo lo llamaré mapa. Haremos además que nuestra actividad implemente la interfaz OnMapReadyCallback, en breve veremos para qué.

En el método onCreate() de la actividad obtendremos en primer lugar una referencia al MapFragment que hemos añadido a nuestro layout a través del fragment manager, y seguidamente llamaremos a su método getMapAsync() pasándole un objeto que implemente la interfaz OnMapReadyCallback, en nuestro caso la propia actividad principal.

Con esto conseguimos que en cuanto esté disponible la referencia al mapa (de tipo GoogleMap) incluido dentro de nuestro MapFragment se llame automáticamente al método onMapReady() de la interfaz. Por el momento la implementación de este método va a ser muy sencilla, limitándonos a guardar el objeto GoogleMap recibido como parámetro en nuestro atributo mapa.

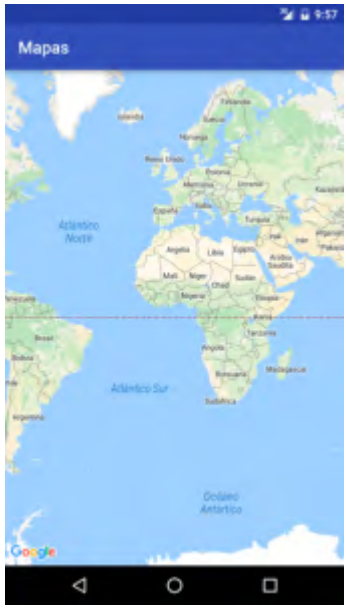
Todo lo anterior quedaría de la siguiente forma:

```
public class MainActivity extends AppCompatActivity
    implements OnMapReadyCallback {

  private GoogleMap mapa;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```

```
        MapFragment mapFragment = (MapFragment) getFragmentMa-
nager()
                .findFragmentById(R.id.map);

        mapFragment.getMapAsync(this);
    }

    @Override
    public void onMapReady(GoogleMap map) {
        mapa = map;
    }
}
```

Y ahora sí, ejecutemos la aplicación para ver el aspecto de nuestro mapa.

Figura 151. Paso 10: Configurar la API de Google Maps



Fuente: Propia

Sobre este mapa, aunque básico, ya podréis moveros y hacer zoom utilizando los gestos clásicos.

Con este artículo espero haber descrito todos los pasos necesarios para comenzar a utilizar los servicios de mapas de Google utilizando la API de Google Maps para Android. Si tenéis cualquier duda o propuesta de mejora no dudéis en escribirlo en los comentarios.

Como habéis podido comprobar hay muchos preparativos que hacer, aunque ninguno de ellos de excesiva dificultad. En los próximos artículos aprenderemos a utilizar más características de esta API.

4.2.2 Mapas en Android – Google Maps Android API (II)

En el artículo anterior del curso vimos cómo realizar todos los preparativos necesarios para comenzar a utilizar la API de Google Maps para Android, a destacar: crear un nuevo proyecto en la consola de desarrolladores, obtener una nueva API Key, y configurar un proyecto básico en Android Studio.

En esta segunda entrega vamos a hacer un repaso de las opciones básicas de un mapa:

- Cambiar las propiedades generales del mapa, como por ejemplo el tipo de vista (satélite, terreno, …) y la activación/desactivación de determinados controles superpuestos.
- Movernos por el mapa de forma programática.
- Obtener los datos de la posición actual mostrada en el mapa.
- Responder a los eventos principales del mapa.

Como aplicación de ejemplo (que podéis descargar al final de este artículo), tomaremos como base la ya creada en el artículo anterior, a la que añadiremos varios botones superiores para demostrar el funcionamiento de las acciones anteriores. El layout lo mantendré muy sencillo, quedaría de la siguiente forma:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
android"
  android:layout_height="match_parent"
  android:layout_width="match_parent" >

  <Button android:id="@+id/btnOpciones"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/opciones" />

  <Button android:id="@+id/btnMover"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toRightOf="@id/btnOpciones"
    android:text="@string/mover" />

  <Button android:id="@+id/btnAnimar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toRightOf="@id/btnMover"
    android:text="@string/animar" />

  <Button android:id="@+id/btnPosicion"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toRightOf="@id/btnAnimar"
    android:text="@string/pos" />

  <fragment
```

```
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.MapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_below="@id/btnOpciones" />

</RelativeLayout>
```

Si hacemos un poco de memoria, recordaremos cómo en la antigua versión de la API de Google Maps era bastante poco homogéneo el acceso y modificación de determinados datos del mapa. Por ejemplo, la consulta de la posición actual o la configuración del tipo de mapa se hacían directamente sobre el control MapView, mientras que la manipulación de la posición y el zoom se hacían a través del controlador asociado al mapa (MapController). Además, el tratamiento de las coordenadas y las unidades utilizadas eran algo peculiares, teniendo estar continuamente convirtiendo de grados a microgrados y de éstos a objetos GeoPoint, etc.

Con la API actual, todas las operaciones se realizarán directamente sobre el objeto GoogleMap, el componente base de la API. En el artículo anterior acabamos viendo cómo obtener, dentro del evento onMapReady(), una referencia al objeto GoogleMap incluido en el MapFragment que añadimos a nuestro layout principal. En este nuevo artículo continuaremos a partir de ese punto para realizar diferentes acciones sobre el mapa.

Así, por ejemplo, para modificar el tipo de vista mostrada en el mapa podremos utilizar una llamada a su método setMapType(), pasando como parámetro el tipo de mapa, a elegir entre los siguientes:

- GoogleMap.MAP_TYPE_NORMAL. Mapa de carreteras normal.

- GoogleMap.MAP_TYPE_SATELLITE. Imágenes de satélite.
- GoogleMap.MAP_TYPE_HYBRID. Una mezcla de los dos anteriores.
- GoogleMap.MAP_TYPE_TERRAIN. Mapa topográfico.

Otra de las opciones que podemos cambiar son los indicadores y controles a mostrar sobre el mapa. Por ejemplo, para que se visualicen los controles de zoom, podemos llamar al método getUiSettings() del mapa para acceder a sus opciones de visualización, y sobre éste al método setZoomControlsEnabled().

Como demostración añadiremos al primero de los botones de nuestra interfaz el cambio de las dos opciones comentadas, que quedaría de la siguiente forma:

```
btnOpciones = (Button)findViewById(R.id.btnOpciones);
btnOpciones.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      cambiarOpciones();
   }
});

//...

private void cambiarOpciones()
{
   mapa.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
   mapa.getUiSettings().setZoomControlsEnabled(true);
}
```

El efecto sobre el mapa en la aplicación sería el siguiente:

Figura 152. Parte 1: Herramientas de Google Maps

Fuente: Propia

En cuanto al movimiento sobre el mapa, podremos mover libremente nuestro punto de vista (o cámara, como lo han llamado los chicos de Android) por un espacio 3D. De esta forma, ya no sólo podremos hablar de latitud-longitud (target) y zoom, sino también de orientación (bearing) y ángulo de visión (tilt). La manipulación de los 2 últimos parámetros unida a posibilidad actual de ver edificios en 3D de muchas ciudades nos abren un mundo de posibilidades.

El movimiento de la cámara se va a realizar siempre mediante la construcción de un objeto CameraUpdate con los parámetros necesarios. Para los movimientos más básicos como la actualización de la latitud y longitud o el nivel de zoom podre-

mos utilizar la clase CameraUpdateFactory y sus métodos estáticos que nos facilitará un poco el trabajo.

Así por ejemplo, para cambiar sólo el nivel de zoom podremos utilizar los siguientes métodos para crear nuestro CameraUpdate:

- CameraUpdateFactory.zoomIn(). Aumenta en 1 el nivel de zoom.
- CameraUpdateFactory.zoomOut(). Disminuye en 1 el nivel de zoom.
- CameraUpdateFactory.zoomTo(nivel_de_zoom). Establece el nivel de zoom.
- Por su parte, para actualizar sólo la latitud-longitud de la cámara podremos utilizar:
- CameraUpdateFactory.newLatLng(lat, long). Establece la lat-lng expresadas en grados.
- Si queremos modificar los dos parámetros anteriores de forma conjunta, también tendremos disponible el método siguiente:
- CameraUpdateFactory.newLatLngZoom(lat, long, zoom). Establece la lat-lng y el zoom.
- Para movernos lateralmente por el mapa (*panning*) podríamos utilizar los métodos de scroll:
- CameraUpdateFactory.scrollBy(scrollHorizontal, scrollVertical). Scroll expresado en píxeles.

Tras construir el objeto CameraUpdate con los parámetros de posición tendremos que llamar a los métodos moveCamera() o animateCamera() de nuestro objeto GoogleMap, dependiendo de si queremos que la actualización de la vista se muestre

directamente en un solo paso o bien de forma animada.

Con esto en cuenta, si quisiéramos por ejemplo centrar la vista en Madrid (España) con un zoom de nivel 5, y lo haremos en el segundo de los botones de la aplicación de ejemplo, podríamos hacer lo siguiente:

```
btnMover = (Button)findViewById(R.id.btnMover);
btnMover.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      moverMadrid();
   }
});

//...

private void moverMadrid()
{
   CameraUpdate camUpd1 =
      CameraUpdateFactory
         .newLatLngZoom(new LatLng(40.41, -3.69), 5);

   mapa.moveCamera(camUpd1);
}
```

Veamos el resultado sobre la aplicación:

Figura 153. Parte 2: Herramientas de Google Maps

Fuente: Propia

Además de los movimientos básicos que hemos comentado, si queremos modificar los demás parámetros de la cámara (orientación e inclinación) o varios de ellos simultáneamente tendremos disponible el método más general CameraUpdateFactory.newCameraPosition() que recibe como parámetro un objeto de tipo CameraPosition. Este objeto los construiremos indicando todos los parámetros de la posición y orientación de la cámara a través de su método Builder().

Así, por ejemplo, si quisiéramos mostrar de una forma más estética el monumento a Alfonso XII de Madrid podríamos hacerlo de la siguiente forma, lo haremos en el tercer botón de demostración de la aplicación de ejemplo:

```
btnAnimar = (Button)findViewById(R.id.btnAnimar);
btnAnimar.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      animarMadrid();
   }
});

//...

private void animarMadrid()
{
   LatLng madrid = new LatLng(40.417325, -3.683081);

   CameraPosition camPos = new CameraPosition.Builder()
      .target(madrid)   //Centramos el mapa en Madrid
      .zoom(19)         //Establecemos el zoom en 19
      .bearing(45)      //Establecemos la orientación con el noreste arriba
      .tilt(70)         //Bajamos el punto de vista de la cámara 70 grados
      .build();

   CameraUpdate camUpd3 =
      CameraUpdateFactory.newCameraPosition(camPos);

   mapa.animateCamera(camUpd3);
}
```

Como podemos comprobar, mediante este mecanismo podemos modificar todos los parámetros de posición de la cámara (o sólo algunos de ellos) al mismo tiempo. En nuestro caso de ejemplo hemos centrado la vista del mapa sobre el parque de El Retiro de Madrid (España), con un nivel de zoom de 19, una orientación de 45 grados para que el noreste esté hacia arriba y un ángulo de visión de 70 grados de forma que veamos en 3D el monumento a Alfonso XII en la vista de mapa NORMAL. En la siguiente imagen vemos el resultado:

Figura 154. Parte 3: Herramientas de Google Maps

Fuente: Propia

Como podéis ver, la API actual de mapas facilita bastante el posicionamiento dentro del mapa, y el uso de las clases CameraUpdate y CameraPosition resulta bastante intuitivo.

Bien, pues ya hemos hablado de cómo modificar nuestro punto de vista sobre el mapa, pero si el usuario se mueve de forma manual por él, ¿cómo podemos conocer en un momento dado la posición de la cámara?

Pues igual de fácil, mediante el método getCameraPosition(), que nos devuelve un objeto CameraPosition como el que ya conocíamos. Accediendo a los distintos métodos y propiedades de este objeto podemos conocer con exactitud la posición de la cámara, la orientación y el nivel de zoom. En la aplicación de ejemplo añadiré la demostración de esto en el cuarto de los bo-

tones añadidos, que mostrará un mensaje *toast* con la posición actual de la cámara.

```
btnPosicion = (Button)findViewById(R.id.btnPosicion);
btnPosicion.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      obtenerPosicion();
   }
});

//...

private void obtenerPosicion()
{
   CameraPosition camPos = mapa.getCameraPosition();

   LatLng coordenadas = camPos.target;
   double latitud = coordenadas.latitude;
   double longitud = coordenadas.longitude;

    Toast.makeText(this, "Lat: " + latitud + " | Long: " + longitud, Toast.
LENGTH_SHORT).show();
}
```

Veamos un ejemplo sobre la propia aplicación:

Figura 155. Parte 4: Herramientas de Google Maps

Fuente: Propia

Por último, vamos a ver cómo responder a ciertos eventos del mapa, entre los que destaco el click y el cambio de posición. Para responder a estos eventos definiremos los *listener* correspondientes sobre nuestro objeto GoogleMap.

Para responder a los clicks del usuario sobre el mapa definiremos el evento onMapClick() llamando al método setOnMapClickListener() del mapa. Dentro de éste recibiremos un objeto LatLng con la posición geográfica, la que mostraremos en un mensaje tipo *toast* junto a la posición X-Y de pantalla (dentro de la vista actual del mapa) donde el usuario ha pulsado. Para obtener esta posición usaremos el método toScreenLocation() del objeto Projection que podemos obtener a partir del mapa llamando a getProjection().

```
mapa.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
   public void onMapClick(LatLng point) {
      Projection proj = mapa.getProjection();
      Point coord = proj.toScreenLocation(point);

      Toast.makeText(
         MainActivity.this,
         "Click\n" +
         "Lat: " + point.latitude + "\n" +
         "Lng: " + point.longitude + "\n" +
         "X: " + coord.x + " - Y: " + coord.y,
         Toast.LENGTH_SHORT).show();
   }
});
```

Un ejemplo sobre la aplicación sería el siguiente:

Figura 156. Parte 5: Herramientas de Google Maps

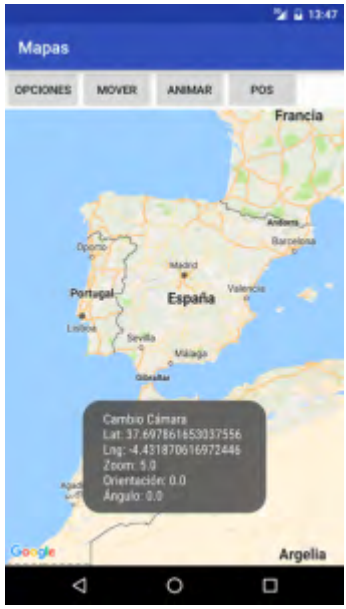
Fuente: Propia

Por su parte, para responder a cambios de posición de la cámara (por ejemplo cuando el usuario desplaza el mapa con el gesto habitual del dedo) debemos definir el evento onCameraChange() llamando al método setOnCameraChangeListener(). Este evento recibe un objeto de tipo CameraPosition, por lo que podremos acceder a cualquier de los datos de posición encapsulados por dicho objeto (latitud, longitud, zoom, orientación, …).

```
mapa.setOnCameraChangeListener(new GoogleMap.OnCameraChangeListener() {
   public void onCameraChange(CameraPosition position) {
      Toast.makeText(
         MainActivity.this,
         "Cambio Cámara\n" +
         "Lat: " + position.target.latitude + "\n" +
         "Lng: " + position.target.longitude + "\n" +
         "Zoom: " + position.zoom + "\n" +
         "Orientación: " + position.bearing + "\n" +
         "Ángulo: " + position.tilt,
         Toast.LENGTH_SHORT).show();
   }
});
```

Sobre nuestra aplicación de ejemplo quedaría de la siguiente forma:

Figura 157. Parte 6: Herramientas de Google Maps



Fuente: Propia

Y con esto habríamos terminado de describir las acciones básicas de configuración y movimiento sobre el mapa. En los próximos artículos veremos más opciones, como la forma de añadir marcadores o dibujar sobre el mapa.

4.2.3 Mapas en Android – Google Maps Android API (III)

En los dos artículos anteriores (I y II) del curso hemos visto cómo crear aplicaciones utilizando la API de Google Maps para Android y hemos descrito las acciones y eventos principales de los mapas.

En este último artículo de la serie nos vamos a centrar en la creación y gestión de marcadores, y en el dibujo de elementos gráficos, como líneas y polígonos, sobre el mapa.

De forma similar al artículo anterior, para esta ocasión vamos a partir de nuevo de la aplicación de ejemplo creada en el primer artículo de la serie sobre mapas, a la que añadiremos tres botones superiores para demostrar las nuevas funcionalidades que presentaremos en esta entrega.

Sin más preámbulos, empecemos por la creación de marcadores. Rara es la aplicación Android que hace uso de mapas sin utilizar también este tipo de elementos para resaltar determinados puntos de interés sobre el mapa. Agregar un marcador básico a un mapa resulta tan sencillo como llamar al método addMarker() pasándole la posición, en forma de objeto LatLng, y el texto a mostrar en la ventana de información del marcador. En nuestra aplicación de ejemplo añadiremos al primer botón una acción de este tipo, de forma que cuando lo pulsemos se añada automáticamente un marcador sobre España con el texto "Pais: España".

```
btnMarcador = (Button)findViewById(R.id.btnMarcador);
btnMarcador.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      insertarMarcador();
   }
});

//...

private void insertarMarcador() {
   mapa.addMarker(new MarkerOptions()
      .position(new LatLng(40.3936945, -3.701519))
      .title("Pais: España"));
}
```

Así de sencillo, basta con llamar al método addMarker() pasando como parámetro un nuevo objeto MarkerOptions sobre el que establecemos la posición del marcador (método position()) y el texto a incluir en la ventana de información del marcador (métodos title() para el título y snippet() para el resto del texto). Si ejecutamos la aplicación de ejemplo y pulsamos el botón "MARCADORES" aparecerá el siguiente marcador sobre el mapa:

Figura 158. Parte 7: Herramientas de Google Maps

Fuente: Propia

Si pulsamos sobre el marcador, la vista se centrará en la posición del marcador y se mostrará la ventana de información con el texto indicado.

Figura 159. Parte 8: Herramientas de Google Maps

Fuente: Propia

Pero vemos que además aparecen por defecto en la esquina inferior derecha dos botones para mostrar dicha ubicación en la aplicación de Google Maps y para calcular la ruta al lugar marcado. Si no queremos que aparezcan estos botones, debemos llamar previamente al método setMapToolbarEnabled(false) sobre nuestro objeto GoogleMap. Podríamos hacer esto por ejemplo dentro del método onMapReady() tras obtener la referencia al mapa:

```
@Override
public void onMapReady(GoogleMap map) {
    mapa = map;

    mapa.getUiSettings().setMapToolbarEnabled(false);
}
```

Si no nos gusta el comportamiento por defecto al pulsar sobre un marcador, también tenemos la posibilidad de definirlo de forma personalizada asignando al mapa dicho evento como cualquiera de los comentados anteriormente. En este caso el listener se asignará al mapa mediante el método setOnMarkerClickListener() y sobrescribiremos el método onMarkerClick(). Dicho método recibe como parámetro el objeto Marker pulsado, de forma que podamos identificarlo accediendo a su información (posición, título, texto, …). Veamos un ejemplo donde mostramos un *toast* con el título del marcador pulsado:

```
mapa.setOnMarkerClickListener(new GoogleMap.OnMarkerClickListener() {
   public boolean onMarkerClick(Marker marker) {
      Toast.makeText(
         MainActivity.this,
         "Marcador pulsado:\n" +
         marker.getTitle(),
         Toast.LENGTH_SHORT).show();

      return true;
   }
});
```

Con el código anterior, si pulsamos sobre el marcador de España aparecerá el siguiente mensaje informativo:

Figura 160. Parte 9: Herramientas de Google Maps

Fuente: Propia

Salvo este último punto, todo lo explicado sobre marcadores corresponde al comportamiento por defecto de la API, sin embargo también es posible por supuesto personalizar determinadas cosas, como por ejemplo el aspecto de los marcadores. Esto se sale un poco del alcance de este artículo, donde pretendía describir los temas más básicos, pero para quien esté interesado tan sólo decir que mediante los métodos icon() y anchor() del objeto MakerOptions que hemos visto antes es posible utilizar una imagen personalizada para mostrar como marcador en el mapa. En la documentación oficial (en inglés) podéis encontrar un ejemplo de cómo hacer esto.

Como último tema, vamos a ver cómo dibujar líneas y polígonos sobre el mapa, elementos muy comúnmente utilizados para trazar rutas o delimitar zonas del mapa. Para realizar esto

vamos a actuar una vez más directamente sobre la vista de mapa, sin necesidad de añadir *overlays* o similares (para quienes recuerden versiones anteriores de la API), y ayudándonos de los objetos PolylineOptions y PolygonOptions respectivamente.

Para dibujar una linea lo primero que tendremos que hacer será crear un nuevo objeto PolylineOptions sobre el que añadiremos, utilizando su método add(), las coordenadas (latitud-longitud) de todos los puntos que conformen la linea. Tras esto estableceremos el grosor y color de la linea llamando a los métodos width() y color() respectivamente, y por último añadiremos la linea al mapa mediante su método addPolyline() pasándole el objeto PolylineOptions recién creado.

En nuestra aplicación de ejemplo haremos esto en el segundo botón de demostración para dibujar un rectángulo sobre España. Veamos cómo queda el código:

```
btnLineas = (Button)findViewById(R.id.btnLineas);
btnLineas.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      mostrarLineas();
   }
});

//...

private void mostrarLineas()
{
   //Dibujo con Lineas

   PolylineOptions lineas = new PolylineOptions()
      .add(new LatLng(45.0, -12.0))
      .add(new LatLng(45.0, 5.0))
```

```
        .add(new LatLng(34.5, 5.0))
        .add(new LatLng(34.5, -12.0))
        .add(new LatLng(45.0, -12.0));

    lineas.width(8);
    lineas.color(Color.RED);

    mapa.addPolyline(lineas);
}
```

Ejecutando esta acción en el emulador veríamos lo siguiente:

Figura 161. Parte 10: Herramientas de Google Maps

Fuente: Propia

Pues bien, esto mismo podríamos haberlo logrado mediante el dibujo de polígonos, cuyo funcionamiento es muy similar. Para ello crearíamos un nuevo objeto PolygonOptions y añadi-

remos las coordenadas de sus puntos en el sentido de las agujas del reloj. En este caso no es necesario cerrar el circuito (es decir, que la primera coordenada y la última fueran iguales) ya que se hace de forma automática. Otra diferencia es que para polígonos el ancho y color de la linea los estableceríamos mediante los métodos strokeWidth() y strokeColor(). Además, el dibujo final del polígono sobre el mapa lo haríamos mediante addPolygon(). En nuestro caso de ejemplo, implementado en el tercer botón de demostración, quedaría como sigue:

```
btnPoligono = (Button)findViewById(R.id.btnPoligono);
btnPoligono.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      mostrarPoligono();
   }
});

//...

private void mostrarPoligono()
{
   //Dibujo con Poligonos
   PolygonOptions rectangulo = new PolygonOptions()
      .add(new LatLng(45.0, -12.0),
         new LatLng(45.0, 5.0),
         new LatLng(34.5, 5.0),
         new LatLng(34.5, -12.0),
         new LatLng(45.0, -12.0));

   rectangulo.strokeWidth(8);
   rectangulo.strokeColor(Color.RED);

   mapa.addPolygon(rectangulo);
}
```

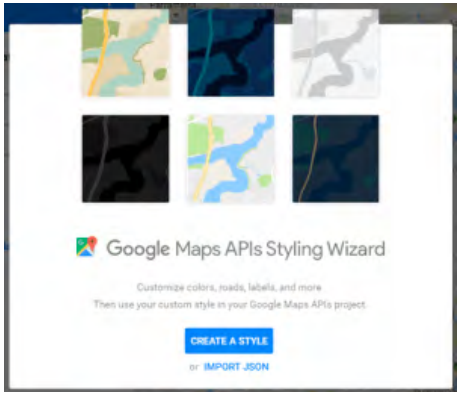
El resultado al ejecutar la acción en el emulador debería ser exactamente igual que el anterior.

4.2.4 Mapas en Android – Google Maps Android API (IV)

Con la nueva versión de los Google Play Services 9.6 (septiembre 2016, novedades) nos ha llegado la posibilidad de editar fácilmente y casi al detalle la apariencia completa de los mapas que utilizamos a través de la API de Google Maps para Android (aunque también para iOS y Web). Google lo anunciaba ayer mismo en su blog *Geo Developers Blog*, donde daba a conocer las novedades más importantes a este respecto.

Esta novedad nos llega en forma de nueva herramienta web, a la que han llamado Google Maps APIs Styling Wizard, con la que podremos dar el estilo que deseemos a nuestros mapas, todo de forma muy visual e intuitiva. Una vez definido el estilo que necesitamos podremos exportarlo a un fichero JSON que más tarde podremos utilizar para asignar dicho estilo a los mapas mostrados por nuestra aplicación Android.

Figura 162. Parte 1: Google Maps APIs Styling Wizard

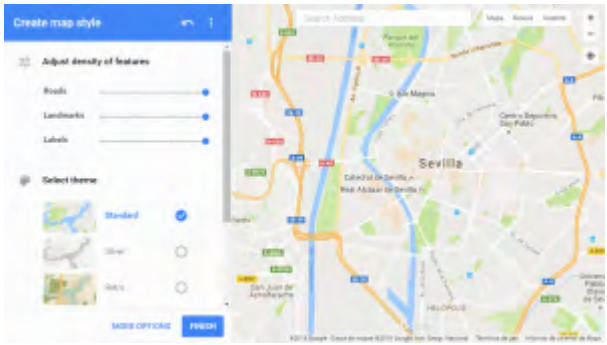


Fuente: Propia

Si entramos por primera vez a esta nueva herramienta veremos la pantalla inicial que se muestra en la imagen anterior, donde se muestra una pequeña descripción y se nos invita a crear un nuevo estilo de mapa o a importar uno ya creado.

Si seleccionamos la opción de crear nuevo estilo veremos a la izquierda el menú principal de la herramienta y a la derecha una muestra de mapa, que podremos situar en la localización que deseemos, donde iremos viendo en vivo los cambios que vayamos realizando.

Figura 163. Parte 2: Google Maps APIs Styling Wizard

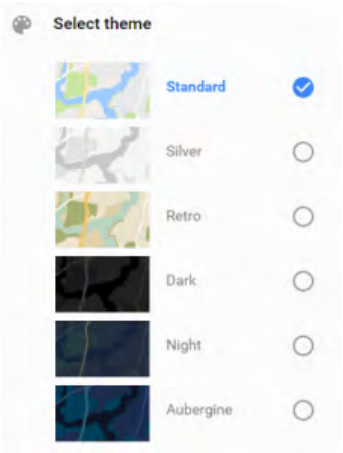


Fuente: Propia

Por ilustrar la utilidad de esta nueva herramienta y seguir un ejemplo práctico, imaginemos que en nuestra aplicación Android queremos mostrar o resaltar de alguna forma sólo las zonas verdes de cada ciudad (parques, jardines, …).

La forma más sencilla de trabajar con Google Maps API Styling Wizard será partir de uno de los temas predefinidos que nos propone la herramienta, seleccionando aquel que más se acerque al aspecto visual que queremos conseguir. Tenemos 6 temas predefinidos entre los que elegir (3 claros y 3 oscuros):

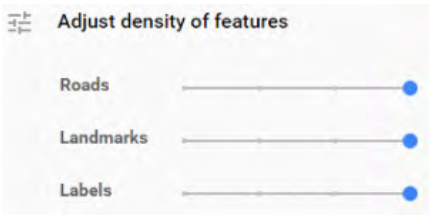
Figura 164. Parte 3: Google Maps APIs Styling Wizard



Fuente: Propia

Tras elegir el tema más parecido al estilo que estamos buscando nos dirigiremos a los controles superiores, apartado llamado «*Adjust density of features*«, donde podremos seleccionar, a grandes rasgos, el grado de detalle de los distintos elementos que se muestran sobre el mapa, por ejemplo las calles y carreteras, los puntos de interés, tipos de zonas, etiquetas, etc. Ajustaremos los controles *Roads*, *Landmarks* y *Labels* hasta encontrar nuevamente el grado de detalle que más se acerque al objetivo que perseguimos.

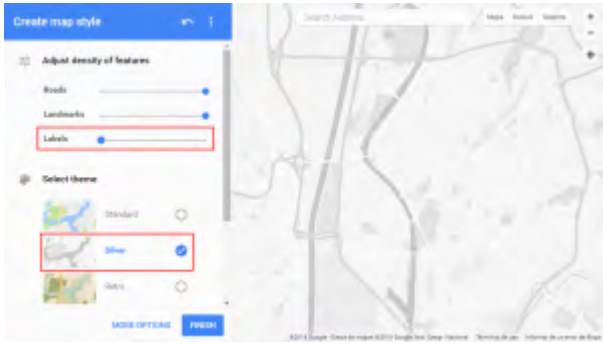
Figura 165. Parte 4: Google Maps APIs Styling Wizard



Fuente: Propia

Para mi ejemplo utilizaré el tema «*Silver*» y bajaré al máximo el nivel de detalle del control *Labels* (etiquetas), de forma que consigamos un mapa más plano, más sencillo, sin tantas distracciones, y donde se puedan distinguir mejor los diferentes tipos de zonas de la ciudad. Quedaría por ahora de la siguiente forma (comparar con la imagen anterior):

Figura 166. Parte 5: Google Maps APIs Styling Wizard



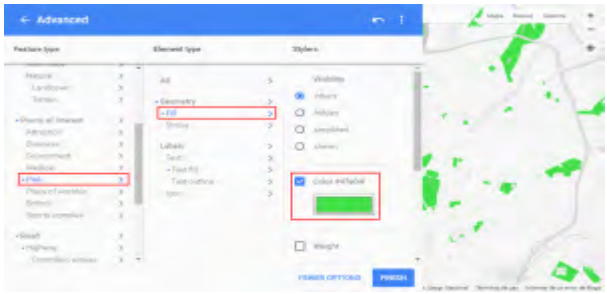
Fuente: Propia

Ya tenemos un punto de partida sobre el que poder trabajar. Para poder afinar los detalles de nuestro estilo personalizado pulsaremos sobre la opción «MORE OPTIONS» de la parte inferior del menú. Esto nos dará acceso a las opciones avanzadas, donde podremos ajustar de forma independiente y en detalle el estilo de cada elemento del mapa, por ejemplo los elementos administrativos (ciudades, localidades, barrios, …), los tipos de terreno, los puntos de interés, los caminos, carreteras, lineas de transporte público, … infinitas posibilidades.

En nuestro caso particular queremos resaltar las zonas verdes, por lo que nos dirigiremos al apartado «Points of interest» (Puntos de interés) y dentro de éste al subapartado «Park» (Parque). Una vez allí la herramienta nos permite modificar la geo-

metría (trazos y rellenos de color) y etiquetas (texto e icono) de este tipo de elementos. Nosotros modificaremos el color a verde de forma que resalte sobre los colores grises que hemos asignado al resto del mapa. Para ello iremos al apartado «Geometry» (geometría) / «Fill» (relleno), activaremos la check de «Color», y seleccionaremos el color deseado.

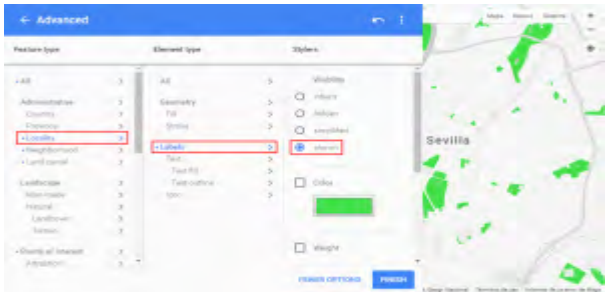
Figura 167. Parte 6: Google Maps APIs Styling Wizard



Fuente: Propia

Dado que en los primeros pasos redujimos al máximo la densidad de etiquetas, en este momento no aparece ninguna etiqueta sobre nuestro mapa. Sin embargo, para el ejemplo me gustaría que al menos se mostraran los nombres de las distintas localidades sobre el mapa. Para ello vamos ahora al apartado «Administrative» / «Locality». Una vez allí, abrimos el apartado «Labels» y le indicamos que queremos mostrarlas (valor «Shown») para este tipo de elementos:

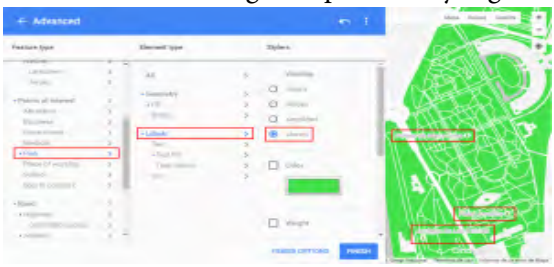
Figura 168. Parte 7: Google Maps APIs Styling Wizard



Fuente: Propia

Ya podemos ver en la imagen anterior como se muestra la etiqueta «Sevilla» sobre el mapa. Casi hemos llegado al resultado que buscábamos, pero si ampliamos el mapa sobre alguna de las zonas verdes resaltadas veremos que no se indica tampoco el nombre de dicha zona. Tendremos por tanto que activar también las etiquetas para estos elementos, así que volvemos al menú «Points of interest» / «Park» y nuevamente seleccionamos el valor «Shown» en el apartado «Labels»:

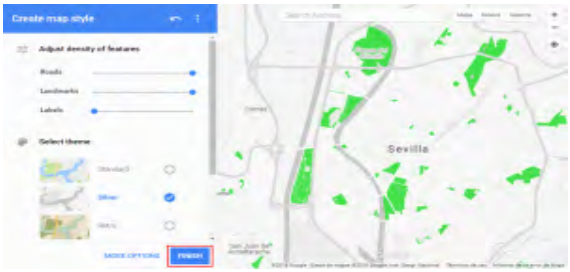
Figura 169. Parte 8: Google Maps APIs Styling Wizard



Fuente: Propia

Y con esto ya habríamos finalizado nuestro estilo personalizado. Por brevedad no me he extendido demasiado pero espero haber ilustrado las posibilidades que nos ofrece esta herramienta, os invito a experimentar por vosotros mismos para descubrir todas las opciones de estilo que se ofrecen.

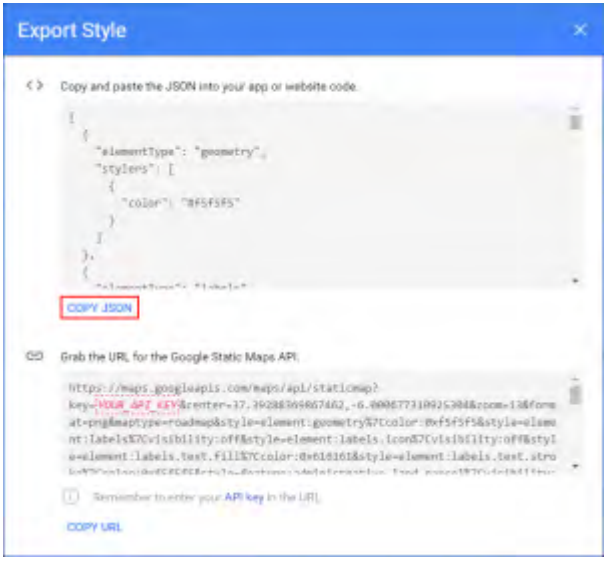
Figura 170. Parte 9: Google Maps APIs Styling Wizard



Fuente: Propia

Ya podemos por tanto exportar el estilo creado a formato JSON para utilizarlo desde nuestra aplicación Android. Pulsaremos para ello el botón «FINISH» (ver imagen anterior) y nos aparecerá el cuadro de diálogo siguiente:

Figura 171. Parte 10: Google Maps APIs Styling Wizard



Fuente: Propia

En este cuadro podemos pulsar la opción «COPY JSON» para copiar el contenido completo y lo pegaremos y guardaremos en un fichero de texto con extensión «.json». En mi caso lo he llamado «*formato_mapa.json*«.

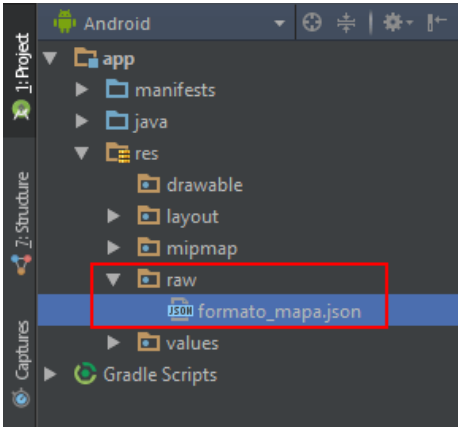
Para la aplicación Android de ejemplo tomaré como base la ya creada en el primer artículo de la serie sobre Google maps.

Lo primero que deberemos hacer será añadir la referencia a la última versión de la librería de mapas de Google Play Services (**versión 9.6**) a nuestro fichero *build.gradle*:

```
dependencies {
   //...
   compile 'com.google.android.gms:play-services-maps:9.6.0'
}
```

Lo siguiente será crear (si no existe ya) una carpeta «raw» en la carpeta de recursos del proyecto. Esta carpeta debe quedar en la ruta «\nombre-del-proyecto\app\src\main\res\raw«. Dentro de esta carpeta colocaremos el fichero de formato de mapa («*formato_mapa.json*«) que hemos creado anteriormente. En Android Studio debería refrescarse la vista de proyecto automáticamente para mostrar estos cambios:

Figura 172. Parte 11: Google Maps APIs Styling Wizard



Fuente: Propia

Ya solo nos quedaría utilizar este nuevo recurso del proyecto para dar formato a nuestro mapa. Hacer esto es tan sencillo como llamar al método setMapStyle() del objeto GoogleMap obtenido en el evento onMapReady(), pasándole como parámetro el nuevo recurso. Para cargar este recurso utilizaremos el método MapStyleOptions.loadRawResourceStyle() pasándole como ID el nombre del recurso precedido de R.raw. Veamos cómo quedaría el código completo:

```
public class MainActivity extends AppCompatActivity
    implements OnMapReadyCallback {

  private GoogleMap mapa;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

     MapFragment mapFragment = (MapFragment) getFragmentMa-
```

```
nager()
            .findFragmentById(R.id.map);

      mapFragment.getMapAsync(this);
   }

   @Override
   public void onMapReady(GoogleMap map) {
      mapa = map;

      //Aplicamos el estilo personalizado de mapa
      mapa.setMapStyle(
         MapStyleOptions.loadRawResourceStyle(
            this, R.raw.formato_mapa));
   }
}
```

Si ejecutamos ahora el proyecto sobre un emulador con los Google Play Services actualizados a la última versión (9.6) o sobre algún dispositivo real actualizado podremos ver nuestro mapa con el estilo personalizado que hemos definido:

Figura 173. Parte 12: Google Maps APIs Styling Wizard

Fuente: Propia

Con esto habríamos terminado este artículo dedicado a una de las novedades más relevantes que nos han llegado con la última versión de los Google Play Services. Espero que os sea de ayuda.

Pasos SHA-1

Ver SHA

https://www.youtube.com/watch?v=jxA—phMORk

CMD

```
cd C:\Program Files\Android\Android Studio\jre\jre\bin
keytool -list -v -keystore C:\Users\USER-COMPUTER\.android\debug.keystore -alias androiddebugkey -storepass android -keypass android
```

Ejemplos en github APIS de Google

https://github.com/sgolivernet

4.2.5 Evaluación 5

1. ¿Cuál es el proveedor de localización geográfica en la que no tenemos que preocuparnos al menos no de forma explícita, de qué queremos utilizar en cada momento ya que se encargará automáticamente de gestionar todas las fuentes de datos disponibles para obtener la información que nuestra aplicación necesita?
2. ¿Cuál es la dependencia que contiene la librería completa de Google Play Services que nos daría acceso a todos los servicios disponibles?
3. Mediante cual método podemos acceder al objeto Location para obtener la latitud de una coordenada geográfica.
4. ¿Cuál método de la localización permite obtener la última posición geográfica conocida?
5. ¿Cuál de los permisos relacionados con la ubicación geográfica en Android permite acceder a datos de localización con una precisión baja, añadiendo la cláusula correspondiente <uses-permission> a nuestro fichero AndroidManifest.xml?
6. Mediante el método setInterval() se puede establecer la periodicidad de actualizaciones de la localización geográfica. ¿Cuál de las siguientes opciones está establecido, si queremos recibir la nueva posición cada 2 segundos?
7. La precisión en la localización de los datos que queremos recibir se establecerá mediante el método setPriority(). ¿Qué valores define el modo PRIORITY_BALANCED_POWER_ACCURACY?
8. ¿En qué modo la precisión en la localización geográfica

de los datos del método setPriority() es más preciso para obtener la ubicación, por lo que utilizará normalmente la señal GPS?

9. Para saber cómo solicitar el inicio de las actualizaciones de localización del dispositivo. ¿Cuál es el método que me ayuda en esta acción?

10. Cuando queremos que de alguna forma el sistema compare los requisitos de nuestra aplicación con la configuración actual de la API de localización se usa el evento que recibe como parámetro un objeto LocationSettingsResult, cuyo método getStatus() contiene el resultado de la comparación. ¿Cuál resultado indica que la configuración actual del dispositivo no es suficiente para nuestra aplicación, pero existe una posible solución por parte del usuario?

11. ¿Qué tarea nos permite realizar el servicio de Google Maps?

12. Actualmente que componente los desarrolladores usan para interactuar con los mapas

13. El método setMapType() permite modificar el tipo de vista mostrada en el mapa cual parámetro nos permite ver en mapa topográfico

14. Para los movimientos básicos en el Mapa se utiliza la clase CameraUpdateFactory. ¿Cuál método establece la latitud, longitud y el zoom para movernos lateralmente por el mapa?

15. ¿Cuál es el evento para responder a los clicks del usuario sobre el mapa?

16. ¿Cuál es el evento que detecta cambios de posición de la cámara en el mapa?

17. En la herramienta de Google Maps cuál método nos permite agregar un marcador básico a un mapa.
18. Si queremos modificar los demás parámetros de la cámara del Mapa con el objeto de tipo CameraPosition. ¿Cuál es el parámetro del ángulo de visión o vista?
19. Mediante cual método del objeto CameraPosition del Mapa podemos conocer con exactitud la posición de la cámara, la orientación y el nivel de zoom.
20. ¿Qué herramienta web nos permite dar estilos a los mapas?

Firebase Cloud Storage

4.3 Firebase Cloud Storage

Cloud Storage se creó para desarrolladores de apps que necesitan almacenar y entregar contenido generado por usuarios, como fotos o videos.

Cloud Storage para Firebase es un servicio de almacenamiento de objetos potente, simple y rentable construido para la escala de Google. Los SDK de Firebase para Cloud Storage agregan la seguridad de Google a las operaciones de carga y descarga de archivos para tus apps de Firebase, sin importar la calidad de la red. Puedes usar nuestros SDK para almacenar imágenes, audio, video y otros tipos de contenido generado por el usuario. En el servidor, puedes usar Google Cloud Storage para acceder a los mismos archivos.

Tabla 1. Características de Firebase Cloud Storage

Funciones clave	Descripción
Operaciones robustas	Los SDK de Firebase para Cloud Storage realizan las operaciones de carga y descarga sin importar la calidad de la red. Las cargas y descargas son robustas, lo que significa que se reinician en el punto en el que se interrumpieron para así ahorrar tiempo y ancho de banda a los usuarios.
Seguridad sólida	Los SDK de Firebase para Cloud Storage se integran con Firebase Authentication a fin de brindar autenticación intuitiva y sencilla para los programadores. Puedes usar nuestro modelo de seguridad declarativa para permitir el acceso según el nombre de archivo, el tamaño, el tipo de contenido y otros metadatos.
Gran escalabilidad	Cloud Storage para Firebase está diseñado para escalar a exabytes si tu app se vuelve viral. Pasa fácilmente de la fase prototipo a la de producción con la misma infraestructura que respalda a Spotify y Google Fotos.

Fuente: Documentación de Firebase Obtenido de https://firebase.google.com/docs/storage?hl=es-419

¿Cómo funciona?

Los programadores usan los SDK de Firebase para Cloud Storage a fin de subir y descargar archivos directamente de los clientes. Si la conexión a la red es deficiente, el cliente puede reintentar la operación donde la dejó de inmediato, lo cual les ahorra tiempo y ancho de banda a los usuarios.

Cloud Storage almacena tus archivos en un depósito de Google Cloud Storage y los hace accesibles a través de Firebase y Google Cloud. Esto te permite tener la flexibilidad para subir y descargar archivos de clientes móviles a través de los SDK de Firebase y realizar procesamiento en el servidor, como el filtrado de imágenes o la transcodificación de videos mediante Google Cloud Platform. Cloud Storage se escala automáticamente, lo que significa que no es necesario migrar a ningún otro proveedor.

Obtén más información acerca de todos los beneficios de nuestra integración a Google Cloud Platform.

Los SDK de Firebase para Cloud Storage se integran perfectamente en Firebase Authentication a fin de identificar a los usuarios. Además, ofrecemos un lenguaje de seguridad declarativo que te permite configurar controles de acceso para archivos individuales o grupos de archivos, de manera que puedas hacer que los archivos sean públicos o privados según lo que desees.

Tabla 2. SDK de Firebase para Cloud Storage

Ruta de implementación	Descripción
Integra los SDK de Firebase para Cloud Storage.	Incluye clientes rápidamente mediante Gradle, CocoaPods o una secuencia de comandos.
Crea una referencia.	Haz una referencia de la ruta al archivo (por ejemplo, "images/montañas.png") que subirás, descargarás o borrarás.
Sube o descarga.	Sube o descarga en tipos nativos en la memoria o en el disco.
Protege tus archivos.	Protege tus archivos con las reglas de seguridad de Firebase para Cloud Storage.

Fuente: Documentación de Firebase obtenido de https://firebase.google.com/docs/storage?hl=es-419

¿Quieres almacenar otros tipos de datos?

- Cloud Firestore es una base de datos flexible y escalable para el desarrollo en servidores, dispositivos móviles y la Web desde Firebase y Google Cloud Platform.
- Firebase Realtime Database almacena datos de aplicación JSON, como estados de juego o mensajes de chat, y sincroniza los cambios en todos los dispositivos conectados en forma instantánea. Para obtener más información sobre las diferencias entre opciones de bases de

datos, consulta Elige una base de datos: Cloud Firestore o Realtime Database.

- Firebase Remote Config almacena pares clave-valor especificados por el desarrollador para cambiar el comportamiento y el aspecto de la app sin que los usuarios tengan que descargar una actualización.
- Firebase Hosting aloja elementos HTML, CSS y JavaScript para el sitio web, así como otros elementos suministrados por el desarrollador, como gráficos, íconos y fuentes.

4.3.1 Problema

Se desarrollará una aplicación móvil que permita mostrar una lista de equipos de fútbol, sus datos se encontran almacenados en una base de datos Realtime Database y los escudos de cada equipo se encontrarán almacenados en el Cloud Storage de Firebase.

4.4 Firebase Cloud Messaging

4.4.1 Notificaciones Push en Android: Firebase Cloud Messaging (I)

Firebase Cloud Messaging (FCM) es sin duda otro de los servicios estrella que nos ofrece la plataforma Firebase. Es posible que hayáis oído hablar de este servicio con otros nombres, como mensajería en la nube o mensajería *push*, pero todos se refieren a lo mismo. Incluso puede que os suene el servicio *Google Cloud Messaging (GCM)*, que no es más que la «versión anterior» de este servicio de mensajería de Google, recientemente trasladado a la plataforma Firebase.

Si has llegado hasta aquí quiero suponer que ya sabes en qué consiste un servicio de este tipo, pero aún así voy a intentar describirlo muy brevemente. Firebase Cloud Messaging permite el envío de mensajes entre un servidor de aplicaciones en la nube y un dispositivo Android (en nuestro caso particular , aunque también podría ser iOS o Web). Aunque en la actualidad también soporta el envío de mensajes en sentido ascendente (desde el dispositivo hacia el servidor), son los mensajes descendentes, de servidor a cliente, los que más nos van a interesar. Estos mensajes nos van a permitir implementar funcionalidades de todo tipo, por ejemplo el envío de notificaciones de usuario a un dispositivo o grupo de dispositivos, o el envío de notificaciones internas (no visibles para el usuario) para «avisar» a nuestra aplicación de que tiene nueva información disponible para descargar. Por ejemplo esto último es muy interesante para evitar que nuestras aplicaciones deban estar constantemente accediendo a la red para comprobar si hay nueva información disponible y así reducir enormemente el gasto de energía.

Durante los próximos artículos iremos descubriendo poco a poco las distintas posibilidades que nos ofrece Firebase Cloud Messaging. En este primer artículo veremos los primeros pasos a seguir, la creación de un proyecto muy sencillo en Android Studio, y enviaremos nuestro primer mensaje.

Al igual que ocurría con muchos de los servicios de Google Play Services o con la base de datos de Firebase que ya conocemos, antes de ponernos a trabajar en nuestro proyecto de Android Studio tendremos dirigirnos a la consola de desarrolladores, en esta ocasión la Consola de Firebase, para crear y configurar el nuevo proyecto y asociar a él nuestra aplicación.

4.4.1.1 Problema:

Si es la primera vez que accedemos a la consola de Firebase, nos aparecerá un mensaje de bienvenida que directamente nos invita a empezar a crear un nuevo proyecto.

Si pulsamos la opción de «CREAR NUEVO PROYECTO» el sistema nos solicitará un nombre identificativo y la región a la que perteneces.

Con estos simples datos quedaría creado el nuevo proyecto de Firebase. El siguiente paso será asociar una aplicación a dicho proyecto, en nuestro caso una aplicación Android, para lo que pulsaremos en la opción correspondiente a dicho sistema.

Esto iniciará un pequeño asistente donde se nos solicitarán algunos datos que ya deberían sonarnos de artículos anteriores. Tendremos que indicar el paquete java principal de nuestra aplicación Android, yo usaré net.sgoliver.android.fcm, y la huella digital SHA-1 del certificado con el que se firmará la aplicación, en principio usaremos como siempre la correspondiente al certificado de pruebas (tienes más información sobre cómo obtener este dato por ejemplo en el primer artículo sobre mapas), pero recuerda cambiarlo por el certificado de producción si subes tu aplicación a Google Play.

En el segundo paso podrás descargar un fichero de configuración, en formato JSON, que tendremos que añadir a la aplicación Android una vez creemos el proyecto en Android Studio. Al pulsar el botón CONTINUAR se descargará el fichero, más adelante veremos qué hacer con él (aunque ya hicimos algo muy similar cuando tratamos el tema de autenticación mediante Google Sign-In).

En el tercer y último paso se ofrecen una serie de indicaciones sobre cómo configurar todas las dependencias necesarias en

el proyecto de Android Studio. Esto lo veremos más adelante, por lo que ya podemos FINALIZAR el asistente.

Con esto ya tendríamos configurado todo lo necesario en la consola de Firebase. En este momento deberíamos estar viendo tanto nuestro proyecto (en la parte superior izquierda) como nuestra aplicación Android asociada (en la parte central). No debemos confundir una entidad con otra. Un mismo proyecto de Firebase puede tener asociadas varias aplicaciones, que además pueden ser de Sistemas distintos (Android, IOS o Web).

Finalizados todos los preparativos en la consola de Firebase, podemos disponernos ya a crear nuestro proyecto en Android Studio. Crearemos un proyecto estándar, con API mínima 15 y utilizando la plantilla *Empty Activity*.

Creado el proyecto lo primero que vamos a hacer es colocar el fichero de configuración «*google-services.json*» que hemos descargado antes en su ubicación correcta. Debemos copiarlo a la carpeta «/app» situada dentro de la carpeta de nuestro proyecto. En mi caso particular he llamado al proyecto «android-fcm-1», por lo que el fichero de configuración debe ir colocado en la carpeta «\android-fcm-1\app».

El siguiente paso será añadir todas las dependencias necesarias a nuestros ficheros *build.gradle*. Tendremos que modificar tanto el fichero *build.gradle* situado a nivel de proyecto, como el de nuestro módulo principal (¿no sabes cuáles son estos ficheros? Consúltalo aquí).

Empezaremos por el *build.gradle* de proyecto. Aquí tendremos que añadir a la sección de dependencias la referencia al plugin de Google Play Services para Gradle.

```
buildscript {
   repositories {
      jcenter()
   }
   dependencies {
      //...
      classpath 'com.google.gms:google-services:3.0.0'
   }
}
```

Por su parte, en el *build.gradle* del módulo principal, tendremos por un lado que aplicar dicho plugin al final del fichero, y por otro añadir la referencia a los servicios de Firebase que vayamos a utilizar. En nuestro caso utilizaremos la funcionalidad de mensajería en la nube o Firebase Cloud Messaging (FCM), por lo que añadiremos tan solo su librería correspondiente (puedes consultar el listado completo de librerías disponibles en la documentación oficial).

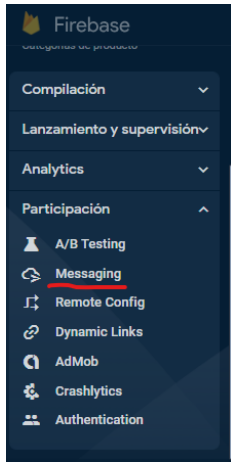
```
//...

dependencies {
   //...
   compile 'com.google.firebase:firebase-messaging:9.8.0'
}

apply plugin: 'com.google.gms.google-services'
```

Después de modificar cada fichero de Gradle recordad que es necesario sincronizar los cambios con el proyecto. Para ello, podemos utilizar la opción «*Sync Now*» que aparece en la zona superior derecha del editor cuando modificamos cualquier fichero de configuración de Gradle.

Bueno, vale, no hemos terminado aún, quedan muchas cosas por explicar, pero lo sorprendente del asunto es que sólo con lo que acabamos de hacer, y sin escribir aún ni una sola línea de código, ya seríamos capaces de enviar notificaciones sencillas a nuestra aplicación. ¿No os lo creéis? Veamos cómo hacerlo.

Quienes recuerden la anterior versión de los servicios de notificaciones push (Google Cloud Messaging o GCM) sabrán que para el envío de mensajes debíamos construir por nosotros mismos una aplicación web que corriera sobre un servidor de aplicaciones independiente al de la mensajería de Google, y que contuviera la lógica necesaria para dicha tarea. Aunque con la mudanza de estos servicios a la plataforma de Firebase esto sigue siendo válido, y de hecho será lo normal (más adelante lo veremos), Firebase también nos ofrece en su propia consola una utilidad que nos permite el envío de mensajes sin necesidad de implementar ninguna aplicación adicional. Podemos acceder a dicha utilidad entrando en la consola y pulsando la opción de «Notifications» del menú lateral.

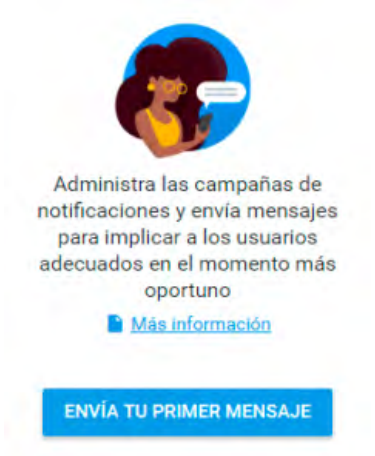
Figura 174. Paso 1: Problema 4.4.1.1 Firebase CloudMessaging



Fuente: Propia

Si es la primera vez que accedemos a esta opción, nos aparecerá una breve descripción de la utilidad de esta herramienta y un botón para comenzar a configurar y enviar notificaciones.

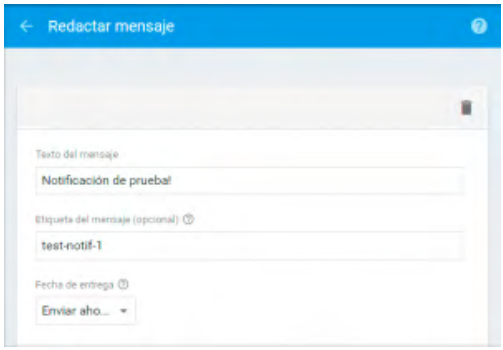
Figura 175. Paso 2: Problema 4.4.1.1 Firebase CloudMessaging



Fuente: Propia

Si pulsamos el botón de «ENVIA TU PRIMER MENSAJE» acccderemos al formulario de envío, donde se nos solicitarán una serie de datos. En primer lugar introduciremos el texto de la notificación y un identificador interno que nos podrá servir posteriormente para diferencias los distintos mensajes que enviemos. También podremos elegir entre enviar el mensaje inmediatamente o programarlo para algún otro momento.

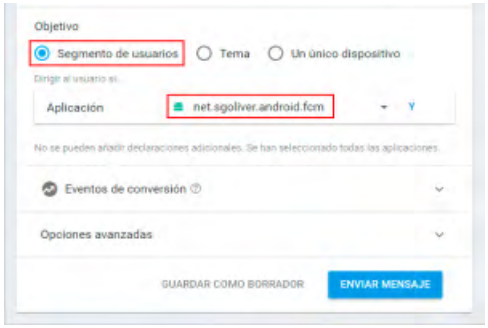
Figura 176. Paso 3: Problema 4.4.1.1 Firebase CloudMessaging



Fuente: Propia

La siguiente opción que podremos elegir será a quién enviar el mensaje, pudiendo seleccionar entre enviarlo a un *usuario determinado*, a un *tema*, o a un *segmento de usuarios*. Por ahora, dado que aún no sabemos lo que es un tema ni sabemos identificar dispositivos concretos, dejaremos seleccionada la opción de «Segmento de usuarios», más adelante hablaremos de las otras dos opciones. Y dado que hemos seleccionado la opción de segmento, tendremos que indicar qué condiciones deberán cumplirse para determinar los usuarios que deben recibir el mensaje. Para este primer ejemplo no nos complicaremos y seleccionaremos todos los usuarios cuya aplicación se corresponda con nuestra aplicación.

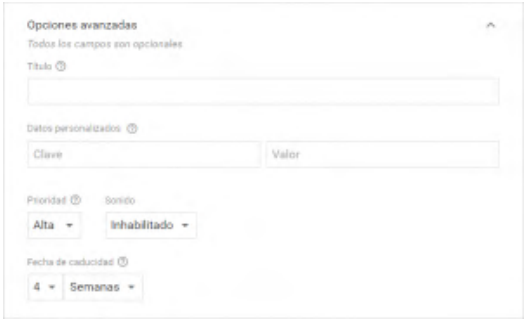
Figura 177. Paso 4: Problema 4.4.1.1 Firebase CloudMessaging



Fuente: Propia

El resto de opciones del formulario nos permiten dar un título a la notificación, añadir datos personalizados al mensaje, cambiar la prioridad, habilitar el sonido, o asignarle una fecha de caducidad, pero por ahora las dejaremos con sus valores por defecto.

Figura 178. Paso 5: Problema 4.4.1.1 Firebase CloudMessaging



Fuente: Propia

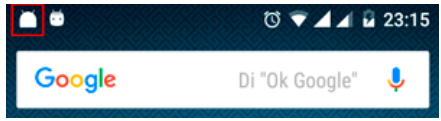
Pues bien, configurado el mensaje ya solo nos quedaría pulsar «ENVIAR MENSAJE», pero antes debemos tener instalada nuestra aplicación. Mi recomendación es este caso es que la apli-

cación se pruebe en un dispositivo real, y no en el emulador, ya que de este modo deben aparecer menos problemas. Ejecutaremos la aplicación desde Android Studio y, esto es importante, NO debemos dejarla ejecutándose en primer plano. Para ello podemos pulsar el botón Atrás o Home del dispositivo para que simplemente no aparezca en pantalla. Más adelante explicaremos por qué es esto necesario (por el momento).

Ahora sí podemos enviar el mensaje pulsando el botón «ENVIAR MENSAJE», y tras una pantalla de confirmación/revisión del mensaje éste se enviará a todos los dispositivos con nuestra aplicación instalada.

Si todo ha ido bien, debería llegar casi de inmediato a nuestro dispositivo la notificación enviada:

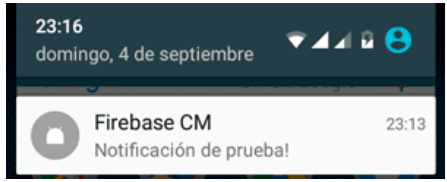
Figura 179. Paso 6: Problema 4.4.1.1 Firebase CloudMessaging



Fuente: Propia

Si desplegamos la bandeja del sistema podremos ver el mensaje que hemos configurado (como título de la notificación aparece por defecto el nombre de la aplicación, en mi caso «*Firebase CM*«):

Figura 180. Paso 7: Problema 4.4.1.1 Firebase CloudMessaging



Fuente: Propia

Finalmente, si pulsamos sobre la notificación, el comportamiento por defecto del sistema será abrir nuestra aplicación de forma que aparezca en primer plano.

¿No os parece fantástico todo esto teniendo en cuenta que aún no hemos escrito nada de código? Sin duda lo es, pero tampoco debemos engañarnos, hemos encontrado varias limitaciones por el camino, entre las más destacables:

- No hemos podido enviar mensajes a usuarios concretos. Ya sabemos que la opción existe en la consola, pero ¿como identificamos a cada usuario?
- Si la aplicación Android hubiera estado en primer plano en el momento de enviar el mensaje la notificación no habría aparecido en la barra de estado ni en la bandeja del sistema.
- Si hubiéramos añadido datos personalizados al mensaje éstos no habrían aparecido por ningún sitio, tan sólo veríamos el título y el texto de la notificación.
- Para el envío de mensajes hemos utilizado la consola de Firebase, que aunque útil y práctica, no deja de ser una utilidad independiente y de terceros no integrada con nuestro sistema.

En los próximos artículos iremos viendo poco a poco cómo resolver todos estos problemas o limitaciones.

 4.4.2 Notificaciones Push en Android: Firebase Cloud Messaging (II)

En este nuevo artículo sobre notificaciones push con Firebase Cloud Messaging (FCM) vamos a intentar dar solución a algunos de los problemas o limitaciones que encontramos al final

del capítulo anterior.

El primero de ellos estaba relacionado con la posibilidad de enviar mensajes a usuarios concretos. En el primer artículo ya vimos cómo enviar un mensaje desde la consola de Firebase, pero tuvimos que hacerlo de forma masiva, es decir, dirigido a TODOS los usuarios de nuestra aplicación, ya que no sabíamos como identificar a usuarios concretos.

Pues bien, esta identificación de usuarios Firebase la resuelve mediante el uso de *tokens de registro*, o *InstanceID Token* como lo encontraréis en la documentación en inglés. Este concepto no será nuevo para quien ya conociera la versión anterior de los servicios de mensajería Google (Google Cloud Messaging o GCM), ya que es algo que se ha mantenido desde entonces. Por explicarlo de forma rápida y sencilla, un token de registro no es más que un dato que identifica de forma única a una aplicación determinada instalada en un dispositivo determinado.

El token de registro se asigna a nuestra aplicación en el momento de su primera conexión con los servicios de mensajería, y en condiciones normales se mantiene invariable en el tiempo. Sin embargo, en determinadas circunstancias este dato puede cambiar durante la vida de la aplicación (por ejemplo cuando se reinstala, cuando el usuario borra los datos de la aplicación, por refrescos de seguridad…), por lo que debemos estar preparados para detectar estos posibles cambios.

4.4.2.1 Problema:

Para conocer el token de registro que tenemos asignado podremos acceder en cualquier momento a la instancia de nuestra aplicación mediante getInstance() y obtener el valor del token llamando a getToken(). En mi aplicación de ejemplo he añadido

un botón a la interfaz para hacer esto:

```
btnToken = (Button)findViewById(R.id.btnToken);
btnToken.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      //Se obtiene el token actualizado
         String refreshedToken = FirebaseInstanceId.getInstance().get-
Token();

      Log.d(LOGTAG, "Token actualizado: " + refreshedToken);
   }
});
```

Por otra parte, para ser notificados cuando se produzca alguna actualización del token, tendremos que definir un servicio que extienda de la clase base FirebaseInstanceIdService. En principio tan sólo tendremos que sobrescribir su método onTokenRefresh(), que se llamará cada vez que el token se actualice, incluida su primera asignación. Para obtener el token en este caso actuaremos exactamente de la misma forma que hemos hecho en el botón. Veamos cómo quedaría el código completo del servicio:

```
package net.sgoliver.android.fcm;

import android.util.Log;
import com.google.firebase.iid.FirebaseInstanceId;
import com.google.firebase.iid.FirebaseInstanceIdService;

public class MyFirebaseInstanceIDService extends FirebaseInstanceId-
Service {

   private static final String LOGTAG = "android-fcm";
```

```
  @Override
  public void onTokenRefresh() {
    //Se obtiene el token actualizado
      String refreshedToken = FirebaseInstanceId.getInstance().get-
Token();

    Log.d(LOGTAG, "Token actualizado: " + refreshedToken);
  }
}
```

Por motivos didácticos, en mi caso de ejemplo muestro el token recibido en un mensaje de log, pero por supuesto esto no es necesario ni recomendable.

Tendremos que añadir también la referencia a este servicio en nuestro fichero *AndroidManifest.xml*:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/an-
droid"
  package="net.sgoliver.android.fcm">

  <application ...>

    <activity android:name=".MainActivity">
      ...
    </activity>

    <service
      android:name=".MyFirebaseInstanceIDService">
      <intent-filter>
        <action android:name="com.google.firebase.INSTANCE_ID_
EVENT"/>
      </intent-filter>
    </service>
```

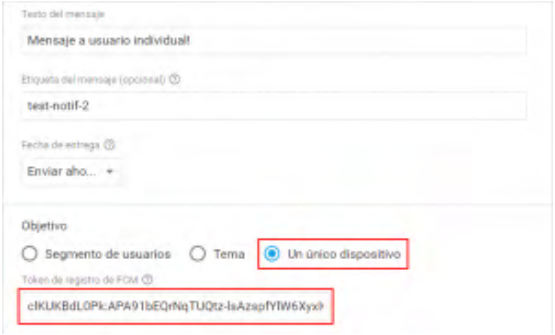
```
  </application>

</manifest>
```

Hecho esto, si volvemos a ejecutar la aplicación, pulsamos el botón que acabamos de añadir, y nos dirigimos al log de Android podremos ver el token de registro asignado a nuestra instancia de aplicación.

Perfecto, ahora ya sí tenemos una forma de identificar a un usuario concreto de nuestra aplicación (o para ser más exactos a una instancia de nuestra aplicación, ya que el usuario puede ejecutar la aplicación desde varios dispositivos, que tendrían tokens diferentes).

Para probarlo, volveremos a la consola de Firebase, a la sección de envío de notificaciones, crearemos un nuevo mensaje como ya hicimos en el artículo anterior, pero seleccionaremos en esta ocasión como objetivo la opción de «Un único dispositivo». Al hacer aparecerá un nuevo campo del formulario donde podremos indicar el token de registro del dispositivo al que queremos enviar el mensaje. Copiamos el token que hemos visualizado en el log y enviamos el mensaje.

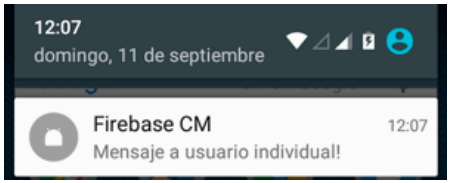
Figura 181. Paso 2: Problema 4.4.2.1 Token – Messaging



Fuente: Propia

Si todo ha ido bien, deberá aparecer la notificación en el dispositivo donde estemos probando (recordad que la aplicación debe estar en segundo plano en el momento de recibirse la notificación).

Figura 182. Paso 3: Problema 4.4.2.1 Token – Messaging



Fuente: Propia

Acabamos de resolver el primero de los problemas que encontramos en el primer artículo de la serie sobre Firebase Cloud Messaging. En los próximos artículos seguiremos mejorando la aplicación para atajar el resto de limitaciones.

4.4.3 Notificaciones Push en Android: Firebase Cloud Messaging (III)

En el artículo anterior de la serie resolvimos el problema de la identificación de usuarios/dispositivos individuales a la hora de enviar mensajes a través de Firebase Cloud Messaging. En este nuevo artículo nos centraremos en explicar por qué las notificaciones que hemos enviado hasta ahora tan sólo aparecen en el dispositivo cuando nuestra aplicación se encuentra en segundo plano, y veremos lo sencillo que es resolverlo.

En el primer artículo de la serie vimos que cuando se recibía un mensaje sobre una aplicación que se encontraba en segundo plano el sistema generaba y mostraba automáticamente la notificación en el dispositivo sin necesidad de hacer nada por nuestra parte, incluso asignaba una acción por defecto a la notificación

de forma que al pulsarla se iniciaba nuestra actividad principal.

Sin embargo, cuando el mensaje va dirigido a la aplicación que está en primer plano deberá ser la propia aplicación la que se encargue de realizar la gestión completa de dicho mensaje, es decir, el sistema no hará nada por nosotros, ni siquiera mostrar la notificación en la barra de estado o en la bandeja del sistema.

4.4.3.1 Problema

Solucionar esto es bastante sencillo, aunque requiere añadir un nuevo componente a la aplicación. Concretamente tendremos que añadir un nuevo servicio, esta vez extendiendo a FirebaseMessagingService. En esta clase tan sólo tendremos que sobrescribir un método llamado onMessageReceived(), que se llamará automáticamente cada vez que la aplicación reciba un mensaje de Firebase Cloud Messaging.

El método onMessageReceived() recibe como parámetro un objeto RemoteMessage que encapsula la información del mensaje recibido. Así, para acceder por ejemplo al título o al texto de la notificación podremos llamar a los métodos correspondientes, getTitle() y getBody() en este caso, sobre dicho objeto. Adicionalmente, si quisiéramos mostrar la notificación recibida en la barra de estado y la bandeja del sistema tendríamos que hacerlo por nosotros mismos, haciendo uso de NotificationCompat. Si necesitas más información sobre la generación de notificaciones en la barra de estado de Android puedes consultar el capítulo del curso dedicado a este tema.

Se muestra a continuación el código completo del servicio de recepción de mensajes. En mi caso de ejemplo me limito a mostrar los datos recibidos en el log del sistema y a generar una notificación en la barra de estado con dichos datos:

```
public class MyFirebaseMessagingService extends FirebaseMessaging-
Service {

   private static final String LOGTAG = "android-fcm";

   @Override
   public void onMessageReceived(RemoteMessage remoteMessage) {

      if (remoteMessage.getNotification() != null) {

         String titulo = remoteMessage.getNotification().getTitle();
         String texto = remoteMessage.getNotification().getBody();

         Log.d(LOGTAG, "NOTIFICACION RECIBIDA");
         Log.d(LOGTAG, "Título: " + titulo);
         Log.d(LOGTAG, "Texto: " + texto);

         //Opcional: mostramos la notificación en la barra de estado
         showNotification(titulo, texto);
      }
   }

   private void showNotification(String title, String text) {

      NotificationCompat.Builder notificationBuilder =
         new NotificationCompat.Builder(this)
            .setSmallIcon(android.R.drawable.stat_sys_warning)
            .setContentTitle(title)
            .setContentText(text);

      NotificationManager notificationManager =
           (NotificationManager) getSystemService(Context.NOTIFICA-
TION_SERVICE);

      notificationManager.notify(0, notificationBuilder.build());
```

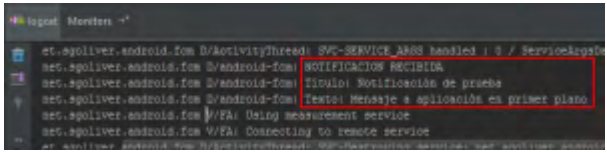
```
    }
}
```

No hay que olvidar nunca incluir también la referencia al nuevo servicio en el fichero *AndroidManifest.xml* de nuestro proyecto de Android Studio:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.sgoliver.android.fcm">

    <application ...>

        ...

        <service
            android:name=".MyFirebaseMessagingService">
            <intent-filter>
                <action android:name="com.google.firebase.MESSAGING_EVENT"/>
            </intent-filter>
        </service>

    </application>

</manifest>
```

Hecho esto, ya podríamos ejecutar de nuevo la aplicación y enviar un mensaje desde la consola de Firebase, esta vez sin necesidad de mandar la aplicación a segundo plano antes del envío. Tras enviar el mensaje podremos revisar el log de Android para ver los datos recibidos:

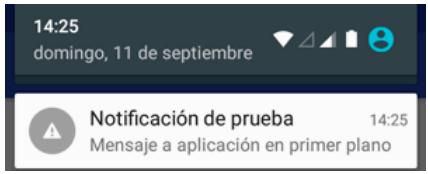
Figura 183. Paso 1: Problema 4.4.3.1 FirebaseMessagingService



Fuente: Propia

Adicionalmente, también debería haber aparecido la notificación en la barra de estado del dispositivo:

Figura 184. Paso 2: Problema 4.4.3.1 FirebaseMessagingService



Fuente: Propia

Con esto, acabamos de solucionar el segundo de los problemas que encontramos en nuestro capítulo inicial. Seguiremos mejorando la aplicación en próximos capítulos.

4.4.4 Notificaciones Push en Android: Firebase Cloud Messaging (IV)

En los artículos anteriores ya hemos resuelto dos de los problemas planteados en el primer capítulo. Por un lado hemos aprendido a identificar usuarios individuales para poder dirigir mensajes a dispositivos concretos. Por otro, ya tenemos preparada nuestra aplicación para recibir notificaciones tanto cuando no se encuentra visible como cuando es la aplicación en primer plano. En esta cuarta entrega de la serie vamos a centrarnos en el envío y recepción de datos adicionales dentro de los mensajes de Firebase Cloud Messaging.

Hasta ahora, he estado utilizando los términos «notificación» y «mensaje» casi indistintamente, pero para ser exactos debería haber utilizado siempre el concepto de «mensaje de notificación» o «mensaje con carga de notificación» (en inglés, «*notification payload*«).

Firebase Cloud Messaging distingue entre dos tipos de mensajes:

1. Mensajes de notificación, o *con carga de notificación*.
2. Mensajes de datos, o *con carga de datos*.

Los **mensajes de notificación**, que son los que hemos estudiado hasta ahora, son los más limitados, aunque suficientes en multitud de ocasiones. Pueden contener hasta 2 Kb de información, pero distribuida en un conjunto de claves o campos predeterminados. Estos campos predeterminados son los que ya nos deben sonar de artículos anteriores: título, texto, prioridad, sonido, … (existen más campos que los que aparecen en la sección de Notificaciones de la consola de Firebase, más adelante veremos algunos de ellos). El utilizar únicamente campos predefinidos permite al sistema gestionar automáticamente los mensajes en determinadas circunstancias, por ejemplo generando las notificaciones por nosotros cuando la aplicación se encuentra en segundo plano.

Por su parte, los **mensajes de datos** permiten enviar conjuntos de clave-valor totalmente personalizados hasta un límite de 4 Kb de información. No tendremos que limitarnos a los campos disponibles, sino que podremos definir los nuestros propios según las necesidades de nuestra aplicación. Sin embargo, al no basarse en campos predefinidos, la gestión completa de estos mensajes dependerá exclusivamente de nuestra aplicación, es decir, el sistema no podrá hacer nada por nosotros como en el caso de las notificaciones.

En cualquier caso, es importante entender que estos dos tipos de mensajes no son excluyentes, es decir, se pueden enviar mensajes que contengan ambos tipos de carga, por un lado la información asociada a la notificación y por otro los datos personalizados. En breve veremos cómo y dónde podremos gestionar los mensajes en cada caso.

4.4.4.1 Problema:

 Como hemos indicado, los mensajes de datos «puros» (es decir, sin carga de notificación) deben ser gestionados en su totalidad por nuestra aplicación. Sin embargo, el lugar donde podremos gestionar estos datos ya lo conocemos, será el mismo servicio de recepción de mensajes (extendido de FirebaseMessagingService) que ya utilizamos en el artículo anterior. En el caso de mensajes de datos puros éste será el único lugar donde se podrá gestionar la información recibida, tanto si la aplicación está visible como en segundo plano.

Dentro del método onMessageReceived() de este servicio podremos acceder a los datos incluidos en el mensaje de forma análoga a como accedíamos a los datos de la carga de notificación, con la salvedad de que usaremos el método getData() en vez de getNotification().

Imaginemos que nuestros mensajes de datos van a incluir por ejemplo dos campos (claves) personalizados para informar del estado de conexión de un usuario. Llamaremos a estos campos «usuario» y «estado». Dentro del método onMessageReceived() podríamos obtener los valores de estas claves obteniendo los datos del mensaje con getData() y llamando posteriormente al método get() pasando como parámetro el nombre de cada clave:

```
@Override
```

```
public void onMessageReceived(RemoteMessage remoteMessage) {

   if (remoteMessage.getNotification() != null) {
      String titulo = remoteMessage.getNotification().getTitle();
      String texto = remoteMessage.getNotification().getBody();

      Log.d(LOGTAG, "NOTIFICACION RECIBIDA");
      Log.d(LOGTAG, "Título: " + titulo);
      Log.d(LOGTAG, "Texto: " + texto);

      //Opcional: mostramos la notificación en la barra de estado
      showNotification(titulo, texto);
   }

   if(remoteMessage.getData() != null) {
      Log.d(LOGTAG, "DATOS RECIBIDOS");
        Log.d(LOGTAG, "Usuario: " + remoteMessage.getData().get("u-
suario"));
       Log.d(LOGTAG, "Estado: " + remoteMessage.getData().get("esta-
do"));
   }
}
```

Como podéis ver hemos dejado también la parte de gestión de notificaciones que ya implementamos en el artículo pasado porque como hemos indicado es posible recibir mensajes que contengan ambos tipos de información (notificación + datos personalizados).

Para probar si todo funciona correctamente vamos a utilizar una vez más la consola de Firebase, pero nos encontraremos con un pequeño problema. Si nos fijamos bien, el campo «Texto del mensaje» es obligatorio en el formulario de envío de mensajes. Este campo es propio de los mensajes de notificación por lo que

acabamos de descubrir que la consola de Firebase no soporta el envío de mensajes de datos puros, sino tan sólo de mensajes de notificación o de mensajes «mixtos», es decir, con carga simultánea de notificación y de datos. Este problema viene a agravar la cuarta de las limitaciones que encontramos en el primer artículo de la serie sobre Firebase Cloud Messaging, es decir, la Consola de Firebase puede resultar muy práctica para el envío de mensajes en multitud de ocasiones, pero puede no adaptarse a nuestras necesidades en muchas otras, por lo que es necesario contar con otra alternativa (lo que veremos en un próximo artículo).

Por ahora nos conformaremos con enviar mensajes que contengan tanto notificación como datos desde la consola. Para añadir datos personalizados a un mensaje basta con indicar los pares de clave y valor en las opciones avanzadas del formulario de envío. Por seguir con nuestro ejemplo podríamos añadir las claves «usuario» y «estado» con dos valores de ejemplo:

Figura 185. Paso 1: Problema 4.4.4.1 onMessageReceived()

Fuente: Propia

Si ejecutamos la aplicación en este momento, dejándola en primer plano, y enviamos el mensaje con los datos adicionales tal como se muestra en la imagen anterior, podremos ver en el log del sistema como se reciben correctamente los datos en el método onMessageReceived()

Podemos observar también como justo encima de los datos recibidos aparece también la información asociada a la notificación, lo que nos confirma que la Consola de Firebase no es capaz de enviar mensajes de datos sin carga de notificación (al menos en el momento de escribir este tutorial).

Aparentemente ya estaría todo resuelto con los mensajes de datos, pero aún nos queda un problema más por solucionar. Vamos a volver a enviar otro mensaje con datos personalizados desde la consola, pero esta vez con nuestra aplicación en segundo plano. ¿Qué ha ocurrido? El sistema ha generado automáticamente una notificación en la bandeja del sistema (por recibirse *carga de notificación* pero no encontrarse la aplicación en primer plano), pero sin embargo no se ha ejecutado el método onMessageReceived(), por lo que no hemos podido obtener los datos personalizados del mensaje.

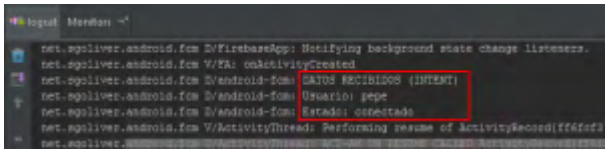
Esto no es ningún error, es el funcionamiento esperado del sistema en estos casos. En el caso de mensajes mixtos (de notificación + datos), cuando la aplicación se encuentra en segundo plano el sistema gestionará por nosotros la información de notificación mostrando automáticamente el mensaje en la bandeja del sistema, pero condicionará la entrega a nuestra aplicación de los datos personalizados a que el usuario pulse sobre la notificación mostrada. Si el usuario pulsa sobre la notificación los datos se recibirán como *extras* del intent que iniciará nuestra aplicación. Podremos obtener estos datos por ejemplo en el método onCreate() llamando a getIntent() y sobre éste a getExtras().

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
```

```
    if (getIntent().getExtras() != null) {
        Log.d(LOGTAG, "DATOS RECIBIDOS (INTENT)");
       Log.d(LOGTAG, "Usuario: " + getIntent().getExtras().getString("u-
suario"));
        Log.d(LOGTAG, "Estado: " + getIntent().getExtras().getString("es-
tado"));
    }

    //...
}
```

Si volvemos a hacer nuevamente la prueba anterior, enviando el mensaje con datos cuando nuestra aplicación esté en segundo plano, ahora sí deberíamos ver en el log los datos recibidos al pulsar sobre la notificación generada automáticamente por el sistema.

Figura 186. Paso 3: Problema 4.4.4.1 onMessageReceived()



Fuente: Propia

Es importante entender que si hubiéramos podido enviar mensajes de datos puros desde la consola no habría sido necesario hacer esto último, ya que los mensajes que únicamente incluyen carga de datos siempre se reciben en el método onMessageReceived(), independientemente de que la aplicación esté visible o en segundo plano.

A modo de resumen, la siguiente tabla muestra cuándo y donde se recibe la información de un mensaje dependiendo de su tipo y la situación actual de nuestra aplicación.

Tabla 3. Modo para recibir la información con Firebase Cloud-Messaging

Situación Aplicación	Notificación	Datos	Notificación + Datos
En primer plano	onMessageReceived()	onMessageReceived()	onMessageReceived()
En segundo plano	Bandeja del sistema	onMessageReceived()	Notificación: Bandeja del sistema Datos: Extras del intent

Fuente: Documentación de Firebase obtenido de https://firebase.google.com/docs/cloud-messaging/?hl=es-419

Con esto finalizaríamos el apartado dedicado a los mensajes de datos y tan sólo nos quedaría por ahora buscar una alternativa a la consola de Firebase, que sea más completa, versátil e integrable con el resto de nuestro sistema.

4.4.5 Evaluación 6

1. ¿Cuál de las siguientes funciones clave de Cloud Storega para Firebase corresponde la siguiente descripción: está diseñado para escalar a exabytes si tu app se vuelve viral? Pasa fácilmente de la fase prototipo a la de producción con la misma infraestructura que respalda a Spotify y Google Fotos.
2. ¿Cuál es la ruta de implementación de Cloud Storage para Firebase?
3. ¿Cuál es la dependencia para la biblioteca de Android de Cloud Storage para Firebase?
4. ¿Cuál método es para descargar un archivo a Cloud Storage?

5. En Cloud Storage para Firebase cual método me ayuda a actualizar metadatos de archivos
6. Los códigos de error se definen en la clase StorageException como constantes de número entero. ¿Cuál error resultaría si no se configuró ningún depósito para Cloud Storage?
7. ¿Cuál servicio de Firebase permite el envío de mensajes entre un servidor de aplicaciones en la nube y un dispositivo Android, iOS o Web?
8. El token de registro se asigna a nuestra aplicación en el momento de su primera conexión con los servicios de mensajería Firebase Cloud Messaging, y en condiciones normales se mantiene invariable en el tiempo. ¿Cuál es el método para conocer el token de registro que tenemos asignado?
9. ¿Cuál es el método que se llamará automáticamente cada vez que la aplicación reciba un mensaje de Firebase Cloud Messaging?
10. Firebase Cloud Messaging distingue entre dos tipos de mensajes: de notificación y de datos. ¿Cuál es el espacio límite que puede contener los mensajes de datos?

Referencias

Bex Ediciones. (2020). *UX Diseño para aplicaciones moviles: Cuaderno para diseñar interfaces digitales.* Independently published.

Carrasco, J. (2020). *Desarrollo de aplicaciones móviles en Kotlin: Introducción a la programación móvil.* Independently published.

Fleitas, F. (2016). *Hacia la quinta generación en tecnologías móviles y sus aplicaciones: La evolución de tecnologías móviles y sus aplicaciones*. Académica Española

Gómez, S. (2018, 15 de noviembre). Curso de programación android. *Sgoliver.net* https://www.sgoliver.net/blog/curso-de-programacion-android/

Lane, J. (2021, 01 de octubre). Aplicación, alternativa a app. *Fundéu-Rae.* https://www.fundeu.es/recomendacion/aplicacion-alternativa-a-app/

Lugo, A. (s.f). ¿Qué es el desarrollo de aplicaciones móviles? *Invid*. https://invidgroup.com/es/que-es-el-desarrollo-de-aplicaciones-moviles/

Lujani, F. (2010, 14 de julio). Andy Rubin, el creador de Android. *Maestros del Web.* http://www.maestrosdelweb.com/andy-rubin-el-creador-de-android/

Nolasco Valenzuela, J.S. (2020). *Desarrollo de aplicaciones con Android*. Ra-Ma.

Puetate, G. (2020). *Aplicaciones móviles híbridas*. Centro de publicaciones PUCE

Ramos, C. (2021). *Aprende apache cordoba desde cero en español: Cordova es una plataforma que se utiliza para crear aplicaciones móviles utilizando HTML, CSS y JS.* Independently published.

Romero, I. (2019). *Aplicaciones Hibridas–Apache Cordova: App's móviles con HTML , CSS y JS. Dirigirse a múltiples plataformas con una base de código Libre y de código abierto*. Académica Española

Santiago, R., Trabaldo, S., Kamijo, M., y Fernández, A. (2019). *Mobile learning: nuevas realidades en el aula*. Grupo Oceano.

Tutoriales Ya. (2018). *Android Ya*. https://www.tutorialesprogramacionya.com/javaya/androidya/

Venturabeat.com. (2020, 29 de junio). *Analyst: There's a great future in iPhone apps*. https://venturebeat.com/2008/06/11/analyst-theres-a-great-future-in-iphone-apps/

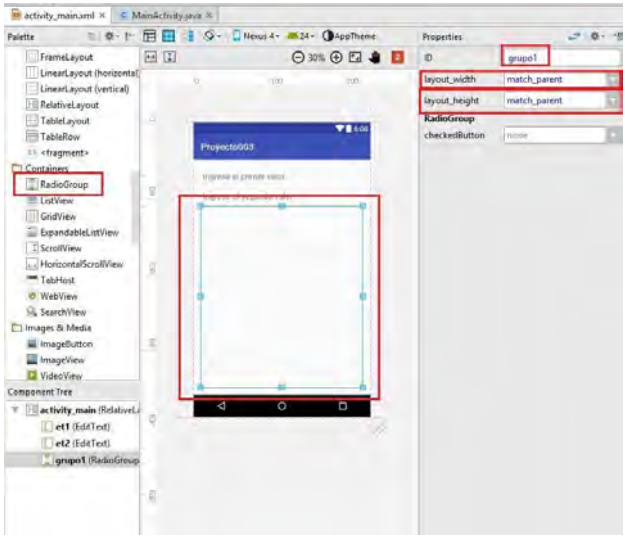
WBYourDesign. (2021). *Diseño UX para apps móviles: Plantillas para diseñar interfaces para aplicaciones móviles. Prototipa tus apps móviles.* Independently published

Anexos

Anexo 1 Solucionario Capitulo 1

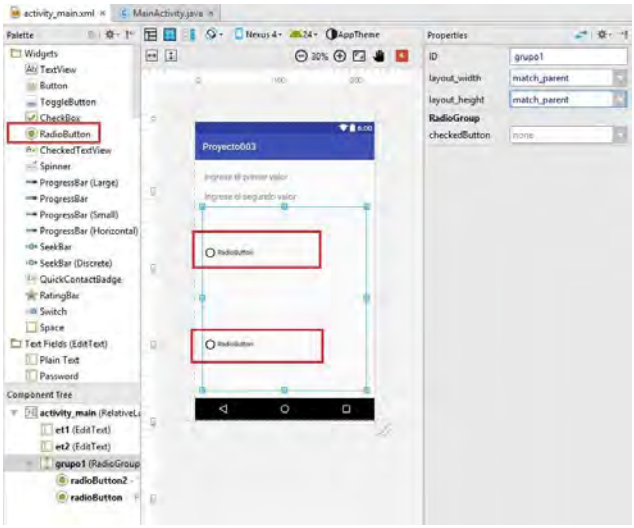
Solución problema 1.5.2.1.

El problema es similar al problema 1.5.1.1. Disponemos dos controles EditText (Number) y configuramos sus propiedades id y hint. Para disponer los controles de tipo RadioButton debemos en realidad *primero insertar* un control de tipo RadioGroup (este control se encuentra en la paleta de componentes en la pestaña Containers de la "Palette"):



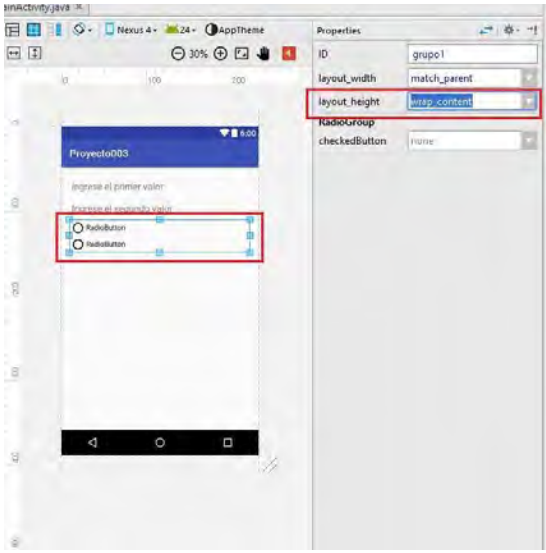
Luego de arrastrar el control RadioGroup iniciamos las propiedades layout_width y layout_height con el valor "match_parent". También definimos su id con el valor grupo1.

Ahora debemos arrastrar dos controles de la clase RadioButton de la pestaña "Widgets" dentro del RadioGroup



Nuestro problema solo requiere dos controles de tipo RadioButton.

Otra cosa muy importante es seleccionar nuevamente el control RadioGroup y cambiar la propiedad layout_height con el valor wrap_parent (para que el control RadioGroup solo ocupe el espacio de los dos controles RadioButton):



Ahora a los dos controles de tipo RadioButton definimos sus id (los llamaremos r1 y r2 respectivamente) Cambiamos sus propiedades text por los textos "sumar" y "restar".

No olvidemos también cambiar los id de los controles EditText por et1 y et2 (igual que en el problema anterior)

Por último agreguemos un botón y un TextView para mostrar el resultado Inicializamos las propiedades del botón con los valores:

id : button

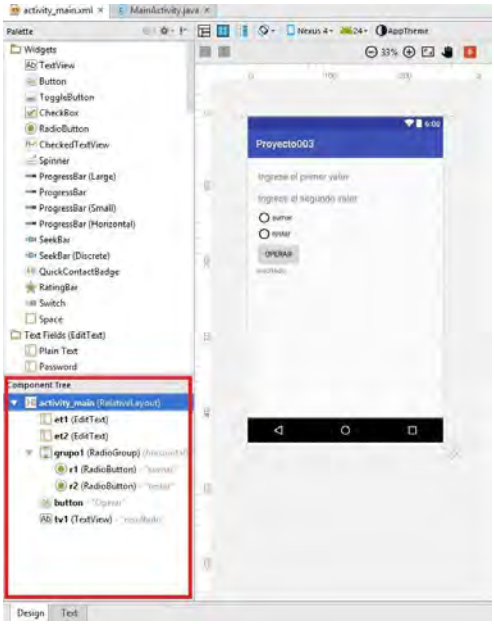
text : operar

Y el TextView con los valores:

id : tv1

text : resultado

Podemos controlar en la ventana "Component Tree" el id definido para cada control (et1, et2, grupo1, r1, r2, tv1) También podemos observar de que clase es cada control visual y el texto de la propiedad text para aquellos controles que tienen sentido su inicialización.



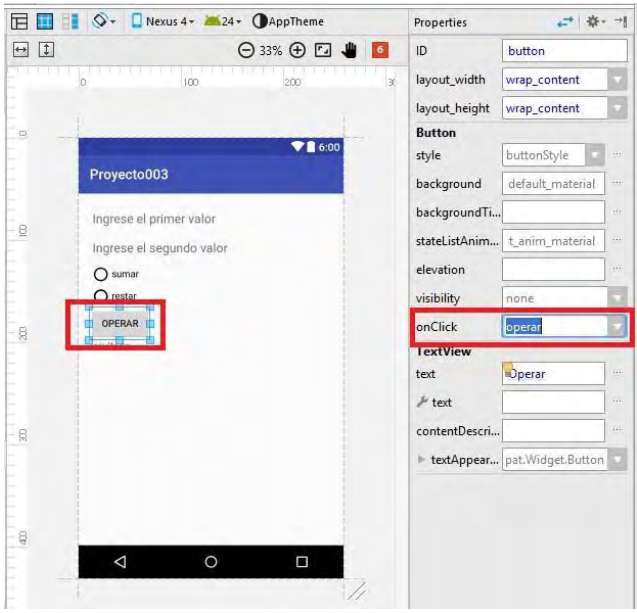
Captura del evento clic del button e identificación del RadioButton seleccionado

El código fuente de la clase MaintActivity es:

```
package com.tutorialesprogramacionya.proyecto003;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.RadioButton;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
   private EditText et1,et2;
   private TextView tv1;
   private RadioButton r1,r2;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        et1=(EditText)findViewById(R.id.et1);
        et2=(EditText)findViewById(R.id.et2);
        tv1=(TextView)findViewById(R.id.tv1);
        r1=(RadioButton)findViewById(R.id.r1);
        r2=(RadioButton)findViewById(R.id.r2);
    }

    //Este método se ejecutará cuando se presione el botón
    public void operar(View view) {
        String valor1=et1.getText().toString();
        String valor2=et2.getText().toString();
        int nro1=Integer.parseInt(valor1);
        int nro2=Integer.parseInt(valor2);
        if (r1.isChecked()==true) {
            int suma=nro1+nro2;
            String resu=String.valueOf(suma);
            tv1.setText(resu);
        } else
        if (r2.isChecked()==true) {
            int resta=nro1-nro2;
            String resu=String.valueOf(resta);
            tv1.setText(resu);
        }
    }

}
```

Primero debemos enlazar el objeto button con el método operar. Para esto similar al problema anterior seleccionamos el control button y cambiamos la propiedad onClick por el valor operar (si no hacemos esto nunca se ejecutará el método operar de la clase MainActivity):



Como podemos ver el código fuente es igual al problema anterior. Tenemos dos objetos más que debemos inicializar en el método onCreate:

```
r1=(RadioButton)findViewById(R.id.r1);
r2=(RadioButton)findViewById(R.id.r2);
```

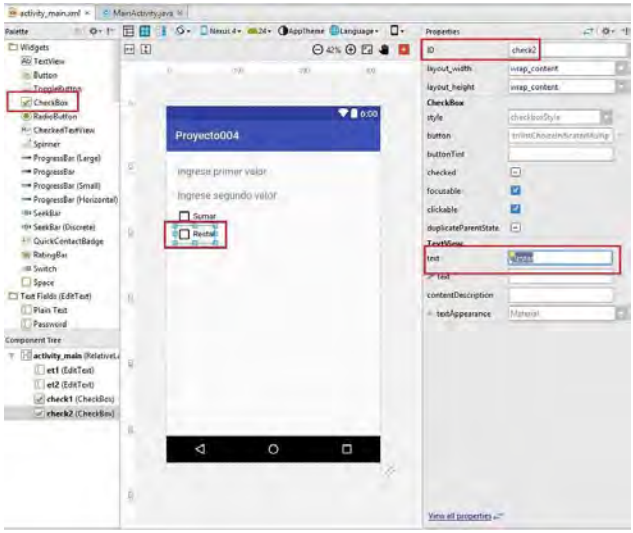
Las variables r1 y r2 son de la clase RadioButton y son necesarios en el método operar para verificar cual de los dos RadioButton están seleccionados. La clase RadioButton tiene un método llamado isChecked que retorna true si dicho elemento está seleccionado:

```
public void operar(View view) {
    String valor1=et1.getText().toString();
    String valor2=et2.getText().toString();
    int nro1=Integer.parseInt(valor1);
    int nro2=Integer.parseInt(valor2);
    if (r1.isChecked()==true) {
       int suma=nro1+nro2;
       String resu=String.valueOf(suma);
       tv1.setText(resu);
    } else
```

```
    if (r2.isChecked()==true) {
       int resta=nro1-nro2;
       String resu=String.valueOf(resta);
       tv1.setText(resu);
    }
}
```

Solución problema 1.5.3.1.

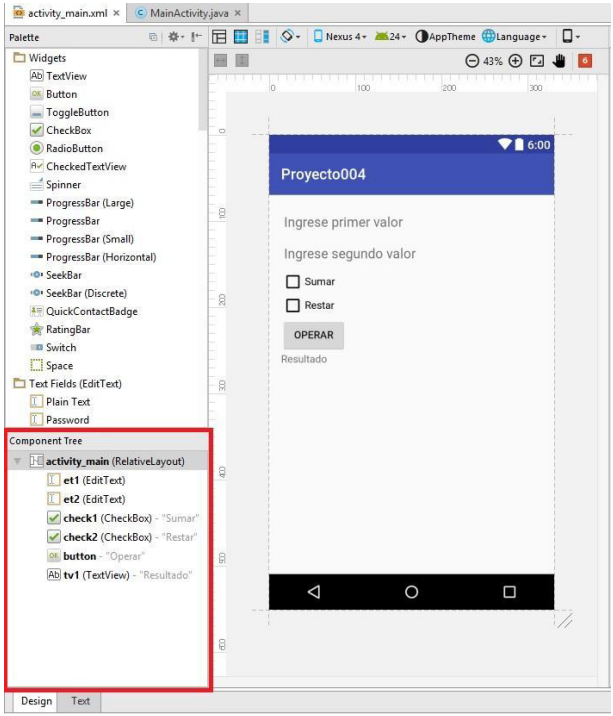
Lo nuevo en este problema es la inserción de dos objetos de la clase CheckBox que se encuentra en la pestaña "Widgets":



Debemos iniciar las propiedades "text" para mostrar un texto y la propiedad "id" para poder hacer referencia al CheckBox en el programa java.

El primer CheckBox definimos su "id" con el valor check1 y el segundo con el valor check2.

Luego la interfaz gráfica final para este problema y los nombres de los controles o componentes visuales los podemos ver en la ventana "Component Tree":



Controlar que fijamos los valores de las propiedades "id" de cada objeto: et1, et2, check1, check2 y tv1.

No olvidemos inicializar la propiedad onClick del objeto button con el valor "operar" (es el nombre del método a ejecutarse cuando se presione el botón y lo implementa la clase que hacemos)

Código fuente:

```
package com.tutorialesprogramacionya.proyecto004;


import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.TextView;
```

```
public class MainActivity extends AppCompatActivity {
    private EditText et1,et2;
    private TextView tv1;
    private CheckBox check1,check2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        et1=(EditText)findViewById(R.id.et1);
        et2=(EditText)findViewById(R.id.et2);
        tv1=(TextView)findViewById(R.id.tv1);
        check1=(CheckBox)findViewById(R.id.check1);
        check2=(CheckBox)findViewById(R.id.check2);
    }

    //Este método se ejecutará cuando se presione el botón
    public void operar(View view) {
        String valor1=et1.getText().toString();
        String valor2=et2.getText().toString();
        int nro1=Integer.parseInt(valor1);
        int nro2=Integer.parseInt(valor2);
        String resu="";
        if (check1.isChecked()==true) {
            int suma=nro1+nro2;
            resu="La suma es: "+ suma;
        }
        if (check2.isChecked()==true) {
            int resta=nro1-nro2;
            resu=resu + " La resta es: "+ resta;
        }
        tv1.setText(resu);
    }
}
```

Definimos dos objetos de la clase CheckBox como atributos de la clase:

```
private CheckBox check1,check2;
```

En el método onCreate los inicializamos con los objetos definidos en el archivo XML:

```
check1=(CheckBox)findViewById(R.id.check1);
check2=(CheckBox)findViewById(R.id.check2);
```

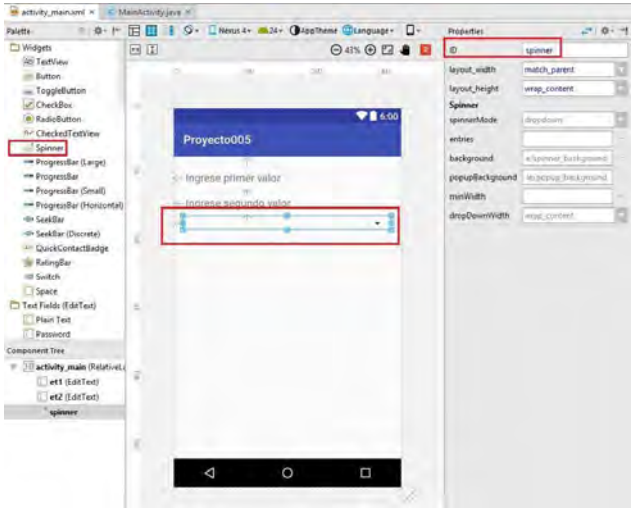
En el método operar debemos definir dos if a la misma altura ya que los dos controles de tipo CheckBox pueden estar seleccionados simultáneamente. Definimos una variable de tipo String y la inicializamos con cadena vacía para el caso en que los dos CheckBox no estén seleccionados:

```
String resu="";
 if (check1.isChecked()==true) {
    int suma=nro1+nro2;
    resu="La suma es: "+ suma;
 }
 if (check2.isChecked()==true) {
    int resta=nro1-nro2;
    resu=resu + " La resta es: "+ resta;
 }
 tv1.setText(resu);
```

Cuando ejecutamos el programa en el emulador tenemos como resultado:

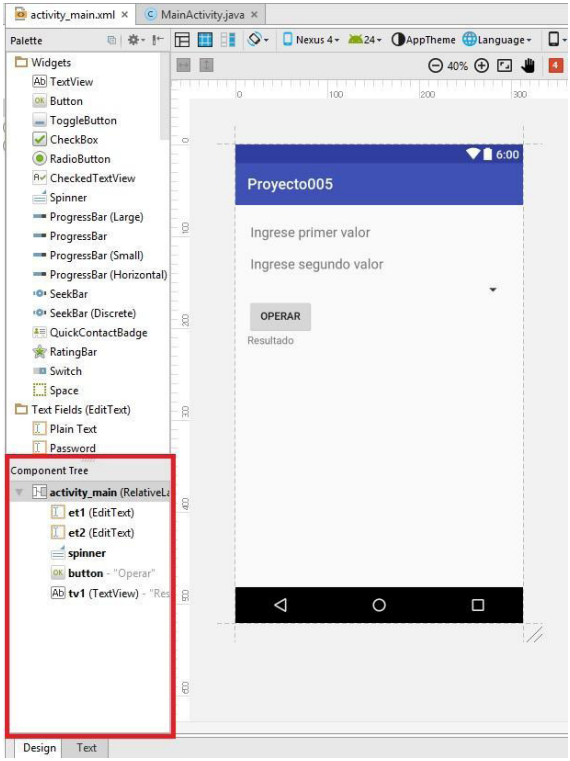
Solución problema 1.5.4.1.

Lo nuevo en este problema es la inserción de un control de tipo Spinner que se encuentra en la pestaña "Widgets" :



Dejamos la propiedad id con el valor spinner (valor por defecto que crea el Android Studio al insertar el objeto de la clase Spinner).

En la siguiente imagen en la ventana "Component Tree" del Android Studio podemos observar los objetos dispuestos en el formulario, sus id, sus textos y de que clase son cada uno:



No olvidemos inicializar la propiedad onClick del objeto button con el valor "operar" (dicho nombre es el método que debemos implementar)

Código fuente:

```
package com.tutorialesprogramacionya.proyecto005;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
```

```
    private Spinner spinner1;
    private EditText et1,et2;
    private TextView tv1;

    public MainActivity() {
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        et1=(EditText)findViewById(R.id.et1);
        et2=(EditText)findViewById(R.id.et2);
        tv1=(TextView)findViewById(R.id.tv1);

        spinner1 = (Spinner) findViewById(R.id.spinner);
        String []opciones={"sumar","restar","multiplicar","dividir"};
            ArrayAdapter <String>adapter = new ArrayAdapter<String>(this,an-
droid.R.layout.simple_spinner_item, opciones);
        spinner1.setAdapter(adapter);
    }

    //Este método se ejecutará cuando se presione el botón
    public void operar(View view) {
        String valor1=et1.getText().toString();
        String valor2=et2.getText().toString();
        int nro1=Integer.parseInt(valor1);
        int nro2=Integer.parseInt(valor2);
        String selec=spinner1.getSelectedItem().toString();
        if (selec.equals("sumar")) {
            int suma=nro1+nro2;
            String resu=String.valueOf(suma);
            tv1.setText(resu);
        } else
```

```
        if (selec.equals("restar")) {
            int resta=nro1-nro2;
            String resu=String.valueOf(resta);
            tv1.setText(resu);
        }
        else
        if (selec.equals("multiplicar")) {
            int multi=nro1*nro2;
            String resu=String.valueOf(multi);
            tv1.setText(resu);
        }
        else
        if (selec.equals("dividir")) {
            int divi=nro1/nro2;
            String resu=String.valueOf(divi);
            tv1.setText(resu);
        }
    }
}
```

Definimos un objeto de la clase Spinner:

```
private Spinner spinner1;
```

En el método onCreate obtenemos la referencia al control visual declarado en el archivo XML (Veamos que no es obligatorio que el nombre de nuestro objeto llamado spinner1 sea igual al id definido en la interfaz visual llamado spinner):

```
spinner1 = (Spinner) findViewById(R.id.spinner);
```

Definimos un vector con la lista de String que mostrará el Spinner:

```
String []opciones={"sumar","restar","multiplicar","dividir"};
```

Definimos y creamos un objeto de la clase ArrayAdapter:

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,android.R.layout.simple_spinner_item, opciones);
```

Al constructor le pasamos como primer parámetro la referencia de nuestro AppCompatActivity (this), el segundo parámetro indica el tipo de Spinner, pudiendo ser las constantes:

android.R.layout.simple_spinner_item

android.R.layout.simple_spinner_dropdown_item

El tercer parámetro es la referencia del vector que se mostrará.

Luego llamamos al método setAdapter de la clase Spinner pasando la referencia del objeto de la clase ArrayAdapter que acabamos de crear:

spinner1.setAdapter(adapter);

En el método operar que se ejecuta cuando presionamos el botón "OPERAR" y donde procedemos a extraer el contenido seleccionado del control Spinner:

String selec=spinner1.getSelectedItem().toString();

Luego mediante una serie de if anidados verificamos si debemos sumar, restar, multiplicar o dividir:

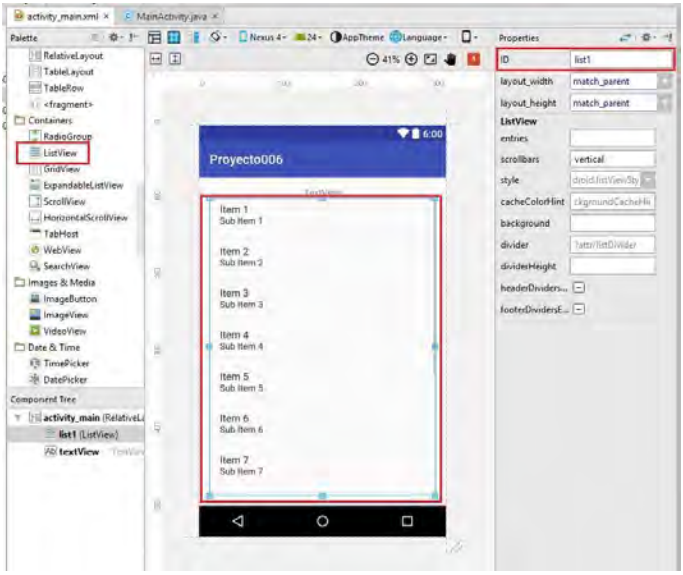
```
if (selec.equals("sumar")) {
    int suma=nro1+nro2;
    String resu=String.valueOf(suma);
    tv1.setText(resu);
} else
if (selec.equals("restar")) {
    int resta=nro1-nro2;
    String resu=String.valueOf(resta);
    tv1.setText(resu);
}
else
if (selec.equals("multiplicar")) {
    int multi=nro1*nro2;
    String resu=String.valueOf(multi);
    tv1.setText(resu);
}
else
if (selec.equals("dividir")) {
    int divi=nro1/nro2;
    String resu=String.valueOf(divi);
    tv1.setText(resu);
}
```

En el emulador tenemos como resultado de ejecutar el programa:

Si queremos que el Spinner no ocupe todo el ancho cambiamos la propiedad layout_width con el valor wrap_content.

Solución problema 1.5.5.1.

La interfaz visual a implementar es la siguiente (primero disponemos un TextView en la parte superior (cuyo id lo definimos con el valor tv1) y un ListView (y definimos su id con el valor list1)):



Código fuente:

```
package com.tutorialesprogramacionya.proyecto006;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
   private String[] paises = { "Argentina", "Chile", "Paraguay", "Bolivia",
         "Peru", "Ecuador", "Brasil", "Colombia", "Venezuela", "Uruguay" };
   private String[] habitantes = { "40000000", "17000000", "6500000",
         "10000000", "30000000", "14000000", "183000000", "44000000",
         "29000000", "3500000" };
   private TextView tv1;
```

```
  private ListView lv1;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    tv1=(TextView)findViewById(R.id.tv1);
    lv1 =(ListView)findViewById(R.id.list1);
        ArrayAdapter <String>adapter = new ArrayAdapter<String>(this,an-
droid.R.layout.simple_list_item_1, paises);
    lv1.setAdapter(adapter);

    lv1.setOnItemClickListener(new AdapterView.OnItemClickListener() {
      @Override
     public void onItemClick(AdapterView adapterView, View view, int i, long l) {
        tv1.setText("Población de "+ lv1.getItemAtPosition(i) + " es "+ habitan-
tes[i]);
      }
    });
  }
}
```

Primero definimos dos vectores paralelos donde almacenamos en uno los nombres de países y en el otro almacenamos la cantidad de habitantes de dichos países:

```
private String[] paises={"Argentina","Chile","Paraguay","Bolivia","Pe-
ru",
"Ecuador","Brasil","Colombia","Venezuela","Uruguay"};
private String[] habitan-
tes={"40000000","17000000","6500000","10000000","30000000",
"14000000","183000000","44000000","29000000","3500000"};
```

Definimos un objeto de tipo TextView y otro de tipo ListView donde almacenaremos las referencias a los objetos que definimos en el archivo XML:

```
private TextView tv1;
```

private ListView lv1;

En el método onCreate obtenemos la referencia a los dos objetos:

tv1=(TextView)findViewById(R.id.tv1);

lv1 =(ListView)findViewById(R.id.list1);

Creamos un objeto de la clase ArrayAdapter de forma similar a como lo hicimos cuando vimos la clase Spinner:

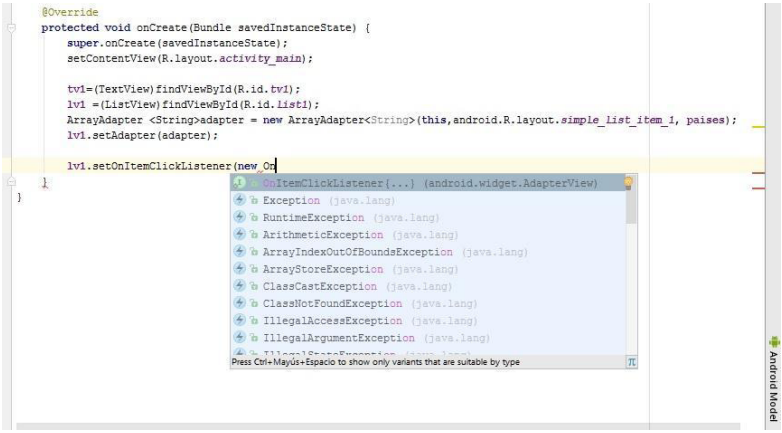
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1, paises);

lv1.setAdapter(adapter);

Llamamos al método setOnItemClicListener de la clase ListView y le pasamos como parámetro una clase anónima que implementa la interfaz OnItemClickListener (dicha interfaz define el método onItemClick que se dispara cuando seleccionamos un elemento del ListView):

```
lv1.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView adapterView, View view, int i, long l) {
        tv1.setText("Población de "+ lv1.getItemAtPosition(i) + " es "+ habitantes[i]);
    }
});
```

Para codificar esta clase anónima podemos utilizar la facilidad que nos brinda el Android Studio para generarla automáticamente, para esto tipeamos:



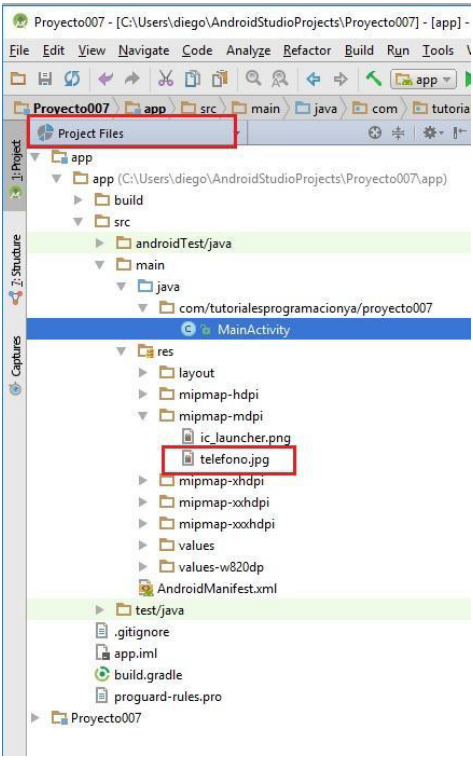
Luego presionamos la tecla "Enter" y el Android Studio automáticamente nos genera la clase anónima implementando la interfaz onItemClick.

Dentro del método onItemClick modificamos el contenido del TextView con el nombre del país y la cantidad de habitantes de dicho país. Este método recibe en el tercer parámetro la posición del item seleccionado del ListView.

Cuando ejecutamos el proyecto podemos ver una interfaz en el emulador similar a esta:

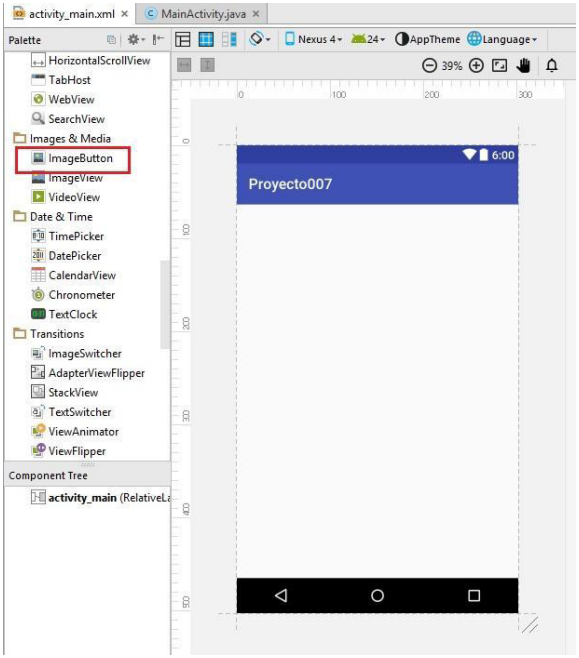
Solución problema 1.5.6.1.

Primero crearemos un proyecto llamado Proyecto007 y luego debemos buscar una imagen en formato jpg que represente un teléfono de 50*50 píxeles.
Nombre del archivo: telefono.jpg (es obligatorio que el nombre de archivo de la imagen siempre debe estar en minúsculas y no tener caracteres especiales, veremos que luego se genera un nombre de variable con dicho nombre)
Ahora copiamos el archivo a la carpeta de recursos de nuestro proyecto : Proyecto007/app/src/main/res/mipmap-mdpi
Esto lo podemos hacer desde el administrador de archivos del sistema operativo Windows.
En la ventana "Project" cambiamos a vista "Project Files" y navegamos hasta la carpeta donde copiamos el archivo, vemos que el archivo está en dicha carpeta:

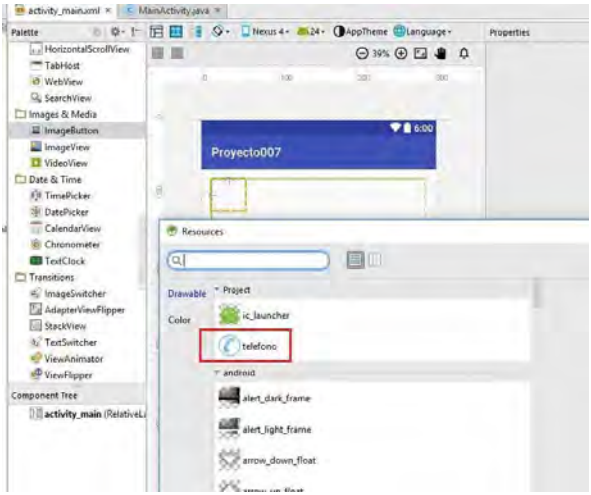


Esto solo a efectos de ver que el proyecto se actualiza con los archivos de recursos que copiamos a las carpetas mipmap-mdpi.

Ahora insertaremos el objeto de la clase ImageButton en el formulario de nuestra aplicación. La clase ImageButton se encuentra en la pestaña "Images & Media":

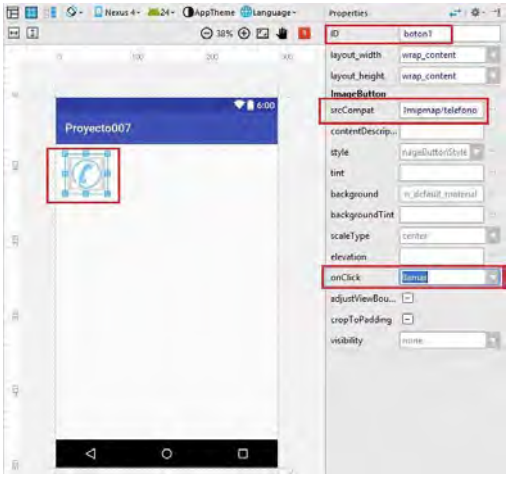


Luego de disponer el objeto se abre un diálogo donde debemos seleccionar la imagen a mostrar:

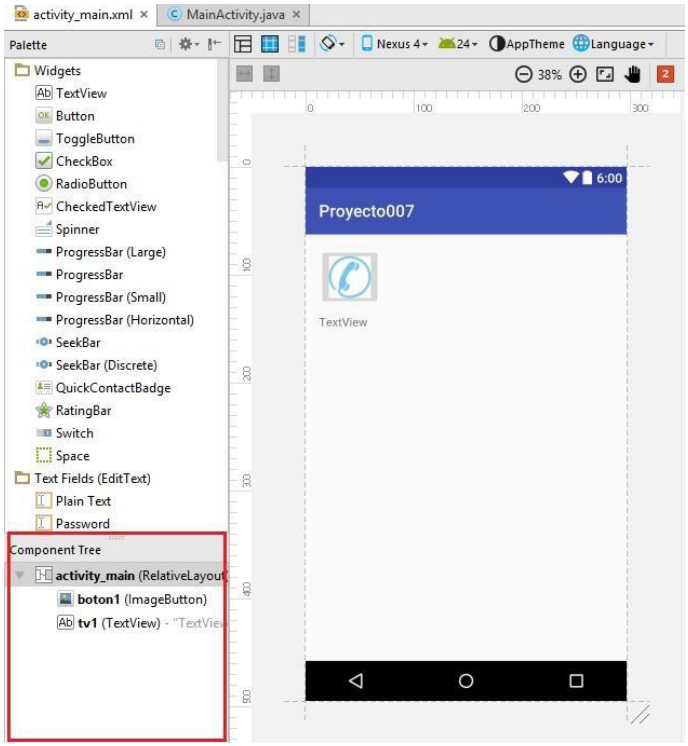


Al cerrar el diálogo se carga la propiedad "srcCompact" la referencia a la imagen (podemos modificar la imagen cambiando esta propiedad)

Inicializamos la propiedad Id con el nombre boton1 y la propiedad "onClick" con el nombre de método "llamar":



Agreguemos un TextView a nuestra aplicación y configuremos sus propiedades Id (con tv1) y text. Luego la interfaz visual debe ser similar a:



Código fuente:

```
package com.tutorialesprogramacionya.proyecto007;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
   private TextView tv1;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);

        tv1=(TextView)findViewById(R.id.tv1);
    }

    //Este método se ejecutará cuando se presione el ImageButton
    public void llamar(View view) {
        tv1.setText("Llamando");
    }
}
```

Definimos el atributo de tipo TextView:

private TextView tv1;

Enlazamos el control definido en el archivo XML y la variable de java:

tv1=(TextView)findViewById(R.id.tv1);

Implementamos el método que se ejecutará cuando se presione el objeto de la clase ImageButton:

public void llamar(View view) {

tv1.setText("Llamando");

}

No olvidemos inicializar la propiedad onClick del objeto boton1 con el nombre del método "llamar" (recordemos que esto lo hacemos accediendo a la propiedad onClick en la ventana de "Properties")

Este proyecto lo puede descargar en un zip desde este enlace: proyecto007.zip

Comentarios extras de este control.

Cuando copiamos el archivo lo hicimos a la carpeta:

mipmap-mdpi

Pero vimos que hay otras carpetas con nombres similares:

mipmap-mdpi

mipmap-hdpi

mipmap-xhdpi

mipmap-xxhdpi

mipmap-xxxhdpi

El objetivo de estas otras carpetas es guardar las mismas imágenes pero con una resolución mayor si la guardamos en mipmap-hdpi.

Esto se hace si queremos que nuestra aplicación sea más flexible si se ejecuta en un celular, en una tablet, en un televisor etc.

Debemos tener en cuenta estos tamaños:

En la carpeta res/mipmap-mdpi/

(guardamos la imagen con el tamaño original)

En la carpeta res/mipmap-hdpi/

(guardamos la imagen con el tamaño del 150% con respecto al de la carpeta mipmap-mdpi)

En la carpeta res/mipmap-xhdpi/

(guardamos la imagen con el tamaño del 200% con respecto al de la carpeta mipmap-mdpi)

En la carpeta res/mipmap-xxhdpi/

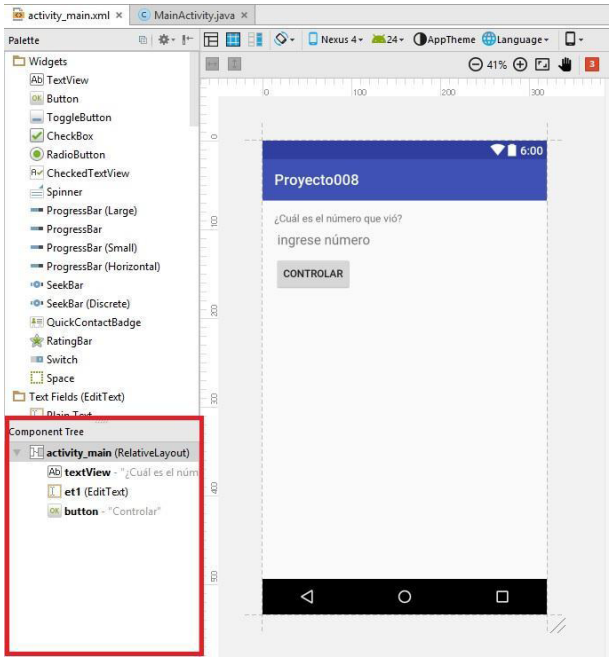
(guardamos la imagen con el tamaño del 300% con respecto al de la carpeta mipmap-mdpi)

En la carpeta res/mipmap-xxxhdpi/

(guardamos la imagen con el tamaño del 400% con respecto al de la carpeta mipmap-mdpi)

Solución problema 1.5.7.1.

Lo primero que hacemos es crear la interfaz siguiente:



Es decir que disponemos un TextView, un EditText ("Number") y un Button. Dejamos por defecto los nombres asignados a las ID de los objetos: TextView y Button pero al objeto de tipo EditText definimos su ID con el valor "et1". Recordemos que en la ventana "Component Tree" tenemos la referencia de todos los objetos que contiene nuestra interfaz.

Luego seleccionamos el botón de la interfaz visual e inicializamos la propiedad onClick con el nombre del método que se ejecutará al ser presionado, dicho nombre debe ser: "controlar" (recordemos que los nombre de métodos en Java por convención empiezan en minúsculas)

Código fuente:

```
package com.tutorialesprogramacionya.proyecto008;


import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
   private EditText et1;
   private int num;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      et1=(EditText)findViewById(R.id.et1);
      num=(int)(Math.random()*100001);
      String cadena=String.valueOf(num);
      Toast notificacion= Toast.makeText(this,cadena,Toast.LENGTH_LONG);
      notificacion.show();
   }

   public void controlar(View v) {
      String valorIngresado=et1.getText().toString();
      int valor=Integer.parseInt(valorIngresado);
      if (valor==num) {
          Toast notificacion=Toast.makeText(this,"Muy bien recordaste el número
mostrado.",Toast.LENGTH_LONG);
         notificacion.show();
      }
      else {
        Toast notificacion=Toast.makeText(this,"Lo siento pero no es el número que
mostré.",Toast.LENGTH_LONG);
         notificacion.show();
      }
   }
}
```

Analicemos el código fuente de la aplicación, lo primero es definir un objeto de la clase EditText y una variable entera donde almacenaremos el valor aleatorio que generamos y mostramos en un principio:

```
private EditText et1;

private int num;
```

En el método onCreate que es el primero que se ejecuta al inicializar la aplicación es donde inicializamos la referencia del EditText:

```
et1=(EditText)findViewById(R.id.et1);
```

Generamos un valor aleatorio comprendido entre 0 y 100000 y lo almacenamos en la variable num:

```
num=(int)(Math.random()*100001);
```

Convertimos el valor entero a String ya que la clase Toast siempre hay que pasarte un String para mostrar:

```
String cadena=String.valueOf(num);
```

Veamos ahora lo nuevo que es la clase Toast, tenemos que crear un objeto de la clase Toast, para ello definimos una variable que la llamamos notificacion y mediante el método estático makeText de la clase Toast creamos el objeto.

El método makeText tiene tres parámetros: el primero hace referencia a la ventana o Activity donde aparece (this), el segundo el la variable de tipo String que se mostrará en pantalla y por último es una constante que indica que la notificación se mostrará por un tiempo largo o corto:

```
Toast notificacion=Toast.makeText(this,cadena,Toast.LENGTH_
LONG);
```

Finalmente cuando queremos que se muestre la notificación en pantalla procedemos a llamar al método show de la clase Toast:

```
notificacion.show();
```

Esto hace que inmediatamente arranquemos la aplicación se mostrará la notificación con el número que deberá memorizar el usuario:

Luego de unos segundos desaparece la notificación de pantalla (es decir en nuestro programa desaparece de pantalla el número aleatorio)

Cuando el operador termina de cargar el número y procede a ejecutar el botón "controlar" se ejecuta el código que dispusimos en el método "controlar":

```
public void controlar(View v) {
    String valorIngresado=et1.getText().toString();
    int valor=Integer.parseInt(valorIngresado);
    if (valor==num) {
        Toast notificacion=Toast.makeText(this,"Muy bien recordaste el número mostrado.",Toast.LENGTH_LONG);
        notificacion.show();
    }
    else {
        Toast notificacion=Toast.makeText(this,"Lo siento pero no es el número que mostré.",Toast.LENGTH_LONG);
        notificacion.show();
    }
}
```

En este método extraemos del control EditText el número que ingreso el usuario:

String valorIngresado=et1.getText().toString();

Lo convertimos a entero:

int valor=Integer.parseInt(valorIngresado);

Y mediante un if verificamos si coincide con el número aleatorio ingresado por el operador, si coincide inmediatamente creamos una nueva notificación y procedemos a mostrar que acertó:

```
if (valor==num) {
    Toast notificacion=Toast.makeText(this,"Muy bien recordaste el número mostrado.",Toast.LENGTH_LONG);
    notificacion.show();
}
```

Por el falso mediante otra notificación mostramos que no ingresó correctamente el número:

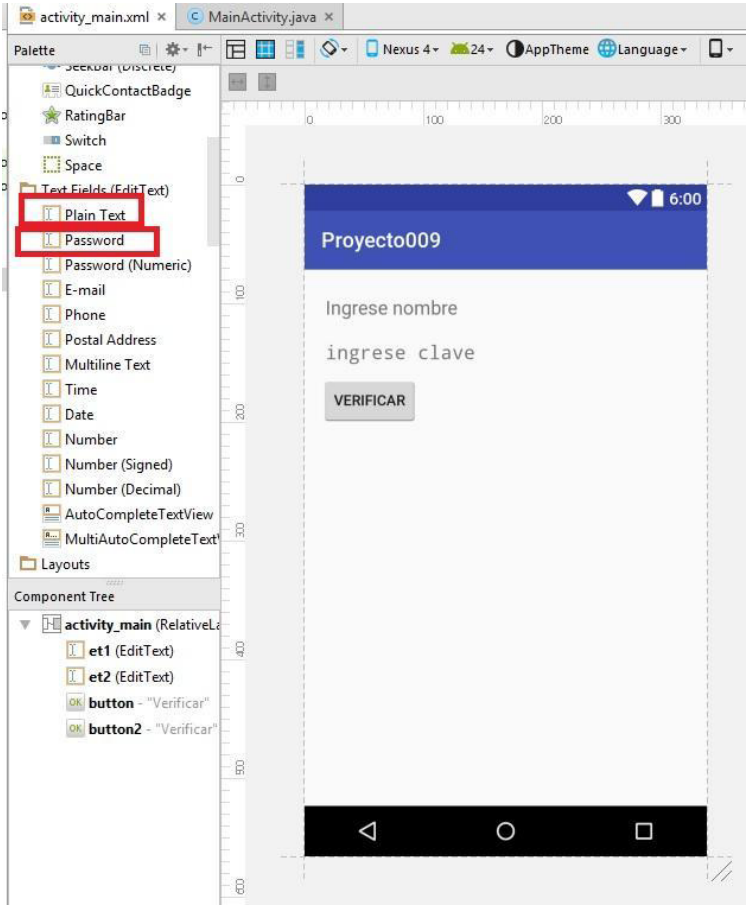
```
else {
    Toast notificacion=Toast.makeText(this,"Lo siento pero no es el número que mostré.",Toast.LENGTH_LONG);
    notificacion.show();
}
```

Si queremos que aparezca de nuevo un número aleatorio debemos cerrar el programa y volverlo a ejecutar.

Solución problema 1.5.8.1.

Utilizar el método length() de la clase String para ver cuantos caracteres se ingresaron.

La interfaz visual debe ser similar a esta:



El EditText para el ingreso del nombre es de tipo “Plain Text”, iniciamos la propiedad “hint” con el valor “ingrese nombre”. El segundo EditText es de tipo “Password” e iniciamos la propiedad “hint” con el valor “ingrese clave”.

Definamos el ID de los dos controles de tipo EditText con los valores “et1” y “et2”. Iniciamos la propiedad onClick del Button con el nombre de método “verificar”

Código fuente:

```
package com.tutorialesprogramacionya.proyecto009;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
   private EditText et1,et2;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      et1=(EditText)findViewById(R.id.et1);
      et2=(EditText)findViewById(R.id.et2);
   }

   public void verificar(View v) {
      String clave=et2.getText().toString();
      if (clave.length()==0) {
         Toast notificacion= Toast.makeText(this,"La clave no puede quedar vacía",
Toast.LENGTH_LONG);
         notificacion.show();
      }
   }
}
```

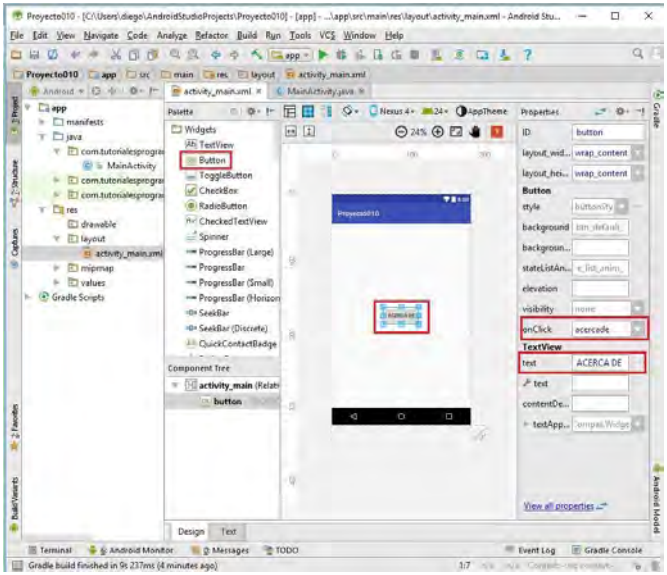
Como podemos ver cuando se presiona el botón "verificar" se procede a extraer el contenido del EditText de la clave y mediante el método length() controlamos si tiene cero caracteres, en caso afirmativo mostramos la notificación en pantalla:

```
public void verificar(View v) {
    String clave=et2.getText().toString();
    if (clave.length()==0) {
        Toast notificacion= Toast.makeText(this,"La clave no puede quedar vacía",-
Toast.LENGTH_LONG);
        notificacion.show();
    }
}
```

En pantalla tendremos un resultado similar a esto si no se ingresa una clave:

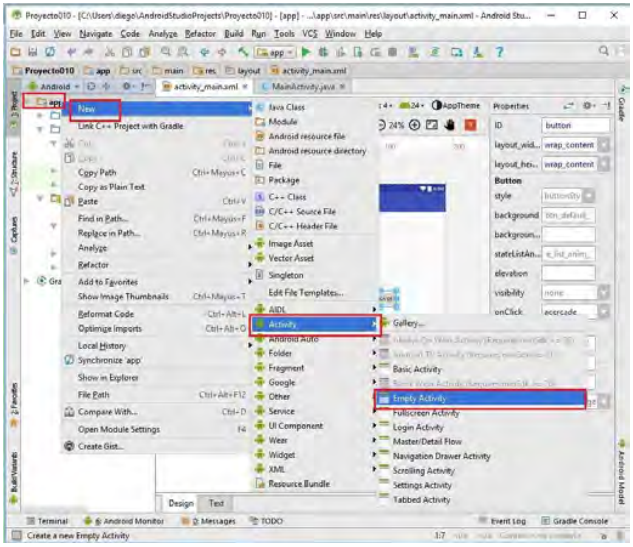
Solución problema 1.5.9.1.

1–Primero creamos un nuevo proyecto que lo llamaremos Proyecto010 y en la ventana principal creamos la siguiente interfaz:

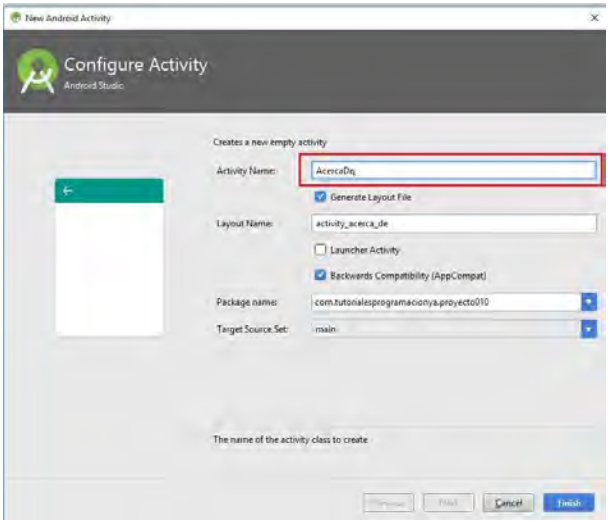


Es decir que nosotros queremos que cuando se presione el botón "ACERCA DE" nos abra otra ventana (Activity) y nos muestre el nombre del programador y un botón para cerrar dicha ventana.

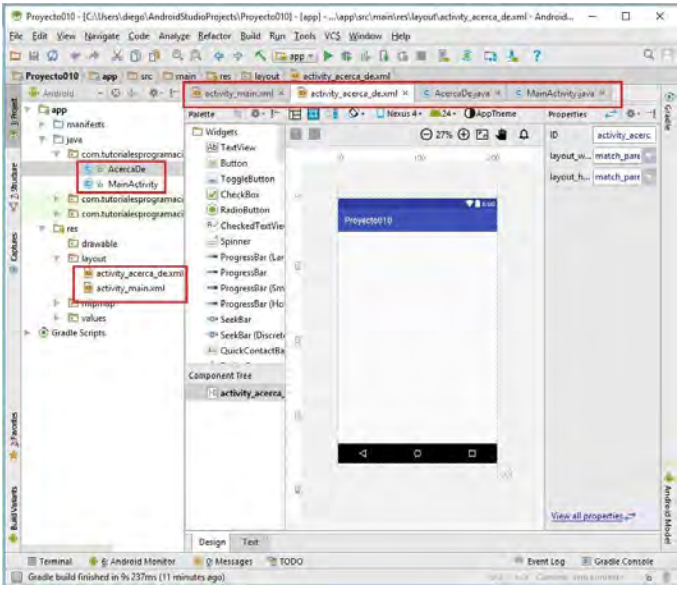
2–Ahora tenemos que crear el segundo Activity. Para esto hacemos clic con el botón derecho del mouse en la ventana de Project donde dice "app" y seleccionamos New -> Activity -> Empty Activity



Aparece un diálogo donde debemos definir el nombre del Activity "Activity Name" y le asignaremos como nombre "AcercaDe" (se crearán dos archivos AcercaDe.java y activity_acerca_de.xml):



Ya tenemos los cuatro archivos necesarios para implementar la aplicación:



Los dos primeros que se crean cuando iniciamos el proyecto:

activity_main.xml

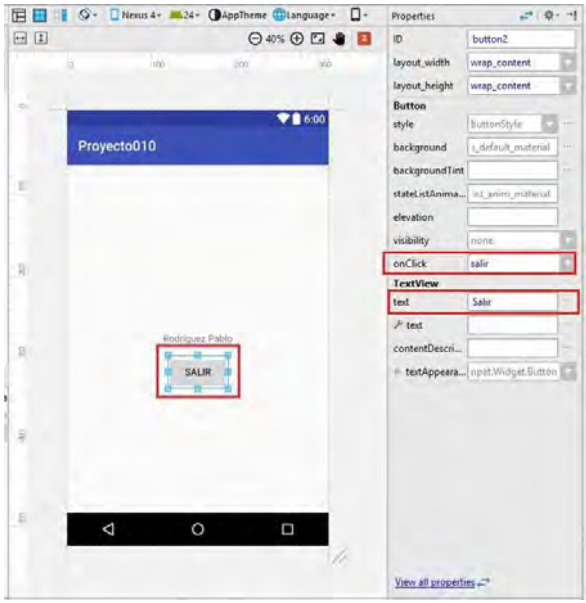
MainActivity.java

Y estos dos nuevos archivos para la segundo ventana:

activity_acerca_de.xml

AcercaDe.java

Implementamos la interfaz visual del segundo Activity es decir del archivo activity_acerca_de.xml con los siguientes controles:



3–Ahora implementaremos la funcionalidad en la actividad (Activity) principal para que se active la segunda ventana.

Inicializamos la propiedad onClick del objeto de la clase Button con el valor "acercade", este es el método que se ejecutará cuando se presione.

El código fuente de la actividad principal queda:

```
package com.tutorialesprogramacionya.proyecto010;


import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
import android.view.View;

public class MainActivity extends AppCompatActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }

   public void acercade(View view) {
      Intent i = new Intent(this, AcercaDe.class );
      startActivity(i);
   }
}
```

En el método acercade creamos un objeto de la clase Intent y le pasamos como parámetros la referencia del objeto de esta clase (this) y la referencia del otro Activity (AcercaDe.class) Llamamos posteriormente al método startActivity pasando el objeto de la clase Intent (con esto ya tenemos en pantalla la ventana del segundo Activity):

```
public void acercade(View view) {
Intent i = new Intent(this, AcercaDe.class );
startActivity(i);
}
```

Si ejecutamos el programa podemos ver que cuando presionamos el botón "Acerca De" aparece la segunda ventana donde se muestra el TextView con el valor "Autor : Rodriguez Pablo" y un botón con el texto "salir" (si presionamos este botón no sucede nada, esto debido a que no hemos asociado ningún evento a dicho botón)

4–Debemos codificar el evento onClick de la segunda actividad. Seleccionemos el botón "salir" y definamos en la propiedad onClick el nombre de método que se ejecutará al presionarse el botón (en nuestro caso lo llamaremos "salir") :

El código fuente de la actividad AcercaDe queda:

```
package com.tutorialesprogramacionya.proyecto010;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class AcercaDe extends AppCompatActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_acerca_de);
   }

   public void salir(View v) {
      finish();
   }
}
```

Cuando se presiona el botón salir se ejecuta el método "salir" llamando al método finish() que tiene por objetivo liberar el espacio de memoria de esta actividad y pedir que se muestre la actividad anterior.

Ahora nuestro programa está funcionando completamente:

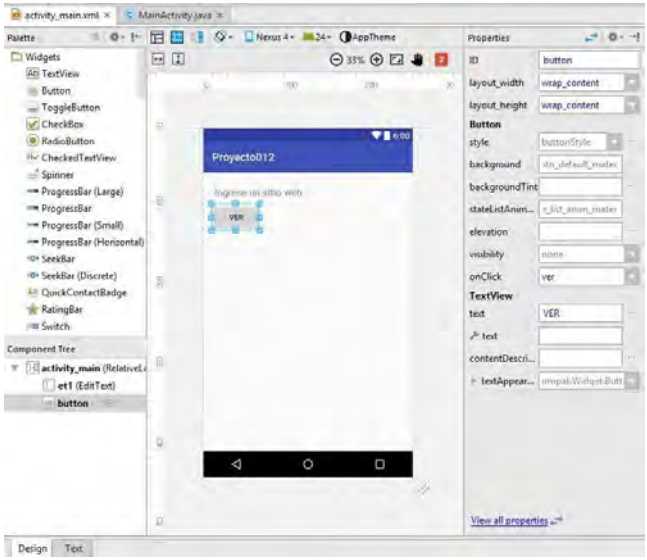
Primer Activity:

Segundo Activity:

Este proyecto lo puede descargar en un zip desde este enlace: proyecto010.zip

Solución problema 1.5.10.1.

1–Nuestro primer Activity tendrá la siguiente interfaz visual (ver controles):



Tenemos un control de tipo tipo EditText (ID = et1, hint = "Ingrese un sitio web") y otro de tipo Button (inicializar la propiedad onClick con el nombre de método llamado "ver").

El código fuente de esta Activity es:

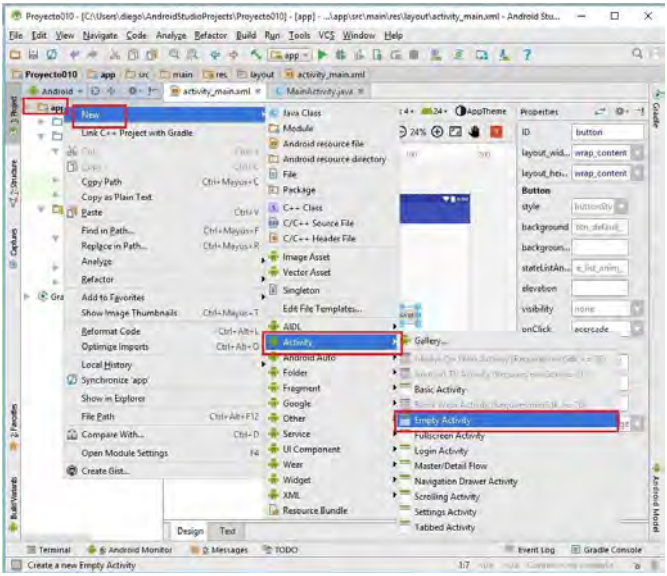
```
package com.tutorialesprogramacionya.proyecto012;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {
   private EditText et1;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        et1=(EditText)findViewById(R.id.et1);
    }

    public void ver (View v) {
        Intent i=new Intent(this,Actividad2.class);
        i.putExtra("direccion", et1.getText().toString());
        startActivity(i);
    }
}
```

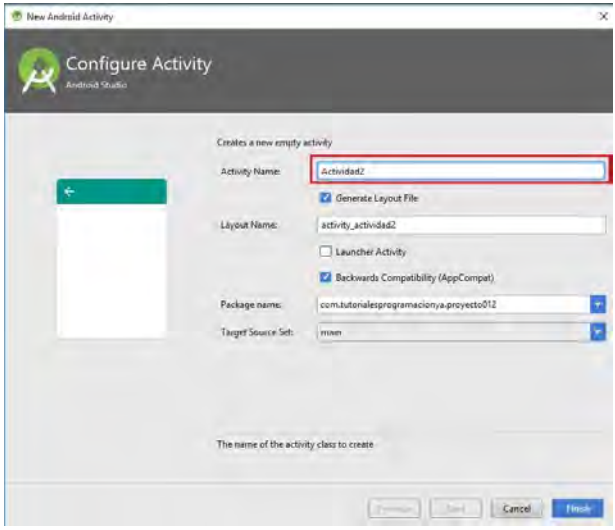
Como podemos ver la diferencia con el concepto anterior es que llamamos al método putExtra de la clase Intent. Tiene dos parámetros de tipo String, en el primero indicamos el nombre del dato y en el segundo el valor del dato:

```
  public void ver(View v) {
      Intent i=new Intent(this,Actividad2.class);
      i.putExtra("direccion", et1.getText().toString());
      startActivity(i);
    }
```

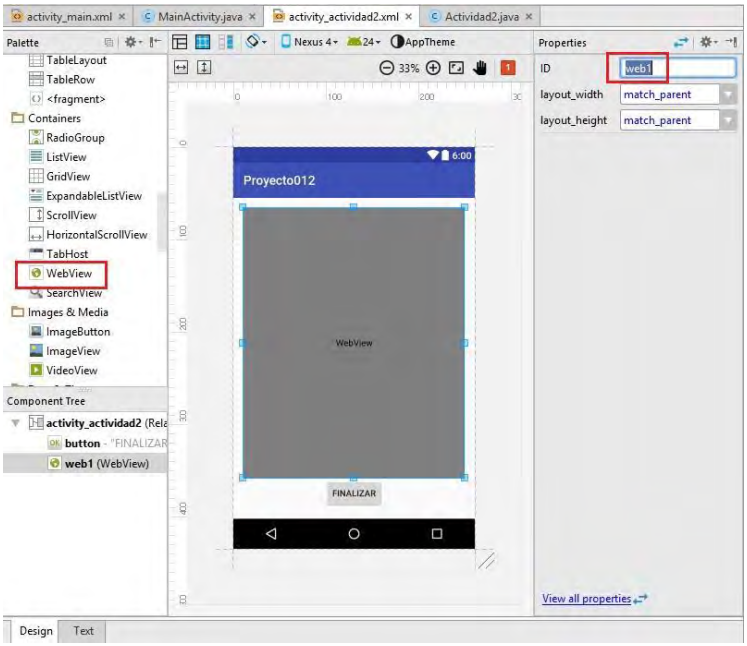
La segunda interfaz visual (recordemos que debemos presionar el botón derecho en la ventana Project (sobre app) y seleccionar la opción New -> Activity -> Emplty Activity):



En el diálogo que aparece especificamos el nombre de nuestra segundo Activity y lo llamaremos Actividad2:



Disponemos un objeto de la clase WebView que se encuentra en la pestaña “Containers” y un Button:



Codificamos la funcionalidad de la segunda actividad

```
package com.tutorialesprogramacionya.proyecto012;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.webkit.WebView;

public class Actividad2 extends AppCompatActivity {
   WebView web1;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_actividad2);

      web1=(WebView)findViewById(R.id.web1);
```

```
    Bundle bundle = getIntent().getExtras();
    String dato=bundle.getString("direccion");
    web1.loadUrl("http://" + dato);
  }

  public void finalizar(View v) {
    finish();
  }
}
```

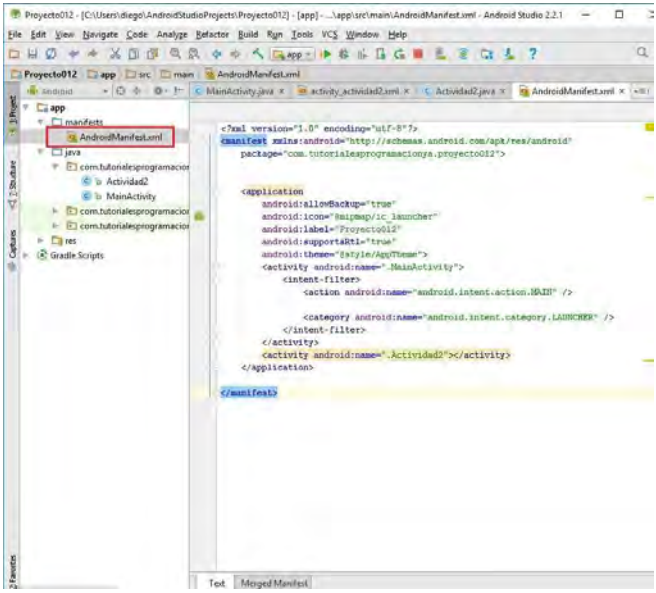
En esta clase definimos una variable de tipo Bundle y la inicializamos llamando al método getExtras() de la clase Intent (esto lo hacemos para recuperar el o los parámetros que envió la otra actividad (Activity)):

```
Bundle bundle = getIntent().getExtras();
String dato=bundle.getString("direccion");
web1.loadUrl("http://" + dato);
```

El método loadUrl de la clase WebView permite visualizar el contenido de un sitio web.

Importante

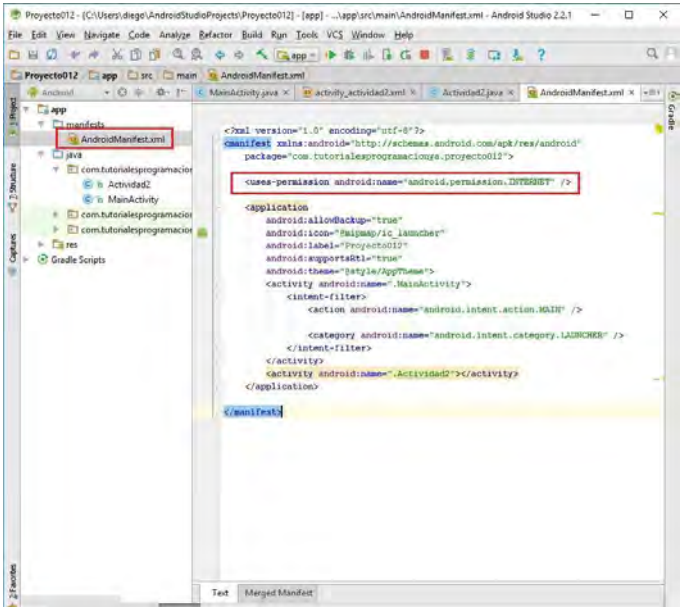
Como nuestra aplicación debe acceder a internet debemos hacer una configuración en el archivo "AndroidManifest.xml", podemos ubicar este archivo:



Agregamos el permiso tipeando lo siguiente en este archivo:

<uses-permission android:name=”android.permission.INTERNET” />

Luego el archivo AndroidManifest.xml queda con el permiso agregado:



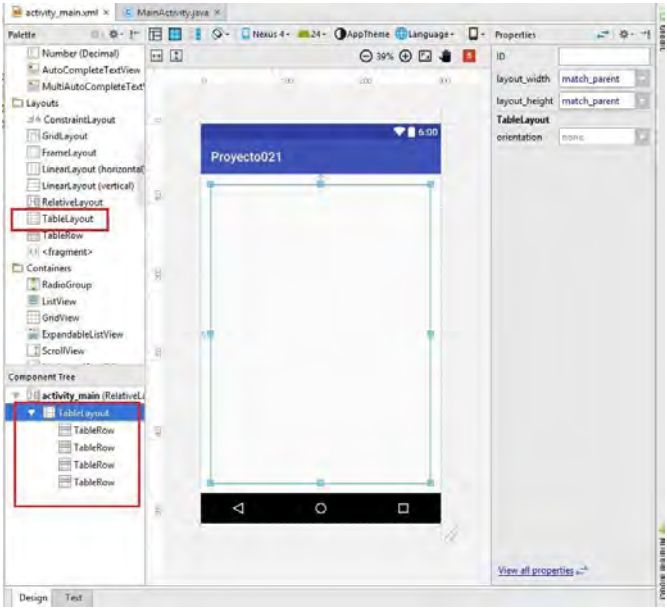
Ahora nuestro programa debería estar funcionando completamente.

La primer ventana debería ser algo similar a esto:

La segunda ventana debería ser algo similar a esto otro:

Solución problema 1.6.3.1.

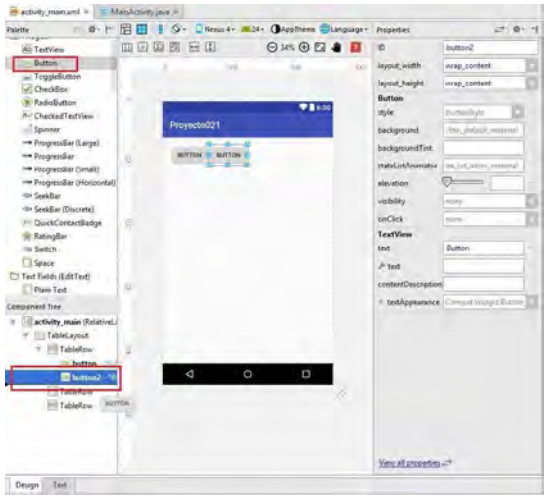
Primero creemos el Proyecto021 y vamos a la pestaña Layout, identifiquemos la componente "TableLayout" y la arrastramos al interior de la interfaz visual, cuando disponemos el objeto dentro de la ventana podemos ver en el "Component Tree" que el TabletLayout contiene 4 objetos de tipo "Table Row":



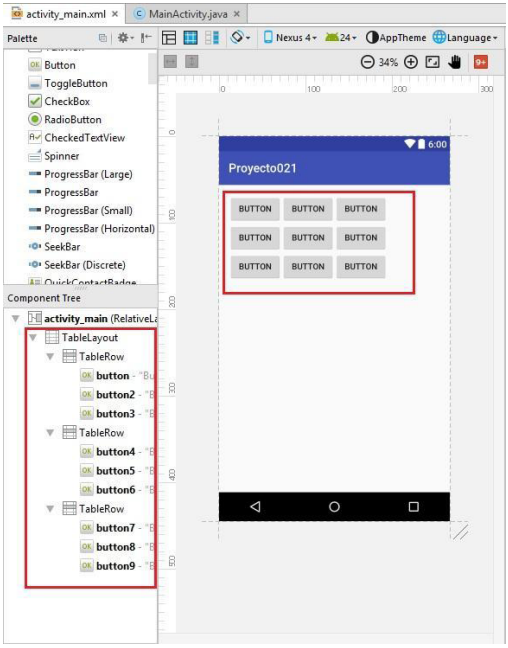
Como nuestro problema requiere solo tres filas procedemos a borrar un "TableRow" desde la ventana "Component Tree".

Luego de esto tenemos toda la pantalla cubierta con una componente de tipo "TableLayout", ahora tenemos que empezar a disponer cada uno de los botones en cada fila.

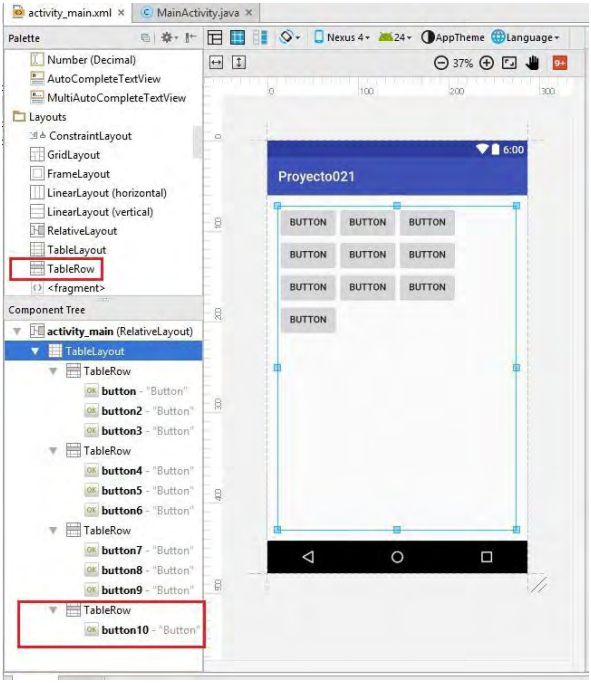
La forma más sencilla es arrastrar los botones no a la vista de diseño sino a la ventana "Component Tree" a la fila correspondiente:



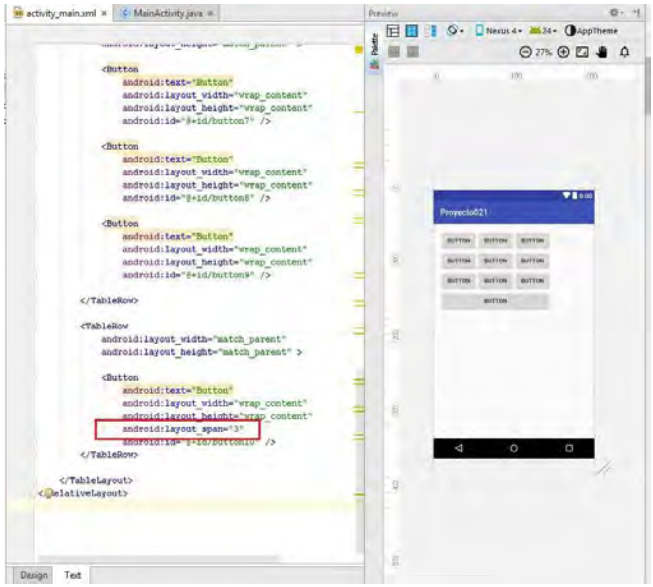
Debemos arrastrar cada uno de los 9 botones al TableRow respectivo :



Otra posibilidad que nos da este Layout es que un control se expanda más de una celda.
Disponer un cuarto "TableRow" de la pestaña "Layouts" y agregar un botón en la cuarta fila del TableLayout:



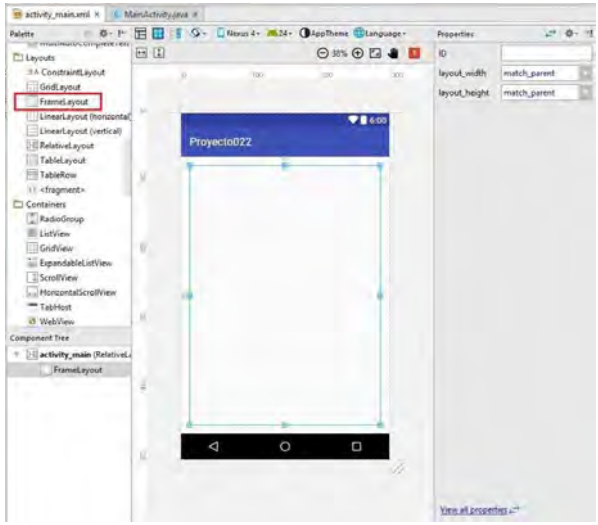
Ahora cambiamos a modo de "Text" en Android Studio y procedemos a agregar la propiedad "layout_span" con el valor 3:



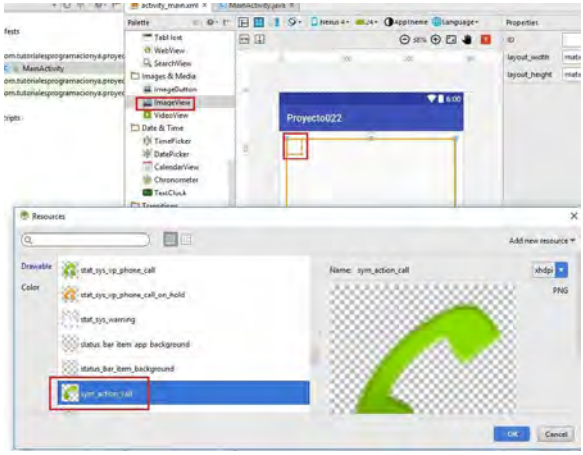
Solución problema 1.6.4.1.

Crear un proyecto en Android Studio y definir como nombre: Proyecto022

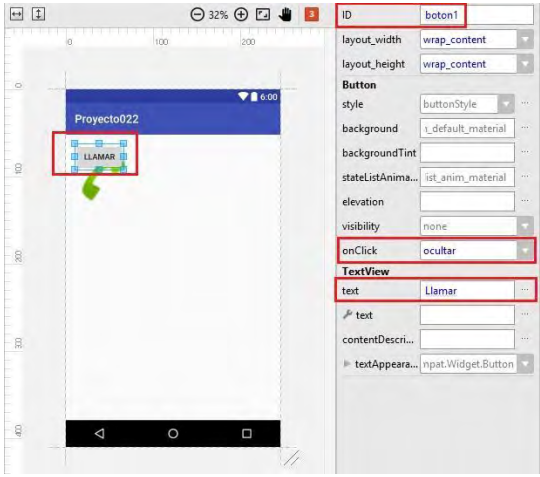
Disponemos un FrameLayout:



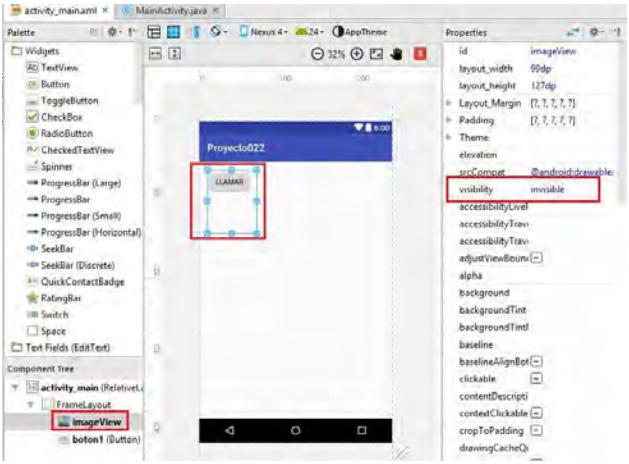
Seguidamente dentro del mismo agregamos un ImageView y seleccionamos una imagen que ya tiene el sistema Android:



También disponemos un botón dentro del FrameLayout e iniciamos las propiedades ID, onClick y Text:



Seleccionamos el control ImageView y fijamos la propiedad visibility con el valor invisible (esto hace que la imagen no se muestre en pantalla), tener en cuenta que para acceder a esta propiedad de la clase ImageView debemos cambiar la vista de propiedades a “View all properties”:

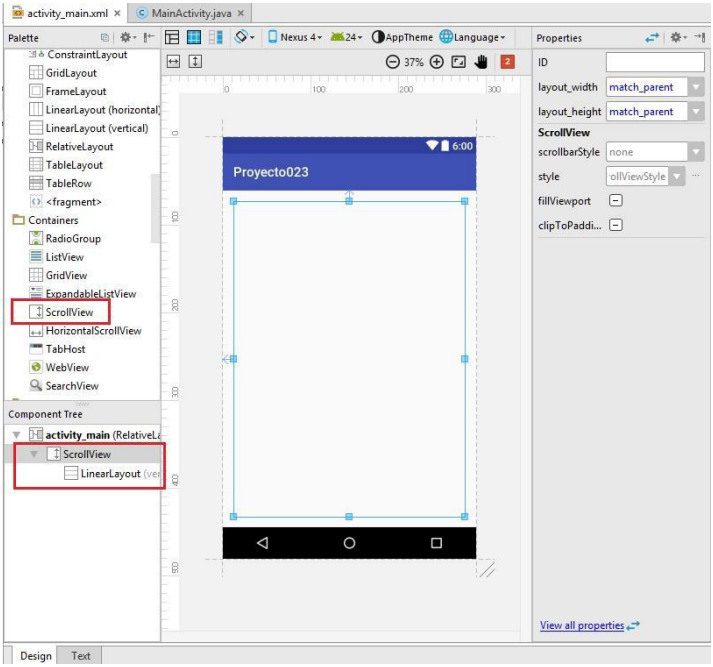


El código fuente de la clase es:

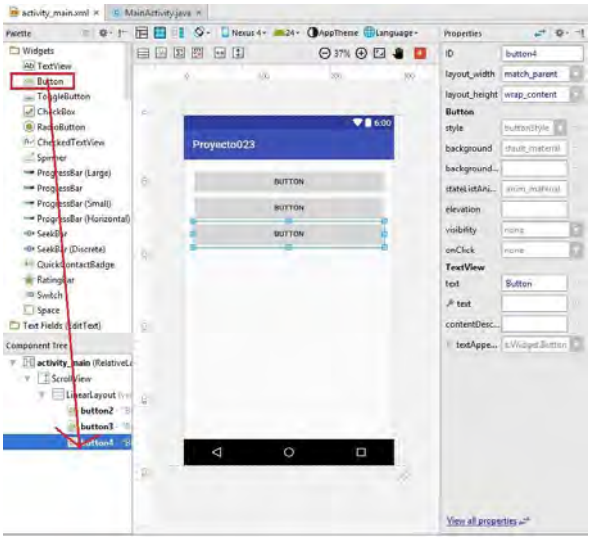
```
package com.tutorialesprogramacionya.proyecto022;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity {
   private ImageView iv1;
   private Button b1;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      iv1=(ImageView)findViewById(R.id.imageView);
      b1=(Button)findViewById(R.id.boton1);
   }
```

```
    public void ocultar(View v) {
        b1.setVisibility(View.INVISIBLE);
        iv1.setVisibility(View.VISIBLE);
    }
}
```

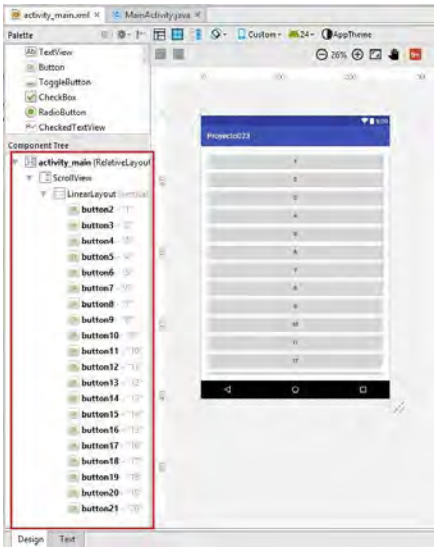
Solución problema 1.6.5.1.



Ahora vamos a disponer 20 objetos de la clase Button dentro del LinearLayout (como no van a entran en pantalla iremos arrastrando los botones a la ventana "Component Tree" dentro del objeto "LinearLayout"):



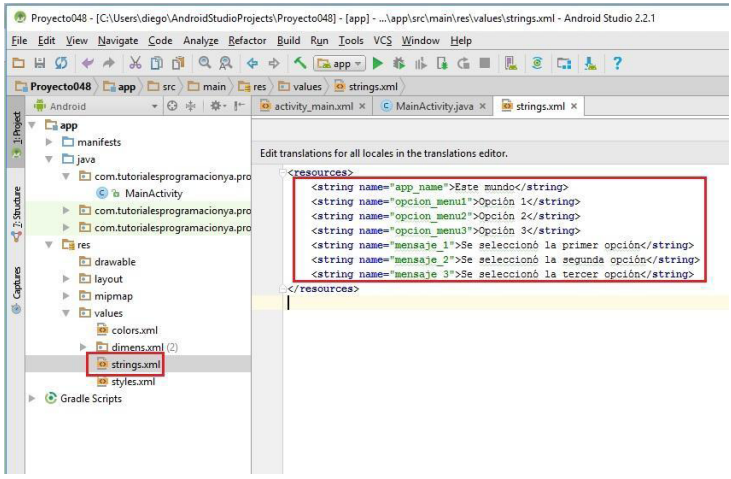
Seguimos arrastrando botones dentro del LinearLayout hasta completar los 20 (luego cambiemos la propiedad text), como podemos observar hay más botones dentro del LinearLayout que los que puede mostrarse en la interfaz del dispositivo (por eso tuvimos que arrastrarlos a la ventana del "Component Tree"):



Gracias a la funcionalidad del ScrollView junto al LinearLayout ahora en tiempo de ejecución podemos hacer scroll:

Solución problema 1.7.1.2.

Creemos un proyecto y luego creamos el archivo strings.xml con todas las constantes necesarias de nuestro programa:



Es decir modificamos el valor de app_name y creamos seis constantes más, tres para las etiquetas de las opciones del menú y tres para los mensajes que deben aparecer cuando se presionen las opciones.

El archivo menuopciones ahora queda codificado así (recordemos que lo podemos hacer en forma visual):

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:android="http://schemas.android.com/apk/res/android">

  <item android:title="@string/opcion_menu1"
    android:id="@+id/ocion1" />
  <item android:title="@string/opcion_menu2"
    android:id="@+id/opcion2" />
  <item android:title="@string/opcion_menu3"
    android:id="@+id/opcion3" />
</menu>
```

Es decir hemos inicializado cada propiedad title de los item con los nombres de las constantes que creamos en el archivo strings.xml.

Finalmente el código fuente de la aplicación sufre los siguientes cambios para utilizar las constantes del archivo strings.xml cuando se deben mostrar los mensajes:

```
package com.tutorialesprogramacionya.proyecto048;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {


  @Override
  protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menuopciones, menu);
    return true;
  }

  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id==R.id.opcion1) {
      Toast.makeText(this,R.string.mensaje_1,Toast.LENGTH_LONG).show();
    }
    if (id==R.id.opcion2) {
      Toast.makeText(this,R.string.mensaje_1,Toast.LENGTH_LONG).show();
    }
    if (id==R.id.opcion3) {
      Toast.makeText(this,R.string.mensaje_1, Toast.LENGTH_LONG).show();
    }
    return super.onOptionsItemSelected(item);
  }
}
```

Lo nuevo aquí es ver que antes para mostrar el mensaje directamente en el código Java hacíamos:

```
if (id==R.id.opcion1) {
      Toast.makeText(this,"Se seleccionó la primer opción",Toast.LENGTH_
LONG).show();
    }
```

Ahora vemos si queremos extraer el dato de la constante que se encuentra en el archivo strings.xml lo debemos hacer llamando al método getString y pasando la referencia al nombre de la constante que se encuentra en la clase R y dentro de esta en una clase interna llamada string:

```
if (id==R.id.opcion1) {
        Toast.makeText(this,getString(R.string.mensaje_1),Toast.LENGTH_
LONG).show();
}
```

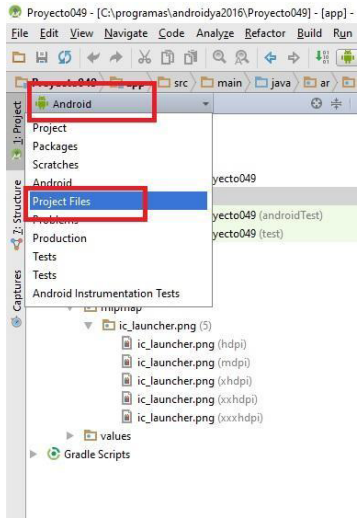
Solución problema 1.7.2.1.

Como podemos observar hay dos botones de acción siempre visibles (podemos mostrar normalmente su ícono, pero si disponemos de dispositivos más grandes podemos mostrar un texto inclusive)

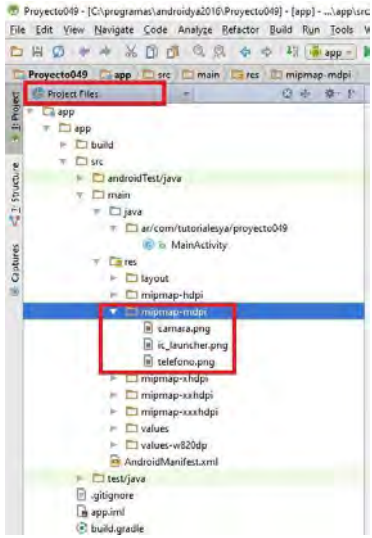
Veamos los pasos que debemos dar para obtener este resultado:

1. Lo primero es disponer dos imágenes de 32x32 píxeles en la carpeta mipmap-mdpi, una que represente un teléfono y otra una cámara fotográfica (recordemos que los nombres de archivos deben estar en minúsculas, sin espacios, no empezar con un número y no contener caracteres especiales, salvo el guión bajo), podemos llamar a los archivos telefono.png y camara.png Dispondremos solo estas dos imágenes en en la carpeta mipmap-mdpi.

Lo más fácil para insertar estos archivos a dicha carpeta es disponer la ventana de “Project” en vista de “Project File”



Y ahora ubicar la carpeta mipmap-mdpi y copiar las dos imágenes:



2. El título del ActionBar lo debemos modificar abriendo el archivo strings que se encuentra en la carpeta res/values y su contenido debe ser:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
        <string name="app_name">ActionBar</string>
</resources>
```

Este archivo de recursos se utiliza para agrupar todos los mensajes que aparecen en pantalla y facilitar la implementación de aplicaciones para varios idiomas independiente de los algoritmos de la misma.

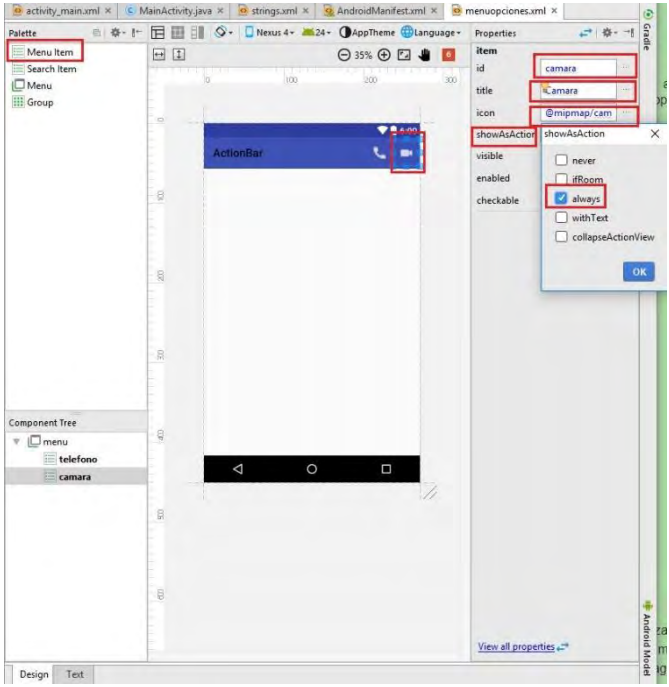
En este archivo hemos creado un string llamado "app_name" con el valor "ActionBar". Luego este string se asocia con el título de la aplicación mediante el archivo AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.tutorialesprogramacionya.proyecto049">

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

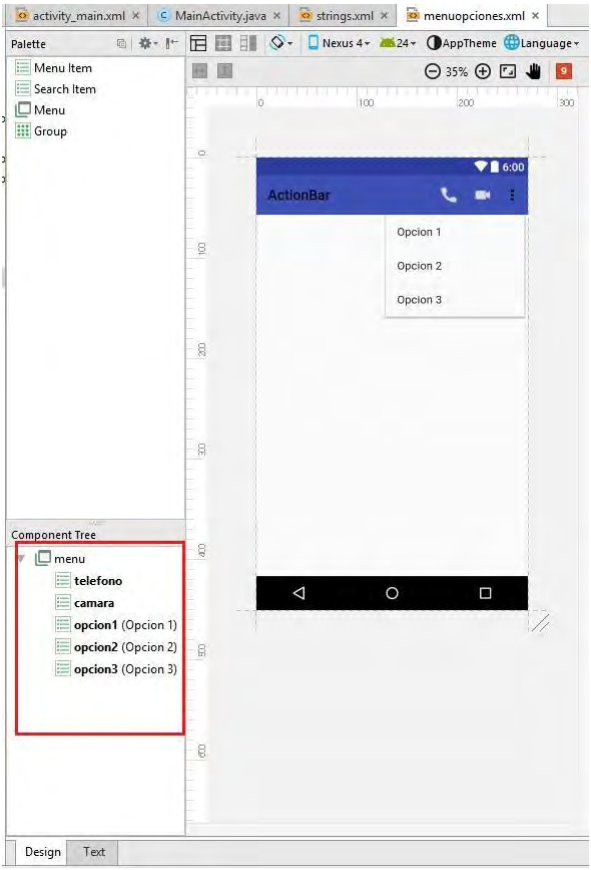
En este archivo XML ya está inicializada la propiedad label de la marca application con el valor definido en el archivo xml (como vemos acá hacemos referencia al string app_name):

android:label="@string/app_name"

3. Ahora tenemos que definir las opciones de nuestro menú desplegable, debemos crear un archivo XML con las opciones, para esto presionamos el botón derecho sobre la carpeta res y seleccionamos la opción New -> Android resource file:
En este diálogo indicamos el nombre del archivo a crear "menuopciones" y el tipo de recurso (Resource type) de tipo Menu:
Por ejemplo al definir la opción de la cámara debemos configurar las siguientes propiedades:



Al final debe quedar la interfaz del menú:



El archivo menuopciones.xml completo debe ser:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:android="http://schemas.android.com/apk/res/android">

   <item android:title="Telefono"
      android:id="@+id/telefono"
      app:showAsAction="always"
      android:icon="@mipmap/telefono"/>
   <item android:title="Camara"
```

```
      android:id="@+id/camara"
      android:icon="@mipmap/camara"
      app:showAsAction="always" />
   <item android:title="Opcion 1"
      android:id="@+id/opcion1" />
   <item android:title="Opcion 2"
      android:id="@+id/opcion2" />
   <item android:title="Opcion 3"
      android:id="@+id/opcion3" />
</menu>
```

Lo nuevo con respecto al concepto anterior es la definición de dos item que inicializamos la propiedad showAsAction con el valor "always" (debemos cambiar en el archivo xml el valor "android" por "app") con lo que indicamos que queremos que siempre se muestre visible esta opción en la barra de acción, y además definimos la propiedad icon con el nombre de la imagen que agregamos a la carpeta mipmap:

```
   <item android:title="Telefono"
      android:id="@+id/telefono"
      app:showAsAction="always"
      android:icon="@mipmap/telefono"/>
   <item android:title="Camara"
      android:id="@+id/camara"
      android:icon="@mipmap/camara"
      app:showAsAction="always" />
```

4. La funcionalidad de nuestro programa será mostrar un Toast cuando se seleccione alguno de las opciones del menú o un botón de acción. El código java de la clase MainActivity debe ser:

```
package com.tutorialesprogramacionya.proyecto049;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
```

```
import android.view.MenuItem;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menuopciones, menu);
    return true;
  }

  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id == R.id.telefono) {
        Toast.makeText(this, "Se presionó el ícono del teléfono", Toast.LENG-
TH_LONG).show();
      return true;
    }
    if (id == R.id.camara) {
        Toast.makeText(this, "Se presionó el ícono de la cámara", Toast.LENG-
TH_LONG).show();
      return true;
    }
    if (id == R.id.opcion1) {
        Toast.makeText(this, "Se presionó la opción 1 del menú", Toast.LENG-
TH_LONG).show();
      return true;
    }
    if (id == R.id.opcion2) {
```

```
            Toast.makeText(this, "Se presionó la opción 2 del menú", Toast.LENG-
TH_LONG).show();
            return true;
        }
        if (id == R.id.opcion3) {
            Toast.makeText(this, "Se presionó la opción 3 del menú", Toast.LENG-
TH_LONG).show();
            return true;
        }       return super.onOptionsItemSelected(item);
    }
}
```

Este proyecto lo puede descargar en un zip desde este enlace: proyecto049.zip

Solución problema 1.7.3.1.

El archivo menuopciones.xml debemos disponer:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:title="Opcion 1"
        android:id="@+id/opcion1" />
    <item android:title="Opcion 2"
        android:id="@+id/opcion2" />
    <item android:title="Opcion 3"
        android:id="@+id/opcion3" />
</menu>
```

El código fuente de la aplicación java es:

```
package com.tutorialesprogramacionya.proyecto051;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menuopciones, menu);
    return true;
  }

  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id==R.id.opcion1) {
        Toast.makeText(this,"Se seleccionó la primer opción",Toast.LENGTH_
LONG).show();
    }
    if (id==R.id.opcion2) {
      Toast.makeText(this,"Se seleccionó la segunda opción",Toast.LENGTH_
LONG).show();
    }
    if (id==R.id.opcion3) {
        Toast.makeText(this,"Se seleccionó la tercer opción", Toast.LENGTH_
LONG).show();
    }
    return super.onOptionsItemSelected(item);
  }
```

```
    public void ocultar(View v) {
        getSupportActionBar().hide();
    }

    public void mostrar(View v) {
        getSupportActionBar().show();
    }
}
```

Luego cuando lo ejecutamos podemos observar según el botón que presionamos el ActionBar se hace visible o se oculta:

O aparece oculta:

Para poder ocultar el ActionBar debemos obtener la referencia del objeto mediante el método getSupportActionBar() que se trata de un método heredado de la clase ActionBarActivity:

```
public void ocultar(View v) {
    getSupportActionBar().hide();
}
```

Para volver a hacer visible el ActionBar llamamos al método show(), esto ocurre cuando presionamos el segundo botón:

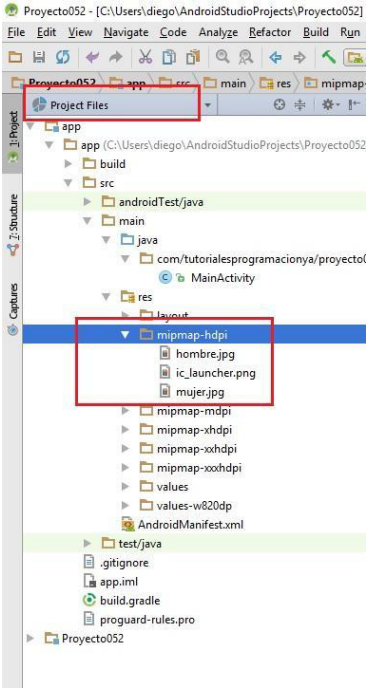
```
public void mostrar(View v) {
    getSupportActionBar().show();
}
```

Solución problema 1.8.1.1.

Creamos un proyecto

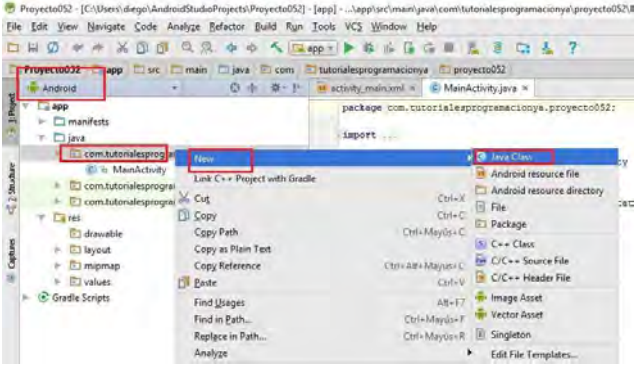
1. Lo primero que haremos es disponer en la carpeta mipmap-hdpi dos archivos de imágenes que representan las caras de un hombre y una mujer. Recordar que los nombres de archivos deben estar en minúsculas y sin caracteres especiales (se crean variables de Java con dichos nombres en el proyecto)

Podemos disponer el Android Studio en la ventana de "Project" con vista de "Project Files" y pegamos los dos archivos en la carpeta respectiva:



2. El segundo paso es crear una clase que represente cada item del ListView (dijimos que cada item representa una persona de género masculino o femenino)

Para crear la clase presionamos el botón derecho del mouse sobre la carpeta que tienen todas las clases del proyecto (hasta ahora solo tenemos MainActivity):



Y creamos la clase Persona, el código fuente de esta clase debe ser:

```
package com.tutorialesprogramacionya.proyecto052;

public class Persona {
   private String nombre;
   private char genero;

   public Persona(String nombre, char genero) {
      this.nombre=nombre;
      this.genero=genero;
   }

   public String getNombre() {
      return nombre;
   }

   public char getGenero() {
      return genero;
   }
}
```

Esta clase representa cada item de la lista y como dijimos tenemos que almacenar el nombre de la persona y el género de la misma:

```
  private String nombre;
   private char genero;
```

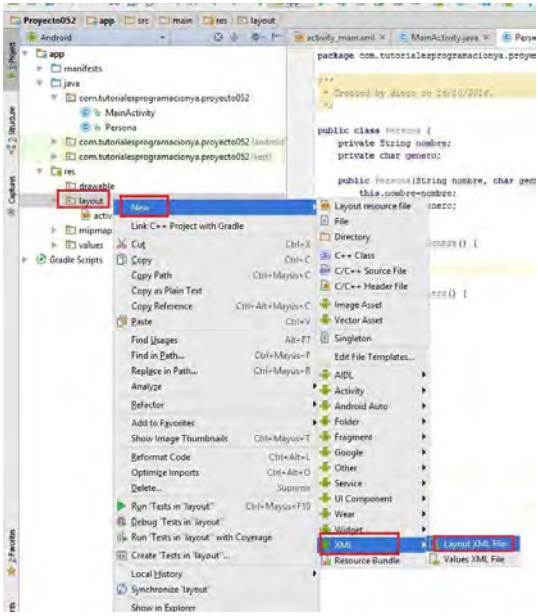
Cuando se ejecute el constructor llegan los dos valores con los que se inicializan los atributos de la clase:

```
   public Persona(String nombre, char genero) {
      this.nombre=nombre;
      this.genero=genero;
   }
```

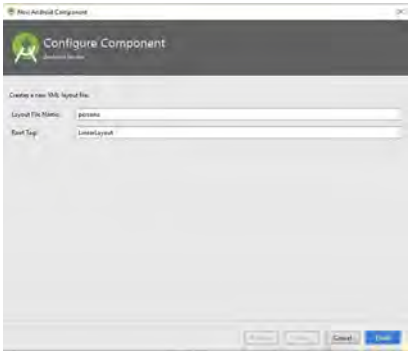
Por otro lado debemos definir dos métodos para poder conocer los valores almacenados en los atributos (recordemos que los atributos están definidos como private y luego mediante estos métodos podemos llamarlos desde otra clase para rescatar sus valores):

```
public String getNombre() {
    return nombre;
}

public char getGenero() {
    return genero;
}
```

3. El tercer paso es crear un archivo XML con la interfaz visual de cada item del ListView. Para esto presionamos el botón derecho en la carpeta layout (que en este momento tenemos solo el archivo activity_main.xml que crea automáticamente el Android Studio cuando creamos el proyecto):

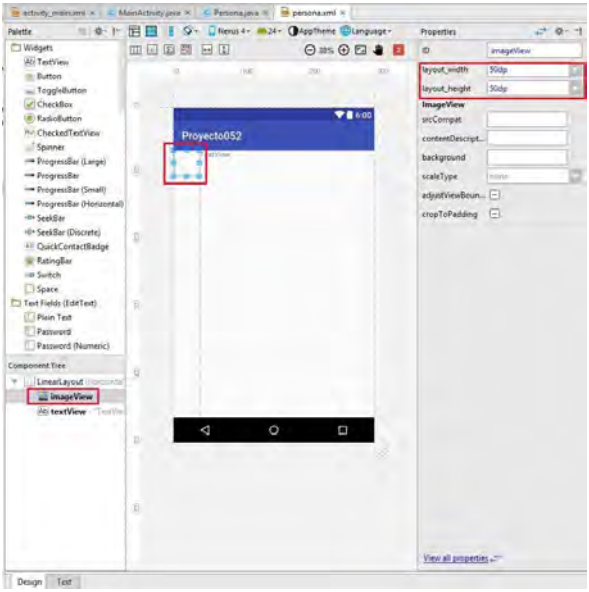


Creamos un archivo llamado persona (debe ser con minúscula el nombre del recurso):

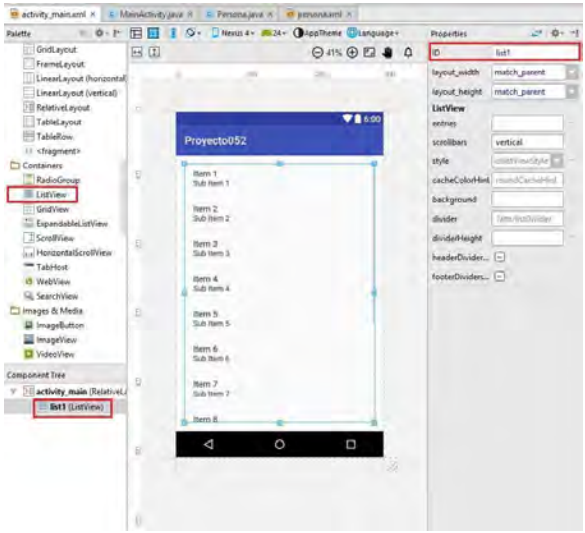


Y creamos la interfaz que tiene que tener cada item, es decir un ImageView y un TextView. No hay que confundir que esta interfaz no se asocia a un Activity (es decir no existirá una ventana que muestre la interfaz que crearemos aquí)

Disponemos un ImageView donde inicializamos las propiedades layout_width y layout_height con los valores de 50dp:



4. El cuarto paso es en nuestro archivo activity_main procedemos a arrastrar un objeto de la clase ListView y definir su ID con el valor list1:



5. Ahora pasaremos a codificar la clase MainActivity.java donde veremos como vinculamos todos los recursos creados hasta este momento:

```
package com.tutorialesprogramacionya.proyecto052;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.TextView;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {
```

```
private ArrayList<Persona> listapersonas;

@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);

   listapersonas=new ArrayList<Persona>();
   listapersonas.add(new Persona("Juan", 'm'));
   listapersonas.add(new Persona("pedro",'m'));
   listapersonas.add(new Persona("luis",'m'));
   listapersonas.add(new Persona("ana",'f'));
   listapersonas.add(new Persona("carla",'f'));
   listapersonas.add(new Persona("maria",'f'));
   listapersonas.add(new Persona("gustavo",'m'));
   listapersonas.add(new Persona("carlos",'m'));
   listapersonas.add(new Persona("marta",'f'));
   listapersonas.add(new Persona("luisa",'f'));
   listapersonas.add(new Persona("fernanda",'f'));
   listapersonas.add(new Persona("jose",'m'));

   AdaptadorPersonas adaptador = new AdaptadorPersonas(this);
   ListView lv1 = (ListView)findViewById(R.id.list1);
   lv1.setAdapter(adaptador);

}

class AdaptadorPersonas extends ArrayAdapter<Persona> {

   AppCompatActivity appCompatActivity;

   AdaptadorPersonas(AppCompatActivity context) {
      super(context, R.layout.persona, listapersonas);
      appCompatActivity = context;
   }
```

```
    public View getView(int position, View convertView, ViewGroup parent) {
      LayoutInflater inflater = appCompatActivity.getLayoutInflater();
      View item = inflater.inflate(R.layout.persona, null);

      TextView textView1 = (TextView)item.findViewById(R.id.textView);
      textView1.setText(listapersonas.get(position).getNombre());

       ImageView imageView1 = (ImageView)item.findViewById(R.id.image-
View);
      if (listapersonas.get(position).getGenero()=='m')
        imageView1.setImageResource(R.mipmap.hombre);
      else
        imageView1.setImageResource(R.mipmap.mujer);
      return(item);
    }
  }
}
```

Como atributo de la clase MainActivity definimos un ArrayList cuyos elementos serán de la clase Persona (recordemos que nosotros creamos la clase Persona que tiene dos atributos: nombre y genero):

```
private ArrayList<Persona> listapersonas;
```

En el método onCreate de nuestra actividad procedemos a crear el ArrayList y cargamos un conjunto de personas, en cada llamada al método add del ArrayList procedemos a crear un objeto de la clase Persona y pasamos al constructor el nombre de la persona y su género (los valores que cargamos siempre serán los mismos pero podríamos recuperar los mismos de una base de datos):

```
listapersonas=new ArrayList<Persona>();
listapersonas.add(new Persona("Juan", 'm'));
listapersonas.add(new Persona("pedro",'m'));
listapersonas.add(new Persona("luis",'m'));
listapersonas.add(new Persona("ana",'f'));
listapersonas.add(new Persona("carla",'f'));
```

```
listapersonas.add(new Persona("maria","f'));
listapersonas.add(new Persona("gustavo","m'));
listapersonas.add(new Persona("carlos","m'));
listapersonas.add(new Persona("marta","f'));
listapersonas.add(new Persona("luisa","f'));
listapersonas.add(new Persona("fernanda","f'));
listapersonas.add(new Persona("jose","m'));
```

Creamos un objeto de la clase AdaptadorPersonas que veremos un poco más abajo:

AdaptadorPersonas adaptador = new AdaptadorPersonas(this);

Obtenemos la referencia del ListView:

ListView lv1 = (ListView)findViewById(R.id.list1);

Informamos al lv1 que los datos se recuperarán mediante el objeto llamado adaptador de la clase AdaptadorPersonas:

lv1.setAdapter(adaptador);

La clase AdaptadorPersonas debe heredar de la clase ArrayAdapter:

class AdaptadorPersonas extends ArrayAdapter {

En el constructor llega como parámetro la referencia del ActionBarActivity que contiene el ListView (definimos un atributo para almacenar dicha referencia), también pasamos al constructor de la clase padre mediante el comando super la referencia del ActionBarActivity y el archivo XML asociado a cada item que lo llamamos persona y finalmente el ArrayList respectivo:

```
class AdaptadorPersonas extends ArrayAdapter<Persona> {

    AppCompatActivity appCompatActivity;
```

```
AdaptadorPersonas(AppCompatActivity context) {
    super(context, R.layout.persona, listapersonas);
    appCompatActivity = context;
}
```

El método getView se ejecuta para cada item que debe mostrar el ListView y es en donde podemos especificar que debe mostrar cada ImageView y cada ListView (el atributo position de este método es útil para poder rescatar de la lista listapersonas cual es la que se debe mostrar y según el género de la misma mostrar una imagen específica en el ImageView):

```
public View getView(int position, View convertView, ViewGroup parent) {
    LayoutInflater inflater = appCompatActivity.getLayoutInflater();
    View item = inflater.inflate(R.layout.persona, null);

    TextView textView1 = (TextView)item.findViewById(R.id.textView);
    textView1.setText(listapersonas.get(position).getNombre());

     ImageView imageView1 = (ImageView)item.findViewById(R.id.image-
View);
    if (listapersonas.get(position).getGenero()=='m')
      imageView1.setImageResource(R.mipmap.hombre);
    else
      imageView1.setImageResource(R.mipmap.mujer);
    return(item);
}
```

6. Por último si queremos hacer un proceso cuando se selecciona algún item del ListView debemos agregar al método onCreate de la actividad lo siguiente:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
```

```
listapersonas=new ArrayList<Persona>();
listapersonas.add(new Persona("Juan", 'm'));
listapersonas.add(new Persona("pedro",'m'));
listapersonas.add(new Persona("luis",'m'));
listapersonas.add(new Persona("ana",'f'));
listapersonas.add(new Persona("carla",'f'));
listapersonas.add(new Persona("maria",'f'));
listapersonas.add(new Persona("gustavo",'m'));
listapersonas.add(new Persona("carlos",'m'));
listapersonas.add(new Persona("marta",'f'));
listapersonas.add(new Persona("luisa",'f'));
listapersonas.add(new Persona("fernanda",'f'));
listapersonas.add(new Persona("jose",'m'));
AdaptadorPersonas adaptador = new AdaptadorPersonas(this);
ListView lv1 = (ListView)findViewById(R.id.listView);
lv1.setAdapter(adaptador);
lv1.setOnItemClickListener(new AdapterView.OnItemClickListener() {
@Override
public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
Toast.makeText(MainActivity.this,listapersonas.get(i).getNombre(), Toast.LENG-
TH_LONG).show();
}
});
}
```

Solución problema 1.8.2.1.

Crear un proyecto

La interfaz visual de la aplicación cuando la ejecutemos será parecida a esta:

El archivo XML 'activity_main.xml' es:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:id="@+id/activity_main"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="com.tutorialesprogramacionya.proyecto053.MainActivity">

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Agregar"
      android:id="@+id/button"
      android:onClick="agregar"
      android:layout_below="@+id/editText"
      android:layout_alignParentLeft="true"
```

```
    android:layout_alignParentStart="true" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Hacer una presión larga sobre el telefono que queremos borrar"
    android:id="@+id/textView"
    android:layout_alignTop="@+id/button"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"
    android:layout_toRightOf="@+id/button"
    android:layout_toEndOf="@+id/button" />

  <ListView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/listView"
    android:layout_below="@+id/button"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />

  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"
    android:hint="Nombre y Telefono" />
</RelativeLayout>
```

Como podemos ver hemos fijado el valor de la propiedad onClick del

botón “agregar”:

android:onClick=”agregar”

El archivo ‘MainActivity.java’ queda codificado de la siguiente manera:

```
package com.tutorialesprogramacionya.proyecto053;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.ListView;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {

   private ArrayList<String> telefonos;
   private ArrayAdapter<String> adaptador1;
   private ListView lv1;
   private EditText et1;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      telefonos=new ArrayList<String>();
      telefonos.add("marcos : 43734843");
      telefonos.add("luis : 6554343");
      telefonos.add("ana : 7445434");
      adaptador1=new ArrayAdapter<String>(this,android.R.layout.simple_list_
item_1,telefonos);
```

```
    lv1=(ListView)findViewById(R.id.listView);
    lv1.setAdapter(adaptador1);

    et1=(EditText)findViewById(R.id.editText);

     lv1.setOnItemLongClickListener(new AdapterView.OnItemLongClickListe-
ner() {
       @Override
      public boolean onItemLongClick(AdapterView<?> adapterView, View view,
int i, long l) {
          final int posicion=i;

           AlertDialog.Builder dialogo1 = new AlertDialog.Builder(MainActivity.
this);
          dialogo1.setTitle("Importante");
          dialogo1.setMessage("¿ Elimina este teléfono ?");
          dialogo1.setCancelable(false);
            dialogo1.setPositiveButton("Confirmar", new DialogInterface.OnClic-
kListener() {
              public void onClick(DialogInterface dialogo1, int id) {
                telefonos.remove(posicion);
                adaptador1.notifyDataSetChanged();
              }
          });
             dialogo1.setNegativeButton("Cancelar", new DialogInterface.OnClic-
kListener() {
              public void onClick(DialogInterface dialogo1, int id) {
              }
          });
          dialogo1.show();

          return false;
        }
    });
  }
```

```
  public void agregar(View v) {
    telefonos.add(et1.getText().toString());
    adaptador1.notifyDataSetChanged();
    et1.setText("");
  }
}
```

Definimos como atributos el ArrayList que almacena en memoria la lista de teléfonos, el ArrayAdapter que cumple como función ser el puente entre el ArrayList y el ListView:

```
private ArrayList<String> telefonos;
private ArrayAdapter<String> adaptador1;
private ListView lv1;
private EditText et1;
```

Creamos el ArrayList y guardamos por defecto los datos de tres personas para que el ListView aparezca con dichos elementos inicialmente:

```
telefonos=new ArrayList<String>();
telefonos.add("marcos : 43734843");
telefonos.add("luis : 6554343");
telefonos.add("ana : 7445434");
```

Creamos el ArrayAdapter y utilizamos el layout más sencillo que es el que muestra solo un TextView en cada Item y lo indicamos mediante el valor android.R.layout.simple_list_item_1:

```
adaptador1=new ArrayAdapter<String>(this,android.R.layout.simple_
list_item_1,telefonos);
```

Obtenemos la referencia del ListView y le asociamos el ArrayAdapter para que pueda recibir los items a mostrar:

```
lv1=(ListView)findViewById(R.id.listView);
lv1.setAdapter(adaptador1);
```

Cuando se presiona el botón agregar procedemos a añadir un elemento al ArrayList y llamar seguidamente al método notifyDataSetChanged() del ArrayAdapter para que informe al ListView que actualice los datos en pantalla:

```
  public void agregar(View v) {
    telefonos.add(et1.getText().toString());
```

```
    adaptador1.notifyDataSetChanged();
    et1.setText("");
  }
```

Cuando se presiona un tiempo prolongado con el dedo un item del ListView se dispara el método onItemLongClick donde procedemos a mostrar un diálogo para que confirme o cancele la eliminación del teléfono seleccionado:

```
    lv1.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
      @Override
      public boolean onItemLongClick(AdapterView<?> adapterView, View view, int i, long l) {
        final int posicion=i;

        AlertDialog.Builder dialogo1 = new AlertDialog.Builder(MainActivity.this);
        dialogo1.setTitle("Importante");
        dialogo1.setMessage("¿ Elimina este teléfono ?");
        dialogo1.setCancelable(false);
        dialogo1.setPositiveButton("Confirmar", new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialogo1, int id) {
            telefonos.remove(posicion);
            adaptador1.notifyDataSetChanged();
          }
        });
        dialogo1.setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialogo1, int id) {
          }
        });
        dialogo1.show();

        return false;
      }
```

```
    });
  }
```

Como podemos ver cuando se presiona la opción “Confirmar” del diálogo procedemos a eliminar el elemento del ArrayList y pedir que se refresque la pantalla mediante la llamada del método notifyDataSetChanged del ArrayAdapter:

```
telefonos.remove(posicion);
adaptador1.notifyDataSetChanged();
```

Solución problema 1.8.3.1.

Crear un proyecto

La interfaz visual de la aplicación cuando la ejecutemos será parecida a esta:

El archivo XML ‘activity_main.xml’ es:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
```

```
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context="com.tutorialesprogramacionya.proyecto054.MainActivity">

<EditText
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:inputType="textPersonName"
   android:ems="10"
   android:id="@+id/editText"
   android:layout_alignParentTop="true"
   android:layout_alignParentLeft="true"
   android:layout_alignParentStart="true"
   android:layout_alignParentRight="true"
   android:layout_alignParentEnd="true"
   android:hint="nombre" />

<EditText
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:inputType="phone"
   android:ems="10"
   android:id="@+id/editText2"
   android:layout_below="@+id/editText"
   android:layout_alignParentLeft="true"
   android:layout_alignParentStart="true"
   android:layout_alignRight="@+id/editText"
   android:layout_alignEnd="@+id/editText"
   android:hint="teléfono" />
<Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Agregar"
   android:id="@+id/button"
   android:layout_below="@+id/editText2"
```

```
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:onClick="agregar" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceSmall"
    android:text="Presión larga sobre el teléfono a borrar"
    android:id="@+id/textView"
    android:layout_below="@+id/editText2"
    android:layout_toRightOf="@+id/button"
    android:layout_toEndOf="@+id/button" />

  <ListView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/listView"
    android:layout_below="@+id/button"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />

</RelativeLayout>
```

Hemos iniciado la propiedad onClick con el nombre del método a ejecutar al ser presionado:

```
android:onClick=”agregar” />
```

Por otro lado hemos fijado con “true” la propiedad focusableInTouchMode del RelativeLayout para que no aparezca con foco el primer EditText:

```
android:focusableInTouchMode=”true”>
```

Ahora pasemos al archivo MainActivity.java

```
package com.tutorialesprogramacionya.proyecto054;

import android.app.AlertDialog;
```

```
import android.content.Context;
import android.content.DialogInterface;
import android.content.SharedPreferences;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.ListView;

import java.util.ArrayList;
import java.util.Map;
import java.util.StringTokenizer;

public class MainActivity extends AppCompatActivity {
   private ArrayList<String> datos;
   private ArrayAdapter<String> adaptador1;
   private ListView lv1;
   private EditText et1,et2;
   private SharedPreferences prefe1;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      datos =new ArrayList<String>();
      leerSharedPreferences();
        adaptador1=new ArrayAdapter<String>(this,android.R.layout.simple_list_
item_1, datos);
      lv1=(ListView)findViewById(R.id.listView);
      lv1.setAdapter(adaptador1);

      et1=(EditText)findViewById(R.id.editText);
      et2=(EditText)findViewById(R.id.editText2);
```

```
        lv1.setOnItemLongClickListener(new AdapterView.OnItemLongClickListe-
ner() {
          @Override
        public boolean onItemLongClick(AdapterView<?> adapterView, View view,
int i, long l) {
            final int posicion=i;


             AlertDialog.Builder dialogo1 = new AlertDialog.Builder(MainActivity.
this);
            dialogo1.setTitle("Importante");
            dialogo1.setMessage("¿ Elimina este teléfono ?");
            dialogo1.setCancelable(false);
              dialogo1.setPositiveButton("Confirmar", new DialogInterface.OnClic-
kListener() {
                public void onClick(DialogInterface dialogo1, int id) {
                   String s=datos.get(posicion);
                   StringTokenizer tok1=new StringTokenizer(s,":");
                   String nom=tok1.nextToken().trim();
                   SharedPreferences.Editor elemento=prefe1.edit();
                   elemento.remove(nom);
                   elemento.commit();

                   datos.remove(posicion);
                   adaptador1.notifyDataSetChanged();
                }
            });
              dialogo1.setNegativeButton("Cancelar", new DialogInterface.OnClic-
kListener() {
                public void onClick(DialogInterface dialogo1, int id) {
                }
            });
            dialogo1.show();

            return false;
        }
```

```
        });
    }

    private void leerSharedPreferences() {
        prefe1=getSharedPreferences("datostelefonos", Context.MODE_PRIVATE);
        Map<String,?> claves = prefe1.getAll();
        for(Map.Entry<String,?> ele : claves.entrySet()){
            datos.add(ele.getKey()+" : " +ele.getValue().toString());
        }
    }
    public void agregar(View v) {
        datos.add(et1.getText().toString()+" : "+et2.getText().toString());
        adaptador1.notifyDataSetChanged();
        SharedPreferences.Editor elemento=prefe1.edit();
        elemento.putString(et1.getText().toString(),et2.getText().toString());
        elemento.commit();
        et1.setText("");
        et2.setText("");
    }
}
```

Definimos como atributos el ArrayList que almacena en memoria la lista de teléfonos y titulares llamado datos, el ArrayAdapter que cumple como función ser el puente entre el ArrayList y el ListView, el ListView que muestra los datos, los dos EditText y finalmente la variable prefe1 de la clase SharedPreferences que nos permite almacenar en forma permanente los datos y su posterior recuperación:

```
private ArrayList<String> datos;
private ArrayAdapter<String> adaptador1;
private ListView lv1;
private EditText et1,et2;
private SharedPreferences prefe1;
```

En el método onCreate creamos el ArrayList:

```
datos =new ArrayList<String>();
```

Procedemos a llamar al método leerSharedPreferences() que tiene por objetivo leer el archivo de preferencias que almacena todos las personas y sus teléfonos.

El método de lectura es:

```
private void leerSharedPreferences() {
   prefe1=getSharedPreferences("datostelefonos", Context.MODE_PRIVATE);
   Map<String,?> claves = prefe1.getAll();
   for(Map.Entry<String,?> ele : claves.entrySet()){
      datos.add(ele.getKey()+" : " +ele.getValue().toString());
   }
}
```

Llamamos al método getSharedPreferences pasando el nombre del archivo de preferencias y el modo de apertura:

```
prefe1=getSharedPreferences("datostelefonos", Context.MODE_PRIVATE);
```

Creamos un objeto de la clase Map llamando al método getAll del SharedPreferences:

```
Map<String,?> claves = prefe1.getAll();
```

Mediante un ciclo recuperamos todas las claves y sus valores del archivo de preferencias y los almacenamos en el ArrayList con el objetivo que luego se puedan visualizar en el ListView:

```
for(Map.Entry<String,?> ele : claves.entrySet()){
   datos.add(ele.getKey()+" : " +ele.getValue().toString());
}
```

Continuando en el método onCreate creamos el ArrayAdapter y utilizamos el layout más sencillo que es el que muestra solo un TextView en cada Item y lo indicamos mediante el valor android.R.layout.simple_list_item_1:

```
adaptador1=new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1, datos);
```

Obtenemos la referencia del ListView y le asociamos el ArrayAdapter para que pueda recibir los items a mostrar:

```
        lv1=(ListView)findViewById(R.id.listView);
        lv1.setAdapter(adaptador1);
```

Cuando se presiona el botón agregar procedemos a añadir un elemento al ArrayList y llamar seguidamente al método notifyDataSetChanged() del ArrayAdapter para que informe al ListView que actualice los datos en pantalla. Por otro lado en forma paralela procedemos a insertar una entrada en el archivo de preferencias indicando como clave el dato del et1 y como valor el valor del et2 :

```
public void agregar(View v) {
  datos.add(et1.getText().toString()+" : "+et2.getText().toString());
  adaptador1.notifyDataSetChanged();
  SharedPreferences.Editor elemento=prefe1.edit();
  elemento.putString(et1.getText().toString(),et2.getText().toString());
  elemento.commit();
  et1.setText("");
  et2.setText("");
}
```

Cuando se presiona un tiempo prolongado con el dedo un item del ListView se dispara el método onItemLongClick donde procedemos a mostrar un diálogo para que confirme o cancele la eliminación del teléfono seleccionado:

```
lv1.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener() {
    @Override
    public boolean onItemLongClick(AdapterView<?> adapterView, View view,
int i, long l) {
        final int posicion=i;

        AlertDialog.Builder dialogo1 = new AlertDialog.Builder(MainActivity.
this);
        dialogo1.setTitle("Importante");
        dialogo1.setMessage("¿ Elimina este teléfono ?");
        dialogo1.setCancelable(false);
        dialogo1.setPositiveButton("Confirmar", new DialogInterface.OnClic-
kListener() {
```

```
            public void onClick(DialogInterface dialogo1, int id) {
                String s=datos.get(posicion);
                StringTokenizer tok1=new StringTokenizer(s,":");
                String nom=tok1.nextToken().trim();
                SharedPreferences.Editor elemento=prefe1.edit();
                elemento.remove(nom);
                elemento.commit();

                datos.remove(posicion);
                adaptador1.notifyDataSetChanged();
            }
        });
            dialogo1.setNegativeButton("Cancelar", new DialogInterface.OnClic-
kListener() {
            public void onClick(DialogInterface dialogo1, int id) {
            }
        });
        dialogo1.show();

        return false;
    }
});
```

Como podemos ver cuando se presiona la opción "Confirmar" del diálogo procedemos a eliminar el elemento del ArrayList y pedir que se refresque la pantalla mediante la llamada del método notifyDataSetChanged del ArrayAdapter:

```
            telefonos.remove(posicion);
            adaptador1.notifyDataSetChanged();
```

Por otro lado debemos eliminar el dato del archivo de preferencias, creamos un objeto de la clase StringTokenizer con el objetivo de dividir el nombre y el teléfono que están separados por el caracter de ":" (con la primer llamada a nextToken obtenemos el nombre del titular del teléfono, llamamos al método trim para eliminar el espacio en blanco que hay al final del nombre del titular):

```
            String s=datos.get(posicion);
            StringTokenizer tok1=new StringTokenizer(s,":");
```

```
String nom=tok1.nextToken().trim();
SharedPreferences.Editor elemento=prefe1.edit();
elemento.remove(nom);
elemento.commit();
```

Solución problema 1.9.1.1.

Para generar notificaciones en la barra de estado del sistema vamos a utilizar una clase incluida en la librería de compatibilidad *android-support-v4.jar* que ya hemos utilizado en otras ocasiones y que debe estar incluida por defecto en vuestro proyecto si lo habéis creado con alguna versión reciente del plugin de Eclipse. Esto es así para asegurar la compatibilidad con versiones de Android antiguas, ya que las notificaciones son un elemento que han sufrido bastantes cambios en las versiones más recientes. La clase en cuestión se llama NotificationCompat.Builder y lo que tendremos que hacer será crear un nuevo objeto de este tipo pasándole el contexto de la aplicación y asignar todas las propiedades que queramos mediante sus métodos *set()*.

En primer lugar estableceremos los iconos a mostrar mediante los métodos setSmallIcon() y setLargeIcon() que se corresponden con los iconos mostrados a la derecha y a la izquierda del contenido de la notificación en versiones recientes de Android. En versiones más antiguas tan sólo se mostrará el icono pequeño a la izquierda de la notificación. Además, el icono pequeño también se mostrará en la barra de estado superior.

A continuación estableceremos el título y el texto de la notificación, utilizando para ello los métodos setContentTitle() y setContentText().

Por último, estableceremos el *ticker* (texto que aparece por unos segundos en la barra de estado al generarse una nueva notificación) mediante setTicker() y el texto auxiliar (opcional) que aparecerá a la izquierda del icono pequeño de la notificación mediante setContentInfo().

La fecha/hora asociada a nuestra notificación se tomará automáticamente de la fecha/hora actual si no se establece nada, o bien puede utilizarse el método setWhen() para indicar otra marca de tiempo. Veamos cómo quedaría nuestro código por el momento:

```
1  NotificationCompat.Builder mBuilder =
2  new NotificationCompat.Builder(MainActivity.this)
3  .setSmallIcon(android.R.drawable.stat_sys_warning)
```

```
4       .setLargeIcon((((BitmapDrawable)getResources()
5       .getDrawable(R.drawable.ic_launcher)).getBit-
        map()))
6       .setContentTitle("Mensaje de Alerta")
7       .setContentText("Ejemplo de notificación.")
8       .setContentInfo("4")
9       .setTicker("Alerta!");
```

El segundo paso será establecer la actividad a la cual debemos dirigir al usuario automáticamente si éste pulsa sobre la notificación. Para ello debemos construir un objeto PendingIntent, que será el que contenga la información de la actividad asociada a la notificación y que será lanzado al pulsar sobre ella. Para ello definiremos en primer lugar un objeto Intent, indicando la clase de la actividad concreta a lanzar, que en nuestro caso será la propia actividad principal de ejemplo (MainActivity.class). Este *intent* lo utilizaremos para construir el PendingIntent final mediante el método PendingIntent.getActivity(). Por último asociaremos este objeto a la notificación mediante el método setContentIntent() del Builder. Veamos cómo quedaría esta última parte comentada:

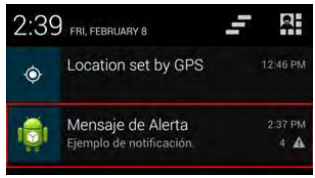
```
1       Intent notIntent =
2       new Intent(MainActivity.this, Main-
        Activity.class);
3
4       PendingIntent contIntent =
5       PendingIntent.getActivity(
6       MainActivity.this, 0, notIntent, 0);
7
8       mBuilder.setContentIntent(contIntent);
```

Por último, una vez tenemos completamente configuradas las opciones de nuestra notificación podemos generarla llamando al método notify() del *Notification Manager*,al cual podemos acceder mediante una llamada a getSystemService() con la constante Context.NOTIFICATION_SERVICE. Por su parte al método notify() le pasaremos como parámetro un identificador único definido por nosotros que identifique nuestra notificación y el resultado del builder que hemos construido antes, que obtenemos llamando a su método build().

```
1   NotificationManager mNotificationManager =
2       (NotificationManager) getSystemService(-
        Context.NOTIFICATION_SERVICE);
3
4   mNotificationManager.notify(NOTIF_ALER-
        TA_ID, mBuilder.build());
```

Ya estamos en disposición de probar nuestra aplicación de ejemplo. Para ello vamos a ejecutarla en el emulador de Android y pulsamos el botón que hemos implementado con todo el código anterior. Si todo va bien, debería aparecer en ese mismo momento nuestra notificación en la barra de estado, con el icono y texto definidos.

Si ahora salimos de la aplicación y desplegamos la bandeja del sistema podremos verificar el resto de información de la notificación tal como muestra la siguiente imagen:



Por último, si pulsamos sobre la notificación se debería abrir de nuevo automáticamente la aplicación de ejemplo.

En definitiva, como podéis comprobar es bastante sencillo generar notificaciones en la barra de estado de Android desde nuestras aplicaciones. Os animo a utilizar este mecanismo para notificar determinados eventos al usuario de forma bastante visual e intuitiva.

For all of the versions

```
NotificationManager notificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
        String NOTIFICATION_CHANNEL_ID = "my_channel_id_01";

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            NotificationChannel notificationChannel = new NotificationChan-
```

```
nel(NOTIFICATION_CHANNEL_ID, "My Notifications", NotificationManager.
IMPORTANCE_MAX);

            // Configure the notification channel.
            notificationChannel.setDescription("Channel description");
            notificationChannel.enableLights(true);
            notificationChannel.setLightColor(Color.RED);
                notificationChannel.setVibrationPattern(new long[]{0, 1000, 500,
1000});
            notificationChannel.enableVibration(true);
            notificationManager.createNotificationChannel(notificationChannel);
        }


        NotificationCompat.Builder notificationBuilder = new NotificationCom-
pat.Builder(MainActivity.this, NOTIFICATION_CHANNEL_ID);

         notificationBuilder.setAutoCancel(true)
              .setDefaults(Notification.DEFAULT_ALL)
              .setWhen(System.currentTimeMillis())
              .setSmallIcon(R.mipmap.ic_launcher)
              .setTicker("Hearty365")
              //     .setPriority(Notification.PRIORITY_MAX)
              .setContentTitle("Default notification")
             .setContentText("Lorem ipsum dolor sit amet, consectetur adipiscing
elit.")
              .setContentInfo("Info");

       notificationManager.notify(/notification id/1, notificationBuilder.build());


package com.juan.notificacion;

import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.NotificationCompat;
```

```
import android.app.Notification;
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.Color;
import android.graphics.drawable.BitmapDrawable;
import android.os.Build;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

 private static final int NOTIF_ALERTA_ID=1;

 private Button btnNotifcation;

 @Override
   protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   btnNotifcation=(Button)findViewById(R.id.btnNotifcation);


    btnNotifcation.setOnClickListener(new View.OnClickListener() {
     @Override
       public void onClick(View view) {

NotificationManager notificationManager = (NotificationManager) getSystem-
Service(Context.NOTIFICATION_SERVICE);
 String NOTIFICATION_CHANNEL_ID = "my_channel_id_01";

 if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
 NotificationChannel notificationChannel = new NotificationChannel(NOTIFI-
CATION_CHANNEL_ID, "My Notifications", NotificationManager.IMPOR-
TANCE_MAX);
```

```
// Configure the notification channel.
        notificationChannel.setDescription("Channel description");
        notificationChannel.enableLights(true);
        notificationChannel.setLightColor(Color.RED);
        notificationChannel.setVibrationPattern(new long[]{0, 1000, 500, 1000});
        notificationChannel.enableVibration(true);
        notificationManager.createNotificationChannel(notificationChannel);
}


 NotificationCompat.Builder notificationBuilder = new NotificationCompat.
Builder(MainActivity.this, NOTIFICATION_CHANNEL_ID);

        notificationBuilder.setAutoCancel(true)
        .setDefaults(Notification.DEFAULT_ALL)
        .setWhen(System.currentTimeMillis())
        .setSmallIcon(R.mipmap.ic_launcher)
        .setTicker("Hearty365")
// .setPriority(Notification.PRIORITY_MAX)
        .setContentTitle("Default notification")
        .setContentText("Lorem ipsum dolor sit amet, consectetur adipiscing
elit.")
        .setContentInfo("Info");

 notificationManager.notify(/*notification id*/1, notificationBuilder.build());
/*
 NotificationCompat.Builder mBuilder =
   new NotificationCompat.Builder(MainActivity.this)
        .setSmallIcon(android.R.drawable.stat_sys_warning)
        .setLargeIcon((((BitmapDrawable)getResources()
            .getDrawable(R.mipmap.ic_launcher)).getBitmap()))
        .setContentTitle("Mensaje de Alerta")
        .setContentText("Ejemplo de notificación.")
        .setContentInfo("4")
        .setTicker("Alerta!");

 Intent notIntent =
        new Intent(MainActivity.this, MainActivity.class);

 PendingIntent contIntent =
```

```
        PendingIntent.getActivity(
                MainActivity.this, 0, notIntent, 0);

        mBuilder.setContentIntent(contIntent);

        NotificationManager mNotificationManager =
         (NotificationManager) getSystemService(Context.NOTIFICATION_SERVI-
CE);

        mNotificationManager.notify(NOTIF_ALERTA_ID, mBuilder.build());
 */
    }
  });
}

/*public void LlamarNotification(View button){

        }*/
}
```

Solución problema 1.9.2.1.

El uso actual de los diálogos en Android se basa en fragmets, pero por suerte tenemos toda la funcionalidad implementada una vez más en la librería de compatibilidad *android-support-v4.jar* (que debe estar incluida por defecto en tu proyecto si lo has creado con una versión reciente del pluin de Eclipse) por lo que no tendremos problemas al ejecutar nuestra aplicación en versiones antiguas de Android. En este caso nos vamos a basar en la clase DialogFragment. Para crear un diálogo lo primero que haremos será crear una nueva clase que herede de DialogFragment y sobrescribiremos uno de sus métodos onCreateDialog(), que será el encargado de construir el diálogo con las opciones que necesitemos.

La forma de construir cada diálogo dependerá de la información y funcionalidad que necesitemos. A continuación mostraré algunas de las formas más habituales.

Diálogo de Alerta

Este tipo de diálogo se limita a mostrar un mensaje sencillo al usuario, y un único botón de OK para confirmar su lectura. Lo construiremos mediante la clase AlertDialog, y más concretamente su subclase AlertDialog.Builder, de forma similar a las notificaciones de barra de estado que ya hemos comenta-

do en el capítulo anterior. Su utilización es muy sencilla, bastará con crear un objeto de tipo AlertDialog.Builder y establecer las propiedades del diálogo mediante sus métodos correspondientes: título [setTitle()], mensaje [setMessage()] y el texto y comportamiento del botón [setPositiveButton()]. Veamos un ejemplo:

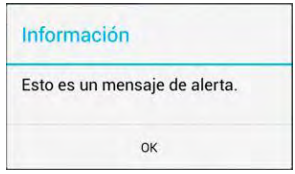
```
1.  public class DialogoAlerta extends DialogFragment {
2.      @Override
3.      public Dialog onCreateDialog(Bundle savedInstanceState) {
4.
5.          AlertDialog.Builder builder =
6.                  new AlertDialog.Builder(getActivity());
7.
8.          builder.setMessage("Esto es un mensaje de alerta.")
9.                  .setTitle("Información")
10.                 .setPositiveButton("OK", new DialogInterface.OnClickListener() {
11.                     public void onClick(DialogInterface dialog, int id) {
12.                         dialog.cancel();
13.                     }
14.                 });
15.
16.         return builder.create();
17.     }
18. }
```

Como vemos, al método setPositiveButton() le pasamos como argumentos el texto a mostrar en el botón, y la implementación del evento onClick en forma de objeto OnClickListener. Dentro de este evento, nos limitamos a cerrar el diálogo mediante su método cancel(), aunque podríamos realizar cualquier otra acción.

Para lanzar este diálogo por ejemplo desde nuestra actividad principal, obtendríamos una referencia al *Fragment Manager* mediante una llamada a getSupportFragmentManager(), creamos un nuevo objeto de tipo DialogoAlerta y por último mostramos el diálogo mediante el método show() pasándole la referencia al fragment manager y una etiqueta identificativa del diálogo.

```
1.  btnAlerta.setOnClickListener(new View.OnClickListener() {
2.    public void onClick(View v) {
3.      FragmentManager fragmentManager = getSupportFragmentManager();
4.        DialogoAlerta dialogo = new DialogoAlerta();
5.      dialogo.show(fragmentManager, "tagAlerta");
6.    }
7.  });
```

El aspecto de nuestro diálogo de alerta sería el siguiente:



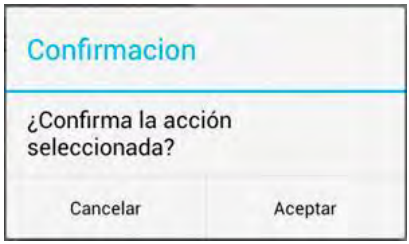
Diálogo de Confirmación

Un diálogo de confirmación es muy similar al anterior, con la diferencia de que lo utilizaremos para solicitar al usuario que nos confirme una determinada acción, por lo que las posibles respuestas serán del tipo Sí/No.

La implementación de estos diálogos será prácticamente igual a la ya comentada para las alertas, salvo que en esta ocasión añadiremos dos botones, uno de ellos para la respuesta afirmativa (setPositiveButton()), y el segundo para la respuesta negativa (setNegativeButton()). Veamos un ejemplo:

```
1.  public class DialogoSeleccion extends DialogFragment {
2.    @Override
3.    public Dialog onCreateDialog(Bundle savedInstanceState) {
4.
5.     final String[] items = {"Español", "Inglés", "Francés"};
6.
7.      AlertDialog.Builder builder =
8.            new AlertDialog.Builder(getActivity());
9.
10.     builder.setTitle("Selección")
11.       .setItems(items, new DialogInterface.OnClickListener() {
12.           public void onClick(DialogInterface dialog, int item) {
```

```
13.                 Log.i("Dialogos", "Opción elegida: " + items[item]);
14.             }
15.         });
16.
17.     return builder.create();
18.   }
19. }
```



Diálogo de Selección

Cuando las opciones a seleccionar por el usuario no son sólo dos, como en los diálogos de confirmación, sino que el conjunto es mayor podemos utilizar los diálogos de selección para mostrar una lista de opciones entre las que el usuario pueda elegir.

Para ello también utilizaremos la clase AlertDialog, pero esta vez no asignaremos ningún mensaje ni definiremos las acciones a realizar por cada botón individual, sino que directamente indicaremos la lista de opciones a mostrar (mediante el método setItems()) y proporcionaremos la implementación del evento onClick() sobre dicha lista (mediante un listener de tipo DialogInterface.OnClickListener), evento en el que realizaremos las acciones oportunas según la opción elegida. La lista de opciones la definiremos como un array tradicional. Veamos cómo:

```
1.  public class DialogoSeleccion extends DialogFragment {
2.    @Override
3.    public Dialog onCreateDialog(Bundle savedInstanceState) {
4.
5.     final String[] items = {“Español”, “Inglés”, “Francés”};
6.
7.       AlertDialog.Builder builder =
8.             new AlertDialog.Builder(getActivity());
```

```
9.
10.         builder.setTitle("Selección")
11.           .setItems(items, new DialogInterface.OnClickListener() {
12.               public void onClick(DialogInterface dialog, int item) {
13.                   Log.i("Dialogos", "Opción elegida: " + items[item]);
14.               }
15.           });
16.
17.         return builder.create();
18.     }
19. }
```

En este caso el diálogo tendrá un aspecto similar a la interfaz mostrada para los controles Spinner.

Este diálogo permite al usuario elegir entre las opciones disponibles cada vez que se muestra en pantalla. Pero, ¿y si quisiéramos recordar cuál es la opción u opciones seleccionadas por el usuario para que aparezcan marcadas al visualizar de nuevo el cuadro de diálogo? Para ello podemos utilizar los métodos setSingleChoiceItems() o setMultiChiceItems(), en vez de el setItems() utilizado anteriormente. La diferencia entre ambos métodos, tal como se puede suponer por su nombre, es que el primero de ellos permitirá una selección simple y el segundo una selección múltiple, es decir, de varias opciones al mismo tiempo, mediante controles CheckBox.

La forma de utilizarlos es muy similar a la ya comentada. En el caso de setSingleChoiceItems(), el método tan sólo se diferencia de setItems() en que recibe un segundo parámetro adicional que indica el índice de la opción marcada por defecto. Si no queremos tener ninguna de ellas marcadas inicialmente pasaremos el valor -1.

```
1.  builder.setTitle("Selección")
2.      .setSingleChoiceItems(items, -1,
3.              new DialogInterface.OnClickListener() {
4.          public void onClick(DialogInterface dialog, int item) {
5.              Log.i("Dialogos", "Opción elegida: " + items[item]);
6.          }
7.      });
```

De esta forma conseguiríamos un diálogo como el de la siguiente imagen:

Si por el contrario optamos por la opción de selección múltiple, la diferencia principal estará en que tendremos que implementar un listener del tipo DialogInterface.OnMultiChoiceClickListener. En este caso, en el evento onClick recibiremos tanto la opción seleccionada (item) como el estado en el que ha quedado (isChecked). Además, en esta ocasión, el segundo parámetro adicional que indica el estado por defecto de las opciones ya no será un simple número entero, sino que tendrá que ser un array de booleanos. En caso de no querer ninguna opción seleccionada por defecto pasaremos el valor null.

```
1.  builder.setTitle("Selección")
2.      .setMultiChoiceItems(items, null,
3.          new DialogInterface.OnMultiChoiceClickListener() {
4.          public void onClick(DialogInterface dialog, int item, boolean isChecked)
    {
5.              Log.i("Dialogos", "Opción elegida: " + items[item]);
6.          }
```

```
7.  });
```

Y el diálogo nos quedaría de la siguiente forma:

Tanto si utilizamos la opción de selección simple como la de selección múltiple, para salir del diálogo tendremos que pulsar la tecla «*Atrás*» de nuestro dispositivo.

Diálogos Personalizados

Por último, vamos a comentar cómo podemos establecer completamente el aspecto de un cuadro de diálogo. Para esto vamos a actuar como si estuviéramos definiendo la interfaz de una actividad, es decir, definiremos un layout XML con los elementos a mostrar en el diálogo. En mi caso voy a definir un layout de ejemplo llamado dialog_personal.xml que colocaré como siempre en la carpeta res/layout. Contendrá por ejemplo una imagen a la izquierda y dos líneas de texto a la derecha:

```
1   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     android:layout_width="match_parent"
3     android:layout_height="match_parent"
4     android:orientation="horizontal"
5     android:padding="3dp" >
6
7     <ImageView
8       android:id="@+id/imageView1"
9       android:layout_width="wrap_content"
10      android:layout_height="wrap_content"
11      android:src="@drawable/ic_launcher" />
12
13    <LinearLayout
14      android:layout_width="wrap_content"
```

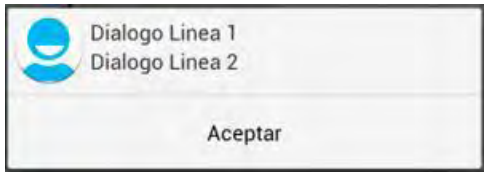
```
15      android:layout_height="match_parent"
16      android:orientation="vertical"
17      android:padding="3dp">
18
19      <TextView
20        android:id="@+id/textView1"
21        android:layout_width="wrap_content"
22        android:layout_height="wrap_content"
23        android:text="@string/dialogo_linea_1" />
24
25      <TextView
26        android:id="@+id/textView2"
27        android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@string/dialogo_linea_2" />
    </LinearLayout>
</LinearLayout>
```

Por su parte, en el método onCreateDialog() correspondiente utilizaremos el método setView() del builder para asociarle nuestro layout personalizado, que previamente tendremos que inflar como ya hemos comentado otras veces utilizando el método inflate(). Finalmente podremos incluir botones tal como vimos para los diálogos de alerta o confirmación. En este caso de ejemplo incluiremos un botón de Aceptar.

```
1   @Override
2    public Dialog onCreateDialog(Bundle savedInstanceState) {
3
4    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
5    LayoutInflater inflater = getActivity().getLayoutInflater();
6
7    builder.setView(inflater.inflate(R.layout.dialog_personal, null))
8      .setPositiveButton("Aceptar", new DialogInterface.OnClickListener() {
9        public void onClick(DialogInterface dialog, int id) {
10           dialog.cancel();
11       }
```

```
12      });
13
14      return builder.create();
15      }
16  }
```

De esta forma, si ejecutamos de nuevo nuestra aplicación de ejemplo y lanzamos el diálogo personalizado veremos algo como lo siguiente:



Y con esto terminamos con el apartado dedicado a notificaciones en Android. Hemos comentado los tres principales mecanismos de Android a la hora de mostrar mensajes y notificaciones al usuario (Toast, Barra de Estado, y Diálogos), todos ellos muy útiles para cualquier tipo de aplicación y que es importante conocer bien para poder decidir entre ellos dependiendo de las necesidades que tengamos.

Solución problema 1.9.3.1.

Según las especificaciones del componente dentro de Material Design, un snackbar debe mostrar mensajes cortos, debe aparecer desde la parte inferior de la pantalla (con la misma elevación que el *floating action button* si existiera, pero menos que los diálogos o el *navigation drawer*), no debe mostrarse más de uno al mismo tiempo, puede contener un botón de texto para realizar una acción, y normalmente puede descartarse deslizándolo hacia un lateral de la pantalla.

Salvo el último punto, que aclararemos más adelante, el nuevo componente Snackbar de la librería de diseño se encargará de asegurar todos los demás, y además mantendrá la facilidad de uso de los toast, siendo su API muy similar.

Entrando ya en detalles, lo primero que tendremos que hacer para utilizar los snackbar será añadir la referencia a la nueva librería de diseño a nuestro proyecto:

```
dependencies {
    //...
    compile 'com.android.support:design:22.2.0'
}
```

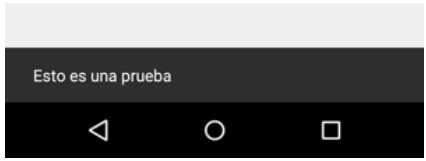
Hecho esto, para mostrar un snackbar procederemos de una forma muy similar a cómo lo hacíamos en el caso de los toast. Construiremos el snackbar mediante el método estático Snackbar.make() y posteriormente lo mostraremos llamando al método show(). A modo de ejemplo haremos esto desde el evento click de un botón añadido a la aplicación.

En este caso el método make() recibe 3 parámetros. El segundo y el tercero son equivalentes a los ya mostrados para los toast, es decir, el texto a mostrar en el snackbar y la duración del mensaje en pantalla (que en este caso podrá ser Snackbar.LENGTH_SHORT o Snackbar.LENGTH_LONG). El primero requiere más explicación. Como primer parámetro debemos pasar la referencia a una vista que permita al snackbar (navegando hacia arriba por la jerarquía de vistas de nuestro layout) descubrir un «*contenedor adecuado*» donde alojarse. Este contenedor será normalmente el *content view* o vista raíz de la actividad, o un contenedor de tipo CoordinatorLayout si se encuentra antes (al final del artículo veremos qué es esto último).

Por tanto, en teoría podríamos pasar en este primer parámetro casi cualquier vista de nuestra actividad (aún no he podido hacer pruebas con layouts complejos). En nuestro caso de ejemplo, aprovechando que creamos el snackbar dentro del evento onClick de un botón pasaremos la referencia al view del botón pulsado que recibimos en el evento:

```
btnSnackbarSimple.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        Snackbar.make(view, "Esto es una prueba", Snackbar.LENGTH_LONG)
                .show();
    }
});
```

Esto es todo lo que hace falta para mostrar un snackbar sencillo, que aparecería en pantalla de la siguiente forma:



Vamos a completar un poco más el ejemplo mostrando también un snackbar con una acción (lanzado desde un segundo botón). Para añadir una acción al snackbar utilizaremos su método addAction(), al que pasaremos como parámetros el texto del botón de acción, y un listener para su evento onClick (análogo al de un botón normal) donde podremos responder a la pulsación del botón realizando las acciones oportunas (en mi caso una simple un simple mensaje de log):

```
1  Snackbar.make(view, "Esto es otra prueba", Snackbar.LENGTH_LONG)
2          .setAction("Acción", new View.OnClickListener() {
3              @Override
4              public void onClick(View view) {
5                  Log.i("Snackbar", "Pulsada acción snackbar!");
6              }
7          })
8          .show();
```

El botón de acción se mostrará por defecto con el *accent color* del tema definido para la aplicación (si se ha definido dicho color) o con el color de selección actual. Sin embargo, también podemos especificarlo en la construcción del snackbar mediante el método setActionTextColor(), pasándole directamente un color, o recuperando alguno de los definidos en los recursos de nuestra aplicación mediante getResources().getColor(id_recurso_color):

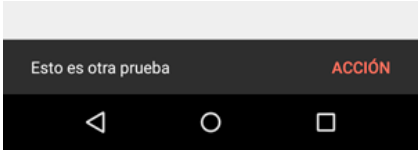
```
1  Snackbar.make(view, "Esto es otra prueba", Snackbar.LENGTH_LONG)
2          //.setActionTextColor(Color.CYAN)
3          .setActionTextColor(getResources().getColor(R.color.snackbar_action))
4          .setAction("Acción", new View.OnClickListener() {
5              @Override
6              public void onClick(View view) {
7                  Log.i("Snackbar", "Pulsada acción snackbar!");
```

```
8               }
9           })
10          .show();
```

En el código anterior establecemos un color que hemos definido en el fichero */res/values/colors.xml* de la siguiente forma:

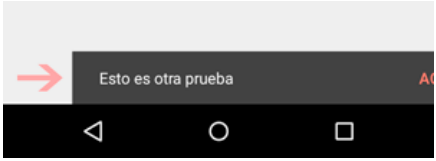
```
1   <resources>
2     <color name="snackbar_action">#ffff7663</color>
3   </resources>
```

Y con esto, nuestro nuevo snackbar con acción incluída quedaría de la siguiente forma:



Por último, vamos a pasar *de puntillas* (por el momento) sobre la posibilidad de descartar un snackbar con el gesto de deslizamiento hacia la derecha. Si intentamos hacer esto sobre cualquiera de los dos ejemplos mostrados hasta ahora veremos que no es posible. Pero si recordáis, antes mencionamos algo de un tipo de contenedor llamado CoordinatorLayout. Este nuevo componente forma también parte de la nueva librería de diseño y puede utilizarse para gestionar de una forma semiautomática algunas de las animaciones habituales de una aplicación android actual, y lo que es más importante, teniendo presente cómo la animación de algún elemento puede afectar a otros. No entraré en mucho detalle por ahora ya que tengo previsto un artículo exclusivo para este tema. Por el momento vamos a conformarnos con saber que si añadimos como elemento padre de nuestro layout un CoordinatorLayout, nuestros snackbar podrán automáticamente descartarse deslizándolos a la derecha. Nuestro layout podría quedar por ejemplo de la siguiente forma:

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingLeft="@dimen/activity_horizontal_margin"
        android:paddingRight="@dimen/activity_horizontal_margin"
        android:paddingTop="@dimen/activity_vertical_margin"
        android:paddingBottom="@dimen/activity_vertical_margin"
        android:orientation="vertical">

        <Button android:id="@+id/BtnSimple"
            android:text="@string/snackbar_simple"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />

        <Button android:id="@+id/BtnAccion"
            android:text="@string/snackbar_accion"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />

    </LinearLayout>

</android.support.design.widget.CoordinatorLayout>
```



MainActivity

```
package com.journaldev.snackbar;

import android.graphics.Color;
import android.os.Bundle;
import android.support.design.widget.CoordinatorLayout;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    CoordinatorLayout coordinatorLayout;
    private Button one, two, three;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "FloatingActionButton is clicked", Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
            }
        });
```

```
coordinatorLayout = (CoordinatorLayout) findViewById(R.id.coordinatorLayout);

    View layout= findViewById(R.id.layout);

    one=(Button)layout.findViewById(R.id.button);
    two=(Button)layout.findViewById(R.id.button2);
    three=(Button)layout.findViewById(R.id.button3);

    one.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Snackbar snackbar = Snackbar
            .make(coordinatorLayout, "www.journaldev.com", Snackbar.LENGTH_LONG);
        snackbar.show();
      }
    });

    two.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Snackbar snackbar = Snackbar
            .make(coordinatorLayout, "Message is deleted", Snackbar.LENGTH_LONG)
            .setAction("UNDO", new View.OnClickListener() {
              @Override
              public void onClick(View view) {
                Snackbar snackbar1 = Snackbar.make(coordinatorLayout, "Message is restored!",
Snackbar.LENGTH_SHORT);
                snackbar1.show();
              }
            });

        snackbar.show();
      }

    });

    three.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Snackbar snackbar = Snackbar
            .make(coordinatorLayout, "Try again!", Snackbar.LENGTH_LONG)
            .setAction("RETRY", new View.OnClickListener() {
              @Override
              public void onClick(View view) {
              }
            });
        snackbar.setActionTextColor(Color.RED);
        View sbView = snackbar.getView();
        TextView textView = (TextView)
```

```
sbView.findViewById(android.support.design.R.id.snackbar_text);
            textView.setTextColor(Color.YELLOW);
            snackbar.show();
        }
    });

}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

    return super.onOptionsItemSelected(item);
}
}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    android:id="@+id/coordinatorLayout"
    tools:context="com.journaldev.snackbar.MainActivity">
```

```
    <android.support.design.widget.AppBarLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:theme="@style/AppTheme.AppBarOverlay">

      <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        app:popupTheme="@style/AppTheme.PopupOverlay" />

    </android.support.design.widget.AppBarLayout>

    <include layout="@layout/content_main"
      android:id="@+id/layout"/>

    <android.support.design.widget.FloatingActionButton
      android:id="@+id/fab"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="bottom|end"
      android:layout_margin="@dimen/fab_margin"
      android:src="@android:drawable/ic_dialog_email" />

</android.support.design.widget.CoordinatorLayout>
```

content_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context="com.journaldev.snackbar.MainActivity"
    tools:showIn="@layout/activity_main">
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="DEFAULT SNACKBAR"
    android:id="@+id/button"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true" />
```

```
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="ACTION CALL SNACKBAR"
    android:id="@+id/button2"
    android:layout_below="@+id/button"
    android:layout_centerHorizontal="true" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="CUSTOM VIEW SNACKBAR"
    android:id="@+id/button3"
    android:layout_below="@+id/button2"
    android:layout_centerHorizontal="true" />
</RelativeLayout>
```

menu/menu_main.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  tools:context="com.journaldev.snackbar.MainActivity">
  <item
    android:id="@+id/action_settings"
    android:orderInCategory="100"
    android:title="@string/action_settings"
    app:showAsAction="never" />
</menu>
```

Strings.xml

```
<resources>
  <string name="app_name">SnackBar</string>
  <string name="action_settings">Settings</string>
</resources>
```

Gradle (Module)

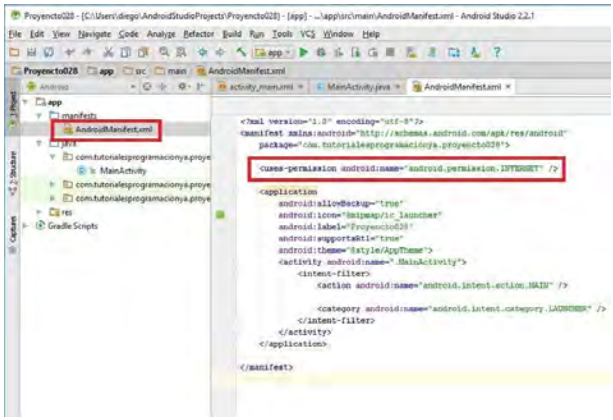
```
apply plugin: 'com.android.application'

android {
  compileSdkVersion 23
  buildToolsVersion "23.0.2"

  defaultConfig {
    applicationId "com.journaldev.snackbar"
    minSdkVersion 15
    targetSdkVersion 23
    versionCode 1
    versionName "1.0"
  }
  buildTypes {
```

```
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.1.1'
    compile 'com.android.support:design:23.1.1'
}
```

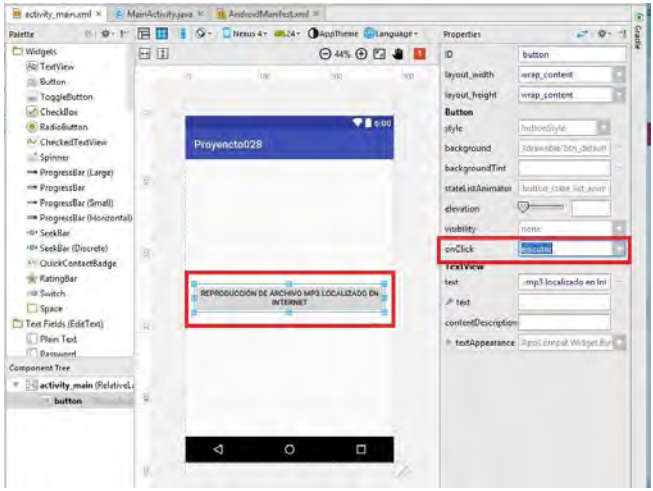
Anexo 2 Solucionario Capitulo 2

Solución problema 2.1.3.1.

Creamos un proyecto (Proyecto028) y el primer paso es modificar el archivo AndroidManifest.xml donde autorizamos a la aplicación a acceder a recursos localizados en internet (debemos abrir este archivo y proceder a agregar el permiso de acceso a internet agregando la marca uses-permission respectivo):



Creamos la interfaz de la aplicación e inicializamos el evento oClick del Button con el método que implementaremos llamado “ejecutar”:



El código fuente es:

```
package com.tutorialesprogramacionya.proyecto028;

import android.media.MediaPlayer;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

import java.io.IOException;

public class MainActivity extends AppCompatActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
```

```
    public void ejecutar(View v) {
        MediaPlayer mp = new MediaPlayer();
        try {
            mp.setDataSource("http://www.javaya.com.ar/recursos/gato.mp3");
            mp.prepare();
            mp.start();
        } catch (IOException e) {
        }
    }
}
```

Para recuperar un archivo mp3 de internet procedemos de la siguiente manera, primero creamos un objeto de la clase MediaPlayer:

MediaPlayer mp=new MediaPlayer();

Luego llamamos al método setDataSource indicando la dirección de internet donde se almacena el archivo mp3:

mp.setDataSource("http://www.javaya.com.ar/recursos/gato.mp3");

Llamamos al método prepare y seguidamente llamamos a start:

mp.prepare();

mp.start();

Todo esto lo hacemos en un bloque try/catch para capturar excepciones de tipo IOException.

Esta primera aproximación para ejecutar un mp3 localizado en internet bloquea la aplicación hasta que se carga por completo el archivo, es decir queda ejecutándose el método mp.prepare() hasta que finaliza la recuperación en forma completa.

Este proyecto lo puede descargar en un zip desde este enlace: proyecto028.zip

Problema:

Confeccionar otra aplicación similar a la anterior pero que no se congele la interfaz de la aplicación mientras se carga el mp3. Mostrar un mensaje que el archivo se está cargando.

Desarrollamos un nuevo proyecto (Proyecto029), asignamos el permiso de acceder a internet en el archivo AndroidManifest.xml y creamos una interfaz similar al problema anterior.

El código fuente es:

```
package com.tutorialesprogramacionya.proyecto029;

import android.media.MediaPlayer;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

import java.io.IOException;

public class MainActivity extends AppCompatActivity implements MediaPlayer.
OnPreparedListener{
   private MediaPlayer mp;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }

   public void ejecutar(View v) {
      mp = new MediaPlayer();
      mp.setOnPreparedListener(this);
      try {
         mp.setDataSource("http://www.javaya.com.ar/recursos/gato.mp3");
         mp.prepareAsync();
      } catch (IOException e) {
      }
      Toast t = Toast.makeText(this,
            "Espere un momento mientras se carga el mp3",
            Toast.LENGTH_SHORT);
```

```
        t.show();
    }

    @Override
    public void onPrepared(MediaPlayer mp) {
        mp.start();
    }
}
```

Para poder capturar el evento que el archivo se terminó de recuperar debemos implementar la interface MediaPlayer.OnPreparedListener:

public class MainActivity extends ActionBarActivity implements MediaPlayer.OnPreparedListener {

Con esto decimos que nuestra clase implementará el método onPrepared donde iniciamos la ejecución del mp3:

```
public void onPrepared(MediaPlayer mp) {
mp.start();
}
```

En el evento click del botón creamos el objeto de la clase MediaPlayer, le pasamos al método setOnPreparedListener la dirección del objeto que capturará el evento de que el recurso está completo. Luego llamamos a los métodos setDataSource y prepareAsync para inicializar la carga del mp3. Finalmente mostramos un mensaje para informar al usuario que el archivo se está descargando:

```
public void ejecutar(View v) {
mp=new MediaPlayer();
mp.setOnPreparedListener(this);
try {
mp.setDataSource("http://www.javaya.com.ar/recursos/gato.mp3");
mp.prepareAsync();
}catch(IOException e) {
}
```

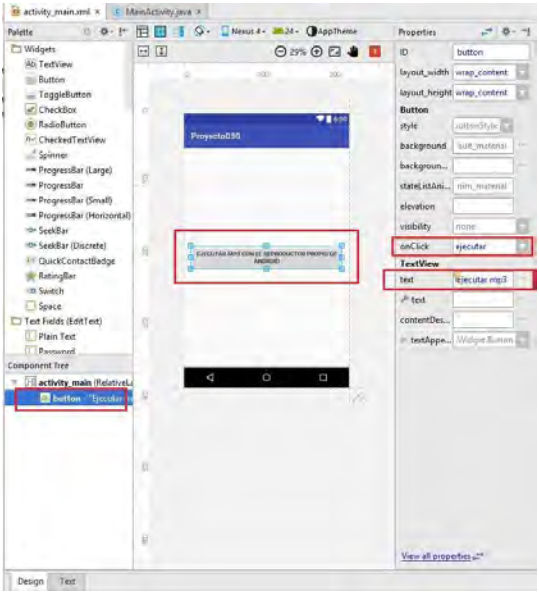
Toast t=Toast.makeText(this,"Espere un momento mientras se carga el mp3", Toast.LENGTH_SHORT);

```
t.show();
}
```

Solución problema 2.1.4.1.

Crearemos el proyecto

Creamos la interfaz con el Button y especificamos el evento onClick con el método "ejecutar":



El código fuente es:

```
package com.tutorialesprogramacionya.proyecto030;

import android.content.Intent;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
```

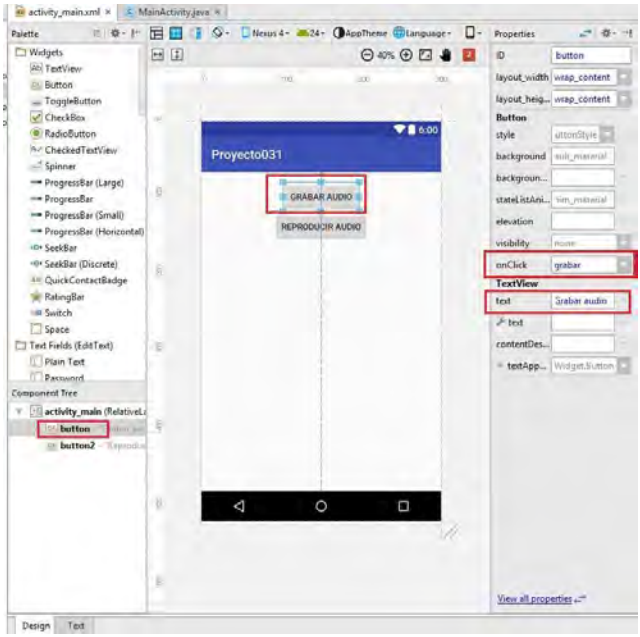
```
        setContentView(R.layout.activity_main);
    }

    public void ejecutar(View v) {
        Intent intent = new Intent(android.content.Intent.ACTION_VIEW);
        Uri data = Uri.parse("file:///sdcard" + "/gato.mp3");
        intent.setDataAndType(data, "audio/mp3");
        startActivity(intent);
    }

}
```

Creamos un objeto de la clase Intent y un objeto de la clase Uri referenciando al archivo mp3 almacenado en la tarjeta SD. Indicamos mediante el método setDataAndType el Uri y el tipo de archivo a reproducir. Activamos la aplicación mediante startActivity.

Cuando presionamos el botón vemos como se activa el reproductor propio de android y es el que realmente reproduce el archivo de audio:

Solución problema 2.1.5.1.

Crear un proyecto (Proyecto031) e implementar la interfaz, inicializar los eventos onClick de cada botón:



El código fuente es:

```
package com.tutorialesprogramacionya.proyecto031;

import android.content.Intent;
import android.media.MediaPlayer;
import android.net.Uri;
import android.provider.MediaStore;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {
   int peticion = 1;
   Uri url1;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  public void grabar(View v) {
    Intent intent = new Intent(MediaStore.Audio.Media.RECORD_SOUND_AC-
TION);
    startActivityForResult(intent, peticion);
  }

  public void reproducir(View v) {
    MediaPlayer mediaPlayer = MediaPlayer.create(this, url1);
    mediaPlayer.start();
  }

  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode == RESULT_OK && requestCode == peticion) {
      url1 = data.getData();
    }
  }
}
```

Cuando se presiona el botón de grabar el audio mediante un Intent activamos la aplicación de grabación propia de Android.

Seguidamente llamamos al método startActivityForResult para poder recuperar la grabación luego de finalizada a través del método onActivityResult:

```
public void grabar(View v) {
    Intent intent =  new Intent(MediaStore.Audio.Media.RECORD_
SOUND_ACTION);
    startActivityForResult(intent, peticion);
}
```

Debemos pasar al método startActivityForResult además de la referencia del Intent una variable con un valor 0 o positivo (luego este valor retornará al método onActivityResult)

Cuando finalizamos la grabación se ejecuta el método onActivityResult, donde almacenamos en la variable url1 la referencia al archivo de audio creado:

```
protected void onActivityResult (int requestCode, int resultCode, Intent data) {
 if (resultCode == RESULT_OK && requestCode == peticion) {
 url1 = data.getData();
 }
}
```

Por último para ejecutar el contenido de la grabación utilizamos la clase ya vista MediaPlayer:

```
public void reproducir(View v) {
   MediaPlayer mediaPlayer = MediaPlayer.create(this, url1);
   mediaPlayer.start();
}
```

Solución problema 2.1.7.1.

Crear un proyecto.

Lo primero que haremos es modificar el archivo AndroidManifest.xml para permitir a la aplicación almacenar archivos en la memoria externa:

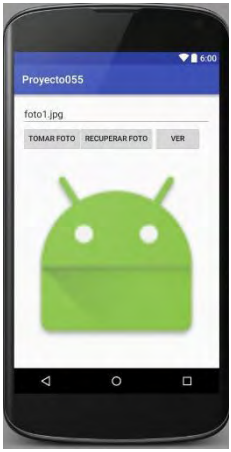
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.tutorialesprogramacionya.proyecto055">

   <uses-permission android:name="android.permission.WRITE_EXTERNAL_
STORAGE" />

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
```

```
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

</manifest>
```

Es decir agregamos la línea:

<uses-permission android:name=”android.permission.WRITE_EXTERNAL_STORAGE” />

La interfaz visual de la ventana principal es:



Disponemos un EditText, 3 botones y un objeto de la clase ImageView. El archivo “activity_main.xml” que implementa la interfaz visual es:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
```

```
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context="com.tutorialesprogramacionya.proyecto055.MainActivity">

<EditText
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:id="@+id/editText"
   android:layout_alignParentTop="true"
   android:layout_alignParentLeft="true"
   android:layout_alignParentStart="true"
   android:layout_alignParentRight="true"
   android:layout_alignParentEnd="true"
   android:text="foto1.jpg" />

<Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Tomar Foto"
   android:id="@+id/button"
   android:layout_below="@+id/editText"
   android:layout_alignParentLeft="true"
   android:layout_alignParentStart="true"
   android:onClick="tomarFoto" />

<Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Recuperar Foto"
   android:id="@+id/button2"
   android:layout_below="@+id/editText"
   android:layout_centerHorizontal="true"
   android:onClick="recuperarFoto" />
```

```
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Ver"
        android:id="@+id/button3"
        android:layout_alignBottom="@+id/button2"
        android:layout_toRightOf="@+id/button2"
        android:layout_toEndOf="@+id/button2"
        android:onClick="ver" />


    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView"
        android:layout_below="@+id/button"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:src="@mipmap/ic_launcher" />
</RelativeLayout>
```

La imagen inicial del ImageView la definimos en la propiedad src y disponemos el recurso por defecto propuesto en el proyecto (luego en tiempo de ejecución la modificamos con las imágenes de las fotos que sacamos):

```
android:src=”@mipmap/ic_launcher” />
```

Los tres botones tienen cargadas la propiedad onClick con el método respectivo:

```
android:onClick=”tomarFoto” />
android:onClick=”recuperarFoto” />
```

y

```
android:onClick=”ver” />
```

El código java asociado a esta vista se encuentra en el archivo “MainActivity.java” y es el siguiente:

```
package com.tutorialesprogramacionya.proyecto055;

import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.net.Uri;
import android.provider.MediaStore;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.ImageView;

import java.io.File;

public class MainActivity extends AppCompatActivity {

    private ImageView imagen1;
    private EditText et1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        imagen1=(ImageView)findViewById(R.id.imageView);
        et1=(EditText)findViewById(R.id.editText);
    }

    public void tomarFoto(View v) {
        Intent intento1 = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
        File foto = new File(getExternalFilesDir(null), et1.getText().toString());
        intento1.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(foto));
        startActivity(intento1);
    }
```

```
  public void recuperarFoto(View v) {
        Bitmap bitmap1 = BitmapFactory.decodeFile(getExternalFilesDir(nu-
ll)+"/"+et1.getText().toString());
    imagen1.setImageBitmap(bitmap1);
  }

  public void ver(View v) {
    Intent intento1=new Intent(this,Actividad2.class);
    startActivity(intento1);
  }
}
```

Definimos las variables et1 e imagen1:

```
private ImageView imagen1;
private EditText et1;
```

En el método onCreate obtenemos la referencia de los objetos definidos en el archivo XML:

```
imagen1=(ImageView)findViewById(R.id.imageView);
et1=(EditText)findViewById(R.id.editText);
```

Cuando se presiona el botón para tomar la foto creamos un objeto de la clase Intent y pasamos como dato la constante ACTION_IMAGE_CAPTURE definida en la clase MediaStore:

```
Intent intento1 = new Intent(MediaStore.ACTION_IMAGE_CAPTU-
RE);
```

Creamos un objeto de la clase File indicando en el primer parámetro el path donde se creará el archivo y lo obtenemos llamando al método getExternalFilesDir(null), y como segundo parámetro indicamos el nombre de archivo donde se guardará la foto:

```
File foto = new File(getExternalFilesDir(null), et1.getText().toString());
```

Pasamos al otro Activity la referencia del archivo que debe crear:

```
intento1.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(foto));
```

Finalmente lanzamos el Activity que permite tomar la foto:

```
startActivity(intento1);
```

Para recuperar una foto almacenada en la memoria externa procedemos a crear un objeto de la clase Bitmap llamando al método estático decodeFile

de la clase BitmapFactory. Debemos pasar como referencia el path donde se encuentra el archivo jpg a recuperar:

```
Bitmap bitmap1 =
BitmapFactory.decodeFile(getExternalFilesDir(null)+"/"+et1.getText().
toString());
```

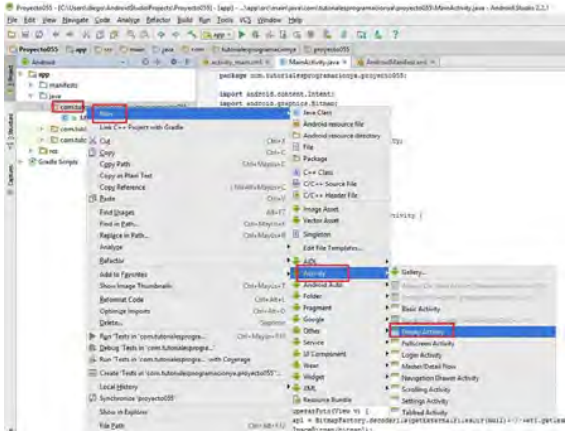
Para visualizar la imagen en el ImageView llamamos al método setImageBitmap:

```
imagen1.setImageBitmap(bitmap1);
```

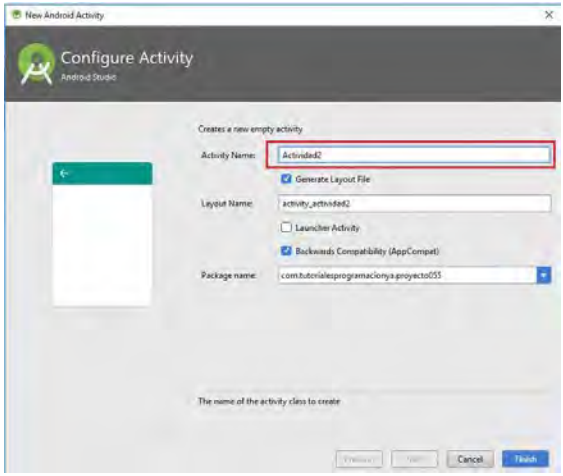
Cuando se presiona el botón ver procedemos a crear un Activity que seguidamente procederemos a crear:

```
Intent intento1=new Intent(this,Actividad2.class);
startActivity(intento1);
```

Ahora procederemos a crear la segunda actividad. Para esto presionamos el botón derecho del mouse sobre el nombre del paquete de nuestro proyecto y en el menú que aparece seleccionamos New -> Activity -> Empty Activity:



En el diálogo que aparece procedemos a cargar en "Activity Name" el valor "Actividad2":



Luego de presionar "Finish" ya tenemos los dos archivos : XML y java de nuestra aplicación.

En el archivo activity_actividad2.xml procedemos a crear la siguiente interfaz:

El contenido para crear esta pantalla está en el archivo activity_actividad2.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:id="@+id/activity_actividad2"
   android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.tutorialesprogramacionya.proyecto055.Actividad2">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:orientation="vertical" >

        <ListView
            android:id="@+id/listView1"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="1" />

        <ImageView
            android:id="@+id/imageView1"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:src="@mipmap/ic_launcher" />

    </LinearLayout>


</RelativeLayout>
```

El archivo java asociado a esta actividad se llama “Actividad2.java” y su contenido es:

```
package com.tutorialesprogramacionya.proyecto055;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.ListView;

import java.io.File;

public class Actividad2 extends AppCompatActivity {

   private ListView lv1;
   private ImageView iv1;
   private String[] archivos;
   private ArrayAdapter<String> adaptador1;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_actividad2);

      File dir=getExternalFilesDir(null);
      archivos=dir.list();
        adaptador1=new ArrayAdapter<String>(this,android.R.layout.simple_list_
item_1,archivos);
      lv1=(ListView)findViewById(R.id.listView1);
      lv1.setAdapter(adaptador1);
```

```
    iv1=(ImageView)findViewById(R.id.imageView1);

    lv1.setOnItemClickListener(new AdapterView.OnItemClickListener(){

      @Override
      public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
                  long arg3) {
        Bitmap bitmap1 = BitmapFactory.decodeFile(getExternalFilesDir(nu-
ll)+"/"+archivos[arg2]);
        iv1.setImageBitmap(bitmap1);
      }});

  }
}
```

Definimos cuatro variables:

```
private ListView lv1;
private ImageView iv1;
private String[] archivos;
private ArrayAdapter<String> adaptador1;
```

Recuperamos la lista de todos los archivos almacenados en la memoria externa y los guardamos en un vector:

```
File dir=getExternalFilesDir(null);
archivos=dir.list();
```

Creamos el ArrayAdapter con todos los nombres de archivos y los visualizamos en el ListView:

```
adaptador1=new ArrayAdapter<String>(this,android.R.layout.simple_
list_item_1,archivos);
lv1=(ListView)findViewById(R.id.listView1);
lv1.setAdapter(adaptador1);
```

Mostramos en un ImageView el contenido de la foto que seleccionamos del ListView:

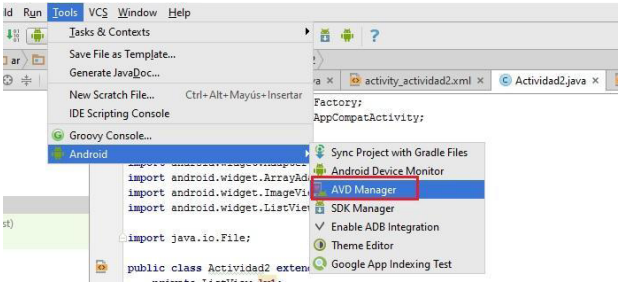
```
public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
long arg3) {
```

Bitmap bitmap1 = BitmapFactory.decodeFile(getExternalFilesDir(null)+"/"+archivos[arg2]);

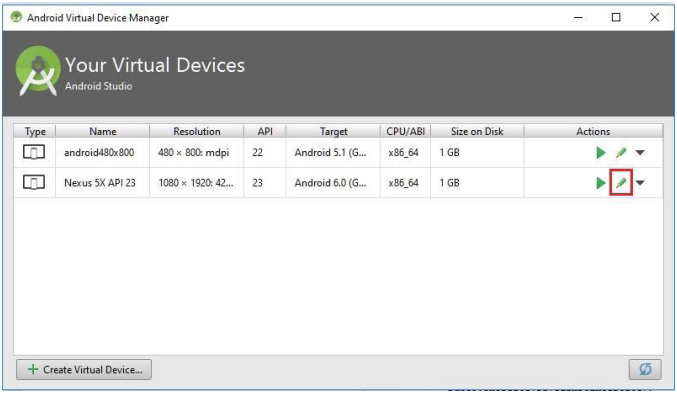
iv1.setImageBitmap(bitmap1);

}});

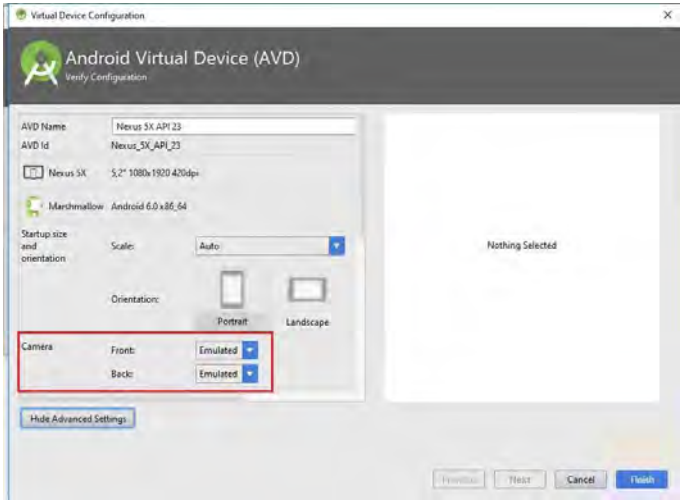
Otra parte muy importante para probar esta aplicación con el emulador de Android es su correcta configuración. Entremos a la opción AVD Manager:



En este diálogo entremos a modificar la configuración del emulador que estemos utilizando:



Y activamos la cámara:



Solución problema 2.1.8.1.

Crear un proyecto

Lo primero que haremos es modificar el archivo AndroidManifest.xml para permitir a la aplicación almacenar archivos en la memoria externa:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.tutorialesprogramacionya.proyecto056">

   <uses-permission android:name="android.permission.WRITE_EXTERNAL_
STORAGE" />

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
```

```
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

</manifest>
```

Es decir agregamos la línea:

<uses-permission android:name=”android.permission.WRITE_EXTERNAL_STORAGE” />

La interfaz visual de la ventana principal es:

Disponemos un EditText, 3 botones y un objeto de la clase VideoView. El archivo “activity_main.xml” que implementa la interfaz visual es:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context="com.tutorialesprogramacionya.proyecto056.MainActivity">

<EditText
   android:id="@+id/editText1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_alignParentLeft="true"
   android:layout_alignParentRight="true"
   android:layout_alignParentTop="true"
   android:ems="10"
   android:hint="Nombre del archivo"
   android:inputType="textPersonName"
   android:text="video1.mp4" >

   <requestFocus />
</EditText>

<Button
   android:id="@+id/button1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_alignParentLeft="true"
   android:layout_below="@+id/editText1"
   android:onClick="tomarVideo"
   android:text="Tomar video" />

<Button
   android:id="@+id/button2"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_below="@+id/editText1"
```

```
    android:layout_toRightOf="@+id/button1"
    android:onClick="recuperarVideo"
    android:text="Recuperar video" />

  <Button
    android:id="@+id/button3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/button2"
    android:layout_alignBottom="@+id/button2"
    android:layout_alignParentRight="true"
    android:onClick="ver"
    android:text="ver" />

  <VideoView
    android:id="@+id/videoView1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/button1" />
</RelativeLayout>
```

Los tres botones tienen cargadas la propiedad onClick con el método respectivo:

android:onClick=”tomarVideo”

android:onClick=”recuperarVideo”

y

android:onClick=”ver”

El código java asociado a esta vista se encuentra en el archivo “MainActivity.java” y es el siguiente:

```
package com.tutorialesprogramacionya.proyecto056;

import android.content.Intent;
import android.net.Uri;
```

```
import android.provider.MediaStore;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.VideoView;

import java.io.File;

public class MainActivity extends AppCompatActivity {

   private EditText et1;
   private VideoView vv1;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      et1=(EditText)findViewById(R.id.editText1);
      vv1=(VideoView)findViewById(R.id.videoView1);
   }

   public void tomarVideo(View v) {
      Intent intento1 = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
      File video = new File(getExternalFilesDir(null), et1.getText().toString());
      intento1.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(video));
      startActivity(intento1);
   }


   public void recuperarVideo(View v) {
       vv1.setVideoURI(Uri.parse(getExternalFilesDir(null)+"/"+et1.getText().toS-
tring()));
      vv1.start();
```

```
    }


    public void ver(View v) {
        Intent intento1=new Intent(this,Actividad2.class);
        startActivity(intento1);
    }
}
```

Definimos las variables et1 y vv1:

```
private EditText et1;
private VideoView vv1;
```

En el método onCreate obtenemos la referencia de los objetos definidos en el archivo XML:

```
et1=(EditText)findViewById(R.id.editText1);
vv1=(VideoView)findViewById(R.id.videoView1);
```

Cuando se presiona el botón para grabar un video creamos un objeto de la clase Intent y pasamos como dato la constante ACTION_VIDEO_CAPTURE definida en la clase MediaStore:

```
Intent intento1 = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
```

Creamos un objeto de la clase File indicando en el primer parámetro el path donde se creará el archivo y lo obtenemos llamando al método getExternalFilesDir(null), y como segundo parámetro indicamos el nombre de archivo mp4 donde se guardará el video:

```
File video = new File(getExternalFilesDir(null), et1.getText().toString());
```

Pasamos al otro Activity la referencia del archivo que debe crear:

```
intento1.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(video));
```

Finalmente lanzamos el Activity que permite tomar el video:

```
startActivity(intento1);
```

Para recuperar el video almacenada en la memoria externa procedemos a llamar al método setVideoUri y pasar un objeto de la clase Uri indicando el path completo donde se encuentra el archivo mp4, finalmente llamamos al método start para comenzar a mostrarlo:

vv1.setVideoURI(Uri.parse(getExternalFilesDir(null)+"/"+et1.get-Text().toString()));

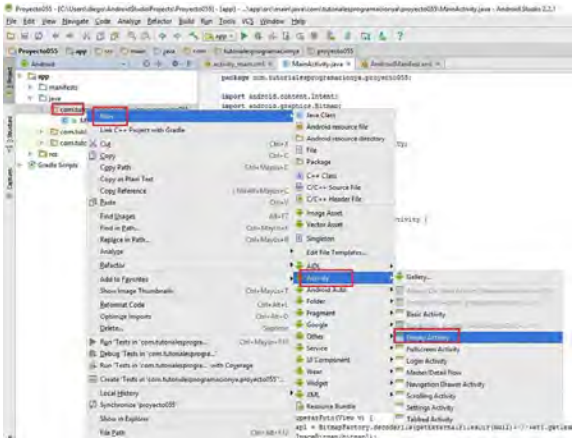
vv1.start();

Cuando se presiona el botón ver procedemos a crear un Activity que seguidamente procederemos a crear:

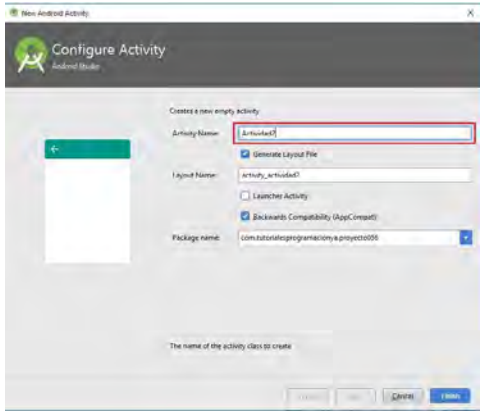
Intent intento1=new Intent(this,Actividad2.class);

startActivity(intento1);

Ahora procederemos a crear la segunda actividad. Para esto presionamos el botón derecho del mouse sobre el nombre del paquete de nuestro proyecto y en el menú que aparece seleccionamos New -> Activity -> Empty Activity:



En el diálogo que aparece procedemos a cargar en "Activity Name" el valor "Actividad2":



Luego de presionar “Finish” ya tenemos los dos archivos : XML y java de nuestra aplicación.

En el archivo activity_actividad2.xml procedemos a crear la siguiente interfaz:

El contenido para crear esta pantalla está en el archivo activity_actividad2.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:id="@+id/activity_actividad2"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="com.tutorialesprogramacionya.proyecto056.Actividad2">

   <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:layout_alignParentBottom="true"
      android:layout_alignParentLeft="true"
```

```
    android:layout_alignParentRight="true"
    android:layout_alignParentTop="true"
    android:orientation="vertical" >

    <ListView
      android:id="@+id/listView1"
      android:layout_width="match_parent"
      android:layout_height="100dp" >

    </ListView>

    <VideoView
      android:id="@+id/videoView1"
      android:layout_width="match_parent"
      android:layout_height="match_parent" />

  </LinearLayout>
</RelativeLayout>
```

El archivo java asociado a esta actividad se llama “Actividad2.java” y su contenido es:

```
package com.tutorialesprogramacionya.proyecto056;

import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.VideoView;

import java.io.File;
```

```
public class Actividad2 extends AppCompatActivity {

   private ListView lv1;
   private VideoView vv1;
   private String[] archivos;
   private ArrayAdapter<String> adaptador1;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_actividad2);

      File dir=getExternalFilesDir(null);
      archivos=dir.list();
        adaptador1=new ArrayAdapter<String>(this,android.R.layout.simple_list_
item_1,archivos);
      lv1=(ListView)findViewById(R.id.listView1);
      lv1.setAdapter(adaptador1);

      vv1=(VideoView)findViewById(R.id.videoView1);

      lv1.setOnItemClickListener(new AdapterView.OnItemClickListener(){

         @Override
         public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
                        long arg3) {
                     vv1.setVideoURI(Uri.parse(getExternalFilesDir(null)+"/"+archi-
vos[arg2]));
            vv1.start();
         }});

   }
}
```

Definimos cuatro variables:

```
private ListView lv1;
private VideoView vv1;
private String[] archivos;
private ArrayAdapter<String> adaptador1;
```

Recuperamos la lista de todos los archivos almacenados en la memoria externa y los guardamos en un vector:

```
File dir=getExternalFilesDir(null);
archivos=dir.list();
```

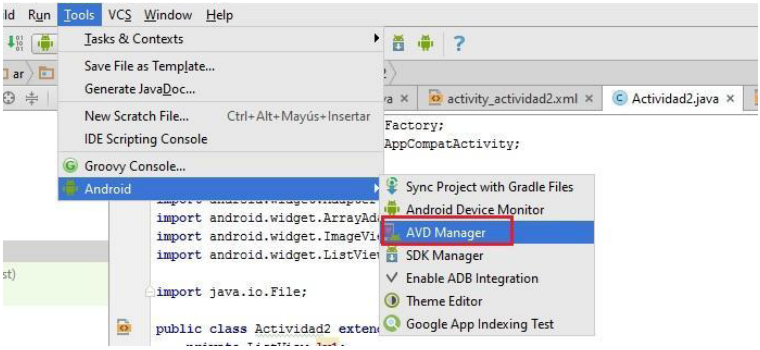
Creamos el ArrayAdapter con todos los nombres de archivos y los visualizamos en el ListView:

```
adaptador1=new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,archivos);
lv1=(ListView)findViewById(R.id.listView1);
lv1.setAdapter(adaptador1);
```

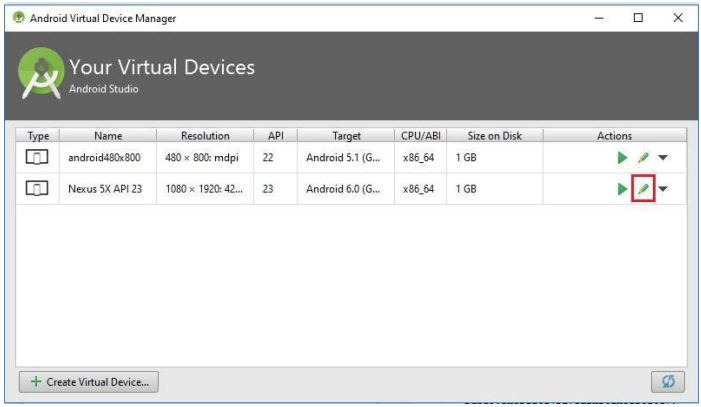
Mostramos en un VideoView el contenido del video que seleccionamos del ListView:

```
public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
long arg3) {
vv1.setVideoURI(Uri.parse(getExternalFilesDir(null)+"/"+archivos[arg2]));
vv1.start();

}});
```

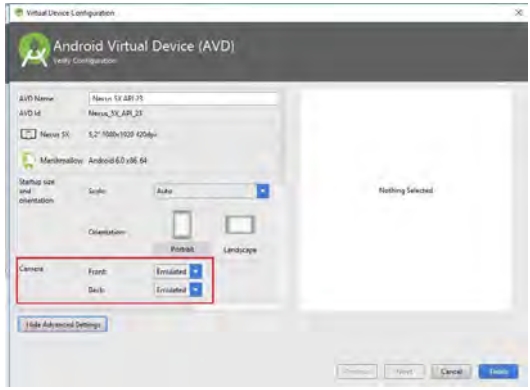
Otra parte muy importante para probar esta aplicación con el emulador de Android es su correcta configuración. Entremos a la opción AVD Manager:



En este diálogo entremos a modificar la configuración del emulador que estemos utilizando:

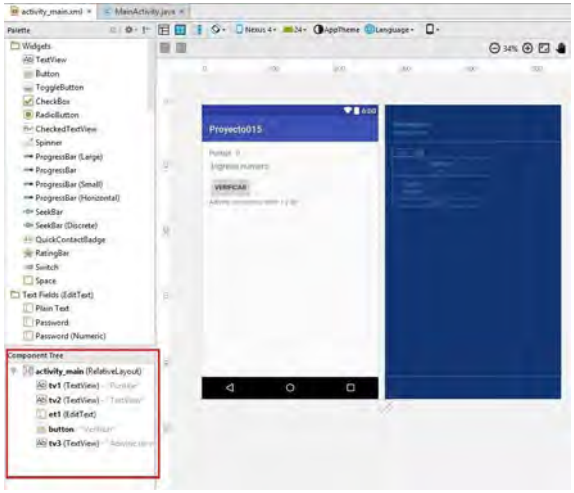


Y activamos la cámara:



Solución problema 2.2.1.3.

La interfaz visual de la aplicación a desarrollar es:

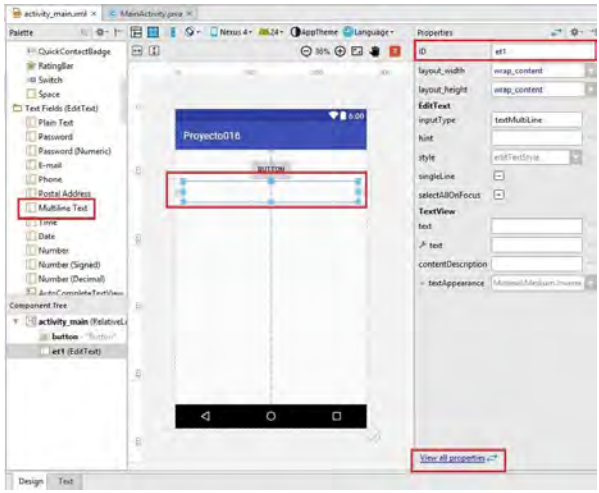


Y en tiempo de ejecución debe ser:

Solución problema 2.2.2.1.

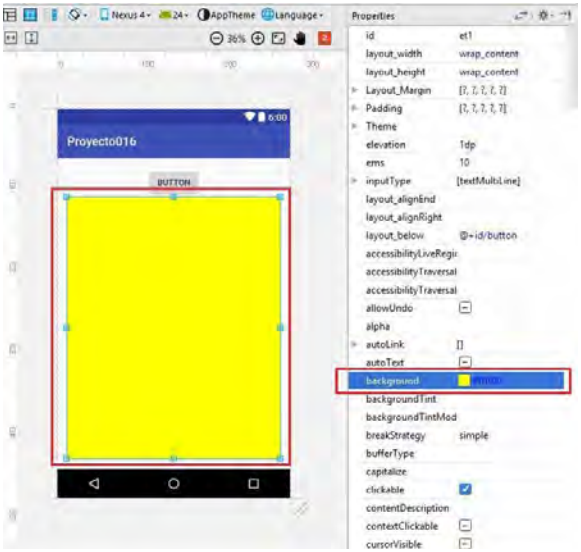
Crear un proyecto en Android Studio y definir como nombre: Proyecto016.

Para crear la interfaz visual primero disponemos un botón alineado en la parte inferior del celular y luego de la pestaña "Text Fields" seleccionamos un objeto de la clase EditText ("Multiline Text") y lo disponemos en la parte superior de la pantalla:

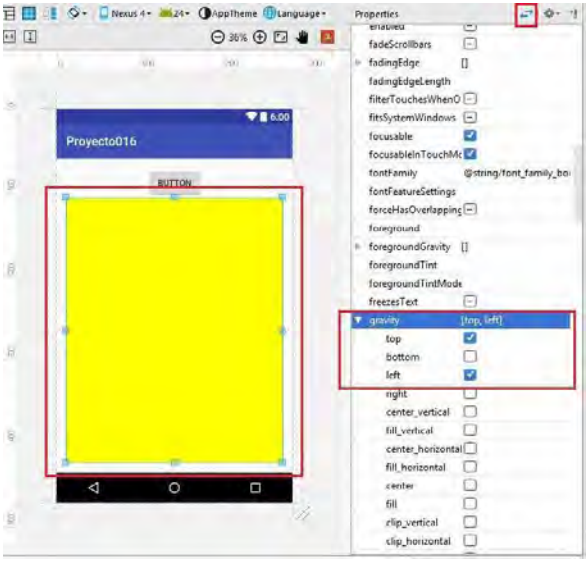


A partir de la versión 2.3.0 de Android Studio las propiedades de cada objeto aparecen las más usadas en la pantalla principal. Si queremos ver todas las propiedades que tiene el objeto debemos seleccionar la opción "View all properties" que aparece en la parte inferior de la ventana de propiedades.

Una vez que aparecen todas las propiedades inicializamos la propiedad background del EditText con el valor #ffff00 (que corresponde con el color amarillo):



Finalmente procedemos a redimensionar el EditText por medio del mouse y configuramos la propiedad gravity tildando los valores top y left para que los datos que ingresa el operador aparezcan en la parte superior izquierda y no centrados:



Podemos volver a ver la ventana de propiedades simplificadas presionando el ícono de dos flechitas de la parte superior derecha de la ventana de propiedades.

Definir para el Button la propiedad onClick con el valor “grabar” y la propiedad text con el valor “Grabar”.

El código fuente de la aplicación:

```
package com.tutorialesprogramacionya.proyecto016;

import android.app.Activity;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

public class MainActivity extends AppCompatActivity {
   private EditText et1;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      et1=(EditText)findViewById(R.id.et1);
      String[] archivos = fileList();

      if (existe(archivos, "notas.txt"))
         try {
            InputStreamReader archivo = new InputStreamReader(
                  openFileInput("notas.txt"));
```

```
            BufferedReader br = new BufferedReader(archivo);
            String linea = br.readLine();
            String todo = "";
            while (linea != null) {
               todo = todo + linea + "\n";
               linea = br.readLine();
            }
            br.close();
            archivo.close();
            et1.setText(todo);
         } catch (IOException e) {
         }
   }
   private boolean existe(String[] archivos, String archbusca) {
      for (int f = 0; f < archivos.length; f++)
         if (archbusca.equals(archivos[f]))
            return true;
      return false;
   }

   public void grabar(View v) {
      try {
         OutputStreamWriter archivo = new OutputStreamWriter(openFileOutput(
               "notas.txt", Activity.MODE_PRIVATE));
         archivo.write(et1.getText().toString());
         archivo.flush();
         archivo.close();
      } catch (IOException e) {
      }
      Toast t = Toast.makeText(this, "Los datos fueron grabados",
            Toast.LENGTH_SHORT);
      t.show();
      finish();
   }
}
```

Veamos primero como grabamos datos en un archivo de texto. Esto se hace en el método grabar que se ejecuta cuando presionamos el botón "grabar" (recordemos de inicializar la propiedad "onClick" del botón con el valor "grabar"):

```
public void grabar(View v) {
try {
OutputStreamWriter archivo = new OutputStreamWriter(openFileOutput("notas.txt",Activity.MODE_PRIVATE));
```

Creamos un objeto de la clase OutputStreamWriter y al constructor de dicha clase le enviamos el dato que retorna el método openFileOutput propio de la clase ActionBarActivity que le pasamos como parámetro el nombre del archivo de texto y el modo de apertura.

Seguidamente si el archivo se creó correctamente procedemos a llamar al método write y le pasamos el String a grabar, en este caso extraemos los datos del EditText:

```
archivo.write(et1.getText().toString());
```

Luego de grabar con el método write llamamos al método flush para que vuelque todos los datos que pueden haber quedado en el buffer y procedemos al cerrado del archivo:

```
archivo.flush();
archivo.close();
```

Todo esto está cerrado en un try/catch para verificar si sucede algún error en la apertura del archivo.

Finalmente mostramos un mensaje temporal en pantalla utilizando la clase Toast:

```
Toast t=Toast.makeText(this,"Los datos fueron grabados", Toast.LENGTH_SHORT);
t.show();
finish();
```

Para crear un objeto de la clase Toast llamamos al método makeText de la clase Toast y le pasamos la referencia del ActionBarActivity actual, el String a mostrar y el tiempo de duración del mensaje. Con el objeto devuelto por el método makeText procedemos a llamar seguidamente al método show para que se muestre el mensaje.

Es común llamar al método show de la clase Toast en una sola línea como esta:

Toast.makeText(this,"Los datos fueron grabados", Toast.LENGTH_SHORT).show();

El método onCreate tiene por objetivo verificar si existe el archivo de texto, proceder a su lectura y mostrar su contenido en el EditText.

Primero obtenemos la lista de todos los archivos creados por la Activity. En nuestro ejemplo puede ser cero o uno:

String []archivos=fileList();

Llamamos a un método que verifica si en el vector de tipo String existe el archivo "notas.txt":

if (existe(archivos,"notas.txt"))

En el caso que me retorne true procedemos a crear un objeto de la clase InputStreamReader y al constructor de dicha clase le pasamos el dato devuelto por el método openFileInput:

InputStreamReader archivo=new InputStreamReader(openFileInput("notas.txt"));

Creamos un objeto de la clase BufferedReader y le pasamos al constructor la referencia del objeto de la clase InputStreamReader:

BufferedReader br=new BufferedReader(archivo);

Leemos la primer línea del archivo de texto:

String linea=br.readLine();

Inicializamos un String vacío:

String todo="";

Mientras el método readLine de la clase BufferedReader devuelva un String:

while (linea!=null)

{

Lo concatenamos al String junto a un salto de línea:

todo=todo+linea+"\n";

Leemos la próxima línea:

linea=br.readLine();

}

Llamamos al método close de la clase BufferedReader y al del InputStreamReader:

br.close();

archivo.close();

Cargamos el EditText con el contenido del String que contiene todos los datos del archivo de texto:

et1.setText(todo);

El método existe llega un vector de tipo String y otro String a buscar. Dentro de un for verificamos el String a buscar con cada uno de los String del vector, si lo encontramos retornamos true. Si recorre todo el for sin encontrarlo fuera del for retornamos false:

```
private boolean existe(String[] archivos,String archbusca)
{
for(int f=0;f<archivos.length;f++)
if (archbusca.equals(archivos[f]))
return true;
return false;
}
```

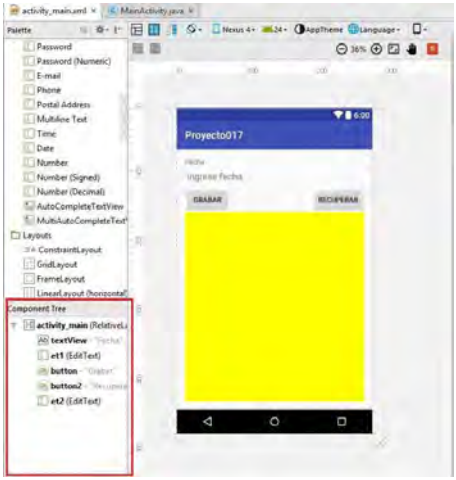
Cada vez que ejecutemos el programa veremos las últimas notas guardadas:

Este proyecto lo puede descargar en un zip desde este enlace: proyecto016.zip

Solución problema 2.2.2.2.

Crear un proyecto en Android Studio y definir como nombre: Proyecto017.

La interfaz visual a crear será la siguiente:



Es decir disponemos un "TextView" donde dice fecha. Un EditText de tipo "Date" donde se carga la fecha. Luego otro EditText de tipo "Multiline Text" y finalmente dos botones:

TextView (text="Fecha")

EditText de tipo "Date" (ID="et1", hint="Ingrese fecha")

Button (text="Grabar", onClick="Grabar")

Button (text="Recuperar" onClick="recuperar")

EditText de tipo "Multiline Text" (id="et2", background="#ffff00")

El código fuente de la aplicación:

```
package com.tutorialesprogramacionya.proyecto017;

import android.app.Activity;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.EditText;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

public class MainActivity extends AppCompatActivity {
   private EditText et1,et2;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      et1=(EditText)findViewById(R.id.et1);
      et2=(EditText)findViewById(R.id.et2);
   }

   public void grabar(View v) {
      String nomarchivo=et1.getText().toString();
      nomarchivo=nomarchivo.replace('/','-');
      try {
         OutputStreamWriter archivo = new OutputStreamWriter(openFileOutput(
               nomarchivo, Activity.MODE_PRIVATE));
         archivo.write(et2.getText().toString());
         archivo.flush();
         archivo.close();
      } catch (IOException e) {
      }
      Toast t = Toast.makeText(this, "Los datos fueron grabados",
            Toast.LENGTH_SHORT);
      t.show();
      et1.setText("");
```

```
        et2.setText("");
    }

    public void recuperar(View v) {
        String nomarchivo=et1.getText().toString();
        nomarchivo=nomarchivo.replace('/','-');
        boolean enco=false;
        String[] archivos = fileList();
        for (int f = 0; f < archivos.length; f++)
            if (nomarchivo.equals(archivos[f]))
                enco= true;
        if (enco==true) {
            try {
                InputStreamReader archivo = new InputStreamReader(
                        openFileInput(nomarchivo));
                BufferedReader br = new BufferedReader(archivo);
                String linea = br.readLine();
                String todo = "";
                while (linea != null) {
                    todo = todo + linea + "\n";
                    linea = br.readLine();
                }
                br.close();
                archivo.close();
                et2.setText(todo);
            } catch (IOException e) {
            }
        } else
        {
            Toast.makeText(this,"No hay datos grabados para dicha fecha", Toast.LEN-
GTH_LONG).show();
            et2.setText("");
        }
    }
}
```

El método grabar que se ejecuta cuando presionamos el botón grabar (no olvidar de inicializar la propiedad onClick de cada botón con el nombre del método respectivo) tenemos primero que extraer la fecha ingresada en el primer EditText y remplazar las barras de la fecha por guiones ya que no se puede utilizar este caracter dentro de un nombre de archivo en Android:

```
String nomarchivo=et1.getText().toString();
nomarchivo=nomarchivo.replace('/','-');
```

Creamos un objeto de la clase OutputStreamWriter y al constructor de dicha clase le enviamos el dato que retorna el método openFileOutput propio de la clase ActionBarActivity que le pasamos como parámetro el nombre del archivo de texto y el modo de apertura.

```
try {
   OutputStreamWriter archivo = new OutputStreamWriter(op-
enFileOutput(
      nomarchivo, Activity.MODE_PRIVATE));
```

Seguidamente si el archivo se creó correctamente procedemos a llamar al método write y le pasamos el String a grabar, en este caso extraemos los datos del EditText:

```
archivo.write(et2.getText().toString());
```

Luego de grabar con el método write llamamos al método flush para que vuelque todos los datos que pueden haber quedado en el buffer y procedemos al cerrado del archivo:

```
archivo.flush();
archivo.close();
```

Cada vez que se graba un dato se genera un archivo de texto para dicha fecha, si ya había un archivo con el mismo nombre (es decir la misma fecha) se pisa el anterior.

Para recuperar los datos de una determinada fecha se ejecuta el método "recuperar":

```
public void recuperar(View v) {
    String nomarchivo=et1.getText().toString();
    nomarchivo=nomarchivo.replace('/','-');
    boolean enco=false;
    String[] archivos = fileList();
    for (int f = 0; f < archivos.length; f++)
```

```
            if (nomarchivo.equals(archivos[f]))
                enco= true;
        if (enco==true) {
            try {
                InputStreamReader archivo = new InputStreamReader(
                        openFileInput(nomarchivo));
                BufferedReader br = new BufferedReader(archivo);
                String linea = br.readLine();
                String todo = "";
                while (linea != null) {
                    todo = todo + linea + "\n";
                    linea = br.readLine();
                }
                br.close();
                archivo.close();
                et2.setText(todo);
            } catch (IOException e) {
            }
        } else
        {
            Toast.makeText(this,"No hay datos grabados para dicha fecha",-
Toast.LENGTH_LONG).show();
            et2.setText("");
        }
    }
```

Lo primero que hacemos es recuperar del EditText la fecha que ingresó el operador para buscar:

```
String nomarchivo=et1.getText().toString();
```

Remplazamos las barras por guiones (recordemos que grabamos guiones en la carga ya que la barra de una fecha no es un caracter válido en un archivo en Android):

```
nomarchivo=nomarchivo.replace('/','-');
```

Obtenemos la lista de todos los archivos que almacena la aplicación mediante la llamada al método fileList():

```
String[] archivos = fileList();
```

Mediante un for recorremos el vector con los nombres de archivos y los comparamos con la fecha ingresada, en el caso de encontrarlo procedemos a cambiar el estado de una bandera:

```
for (int f = 0; f < archivos.length; f++)
   if (nomarchivo.equals(archivos[f]))
      enco= true;
```

Si la bandera está en true significa que existe el archivo por lo que procedemos a abrirlo, leerlo y cargar el et2 con los datos del archivo:

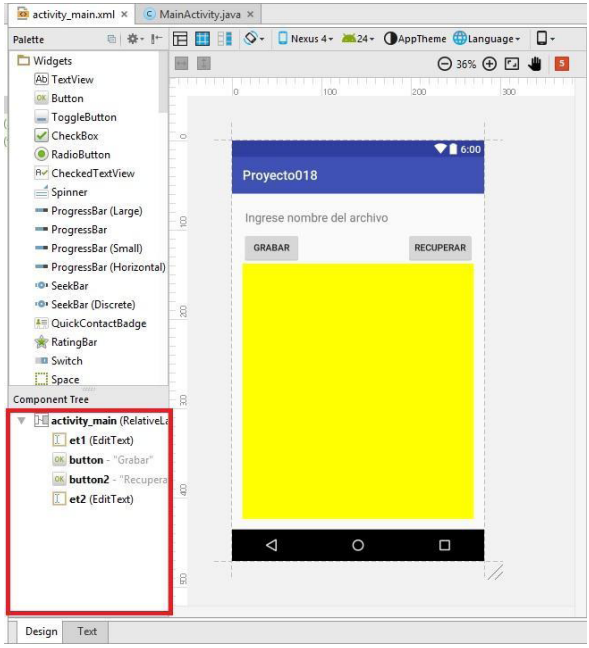
```
if (enco==true) {
   try {
      InputStreamReader archivo = new InputStreamReader(
            openFileInput(nomarchivo));
      BufferedReader br = new BufferedReader(archivo);
      String linea = br.readLine();
      String todo = "";
      while (linea != null) {
         todo = todo + linea + "\n";
         linea = br.readLine();
      }
      br.close();
      archivo.close();
      et2.setText(todo);
   } catch (IOException e) {
   }
```

La interfaz visual en tiempo de ejecución debe ser similar a esta:

Solución problema 2.2.3.1.

Crear un proyecto en Android Studio y definir como nombre: Proyecto018

La interfaz visual a implementar es la siguiente:



Podemos ver en la ventana “Component Tree” que la interfaz contiene dos EditText y dos Button, los valores que debemos iniciar sus propiedades son:

EditText de tipo “Plaint Text”(ID=”et1”, hint=”Ingrese nombre del archivo”, text=””)

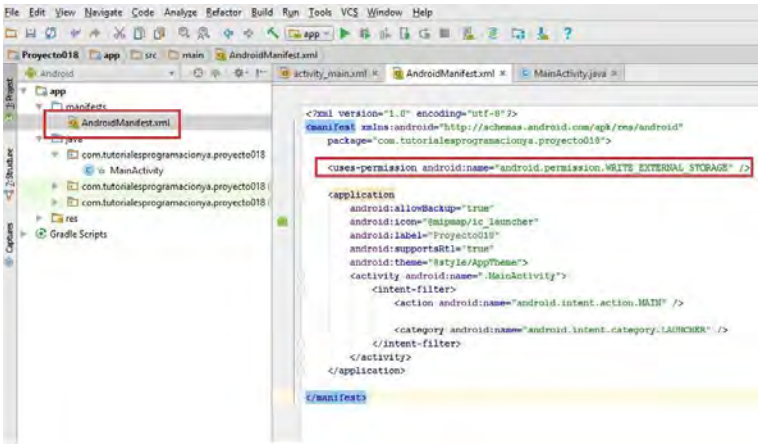
EditText de tipo “Multiline Text”(ID=”et2”, background=”#ff0000”)

Button (ID=”button”, text=”grabar”, onClick=”grabar”)

Button (ID=”button2”, text=”Recuperar”, onClick=”recuperar”)

El primer paso es modificar el archivo AndroidManifest.xml para permitir el acceso a la tarjeta SD desde nuestra aplicación esto lo hacemos desde el editor de texto del Android Studio

En la carpeta app/manifests podemos abrir el archivo “AndroidManifest.xml” y agregar la línea de permiso de acceso a la memoria externa del dispositivo:



Tenemos que agregar el permiso siguiente:

<uses-permission android:name=”android.permission.WRITE_EXTERNAL_STORAGE” />

Tener cuidado que debe estar fuera del elemento application pero dentro del elemento manifest:

<?xml version=”1.0” encoding=”utf-8”?>

<manifest xmlns:android=”http://schemas.android.com/apk/res/android”

```
package=”com.tutorialesprogramacionya.proyecto018”>
<uses-permission android:name=”android.permission.WRITE_EXTERNAL_STORAGE” />
<application
android:allowBackup=”true”
android:icon=”@mipmap/ic_launcher”
android:label=”@string/app_name”
android:supportsRtl=”true”
android:theme=”@style/AppTheme”>
<activity android:name=”.MainActivity”>
<intent-filter>
<action android:name=”android.intent.action.MAIN” />
<category android:name=”android.intent.category.LAUNCHER” />
</intent-filter>
</activity>
</application>
</manifest>
```

El código fuente es:

```
package com.tutorialesprogramacionya.proyecto018;

import android.os.Environment;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
```

```
public class MainActivity extends AppCompatActivity {
   private EditText et1,et2;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      et1=(EditText)findViewById(R.id.et1);
      et2=(EditText)findViewById(R.id.et2);
   }

   public void grabar(View v) {
      String nomarchivo = et1.getText().toString();
      String contenido = et2.getText().toString();
      try {
         File tarjeta = Environment.getExternalStorageDirectory();
            Toast.makeText(this,tarjeta.getAbsolutePath(),Toast.LENGTH_LONG).
show();
         File file = new File(tarjeta.getAbsolutePath(), nomarchivo);
         OutputStreamWriter osw = new OutputStreamWriter(
               new FileOutputStream(file));
         osw.write(contenido);
         osw.flush();
         osw.close();
         Toast.makeText(this, "Los datos fueron grabados correctamente",
               Toast.LENGTH_SHORT).show();
         et1.setText("");
         et2.setText("");
      } catch (IOException ioe) {
         Toast.makeText(this, "No se pudo grabar",
               Toast.LENGTH_SHORT).show();
      }
   }
```

```
  public void recuperar(View v) {
    String nomarchivo = et1.getText().toString();
    File tarjeta = Environment.getExternalStorageDirectory();
    File file = new File(tarjeta.getAbsolutePath(), nomarchivo);
    try {
      FileInputStream fIn = new FileInputStream(file);
      InputStreamReader archivo = new InputStreamReader(fIn);
      BufferedReader br = new BufferedReader(archivo);
      String linea = br.readLine();
      String todo = "";
      while (linea != null) {
        todo = todo + linea + " ";
        linea = br.readLine();
      }
      br.close();
      archivo.close();
      et2.setText(todo);

    } catch (IOException e) {
      Toast.makeText(this, "No se pudo leer",
          Toast.LENGTH_SHORT).show();
    }
  }
}
```

El método para grabar los datos en un archivo de texto localizado en una tarjeta SD comienza obteniendo el directorio raiz de la tarjeta a través del método getExternalStorageDirectory(), el mismo retorna un objeto de la clase File.

```
  public void grabar(View v) {
    String nomarchivo = et1.getText().toString();
    String contenido=et2.getText().toString();
    try
    {
```

```
        File tarjeta = Environment.getExternalStorageDirectory();
```

Creamos un nuevo objeto de la clase File indicando el camino de la unidad SD y el nombre del archivo a crear:

```
        File file = new File(tarjeta.getAbsolutePath(), nomarchivo);
```

Por último similar al acceso de un archivo interno creamos un objeto de la clase OutputStreamWriter:

```
          OutputStreamWriter osw =new OutputStreamWriter(new FileOutputS-
tream(file));
```

Grabamos el contenido del EditText:

```
        osw.write(contenido);
```

Cerramos el archivo:

```
        osw.flush();
        osw.close();
           Toast.makeText(this,"Los datos fueron grabados correctamente",Toast.
LENGTH_SHORT).show();
        et1.setText("");
        et2.setText("");
      }
      catch (IOException ioe)
      {
        Toast.makeText(this, "No se pudo grabar",
            Toast.LENGTH_SHORT).show();


      }
  }
```

Para la lectura del archivo nuevamente obtenemos la referencia de la tarjeta SD para obtener el path de la unidad de almacenamiento, el resto del algoritmo es similar al visto con un archivo interno:

```
  public void recuperar(View v) {
    String nomarchivo = et1.getText().toString();
    File tarjeta = Environment.getExternalStorageDirectory();
    File file = new File(tarjeta.getAbsolutePath(), nomarchivo);
    try {
      FileInputStream fIn = new FileInputStream(file);
      InputStreamReader archivo=new InputStreamReader(fIn);
      BufferedReader br=new BufferedReader(archivo);
```

```
            String linea=br.readLine();
            String todo="";
            while (linea!=null)
            {
               todo=todo+linea+"\n";
               linea=br.readLine();
            }
            br.close();
            archivo.close();
            et2.setText(todo);

        } catch (IOException e)
        {
           Toast.makeText(this, "No se pudo leer",
                   Toast.LENGTH_SHORT).show();
        }
    }
```

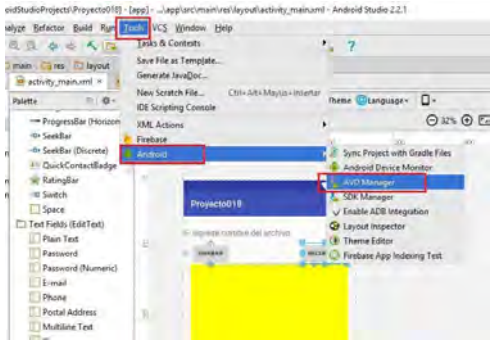
Este proyecto lo puede descargar en un zip desde este enlace: proyecto018.zip

Importante.

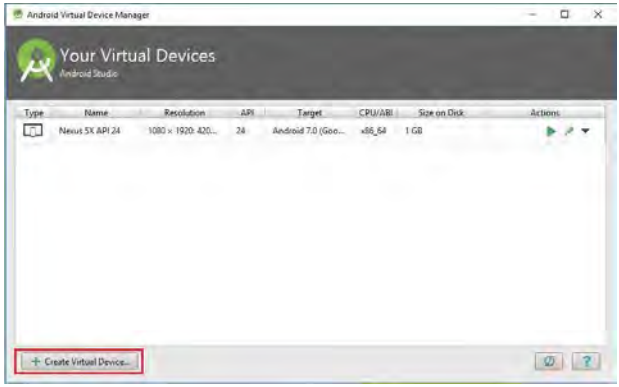
Si lo probamos con el emulador del Nexus 5x en el Android Studio cuando tratemos de grabar nos mostrará la notificación "No se pudo grabar", esto debido a que dicho celular no permite extender la memoria mediante tarjetas sd.

La solución para probar es crear otro dispositivo virtual. Los pasos para crear otro dispositivo virtual en Android Studio son los siguientes:

1. Desde el menú de opciones del Android Studio accedemos a Tools->Android->AVD Manager.

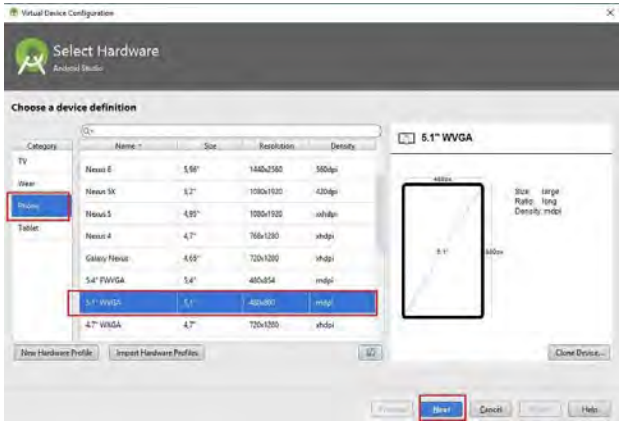


2. Aparece un diálogo con todas las máquinas virtuales creadas hasta el momento (en las primeras versiones de Android Studio crea una máquina virtual para el Nexus 5x)

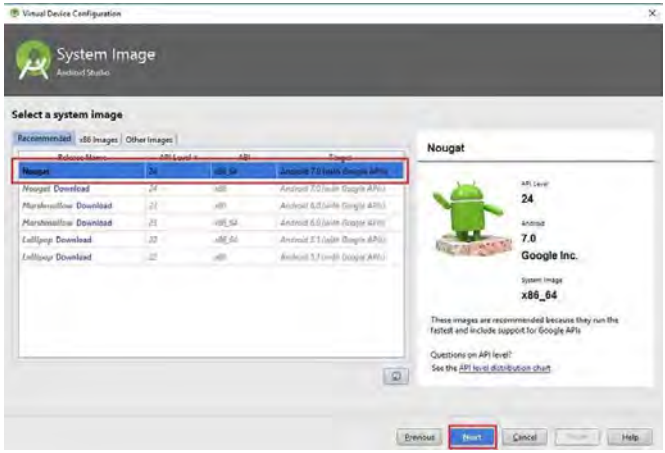


Presionamos el botón “Create Virtual Device”.

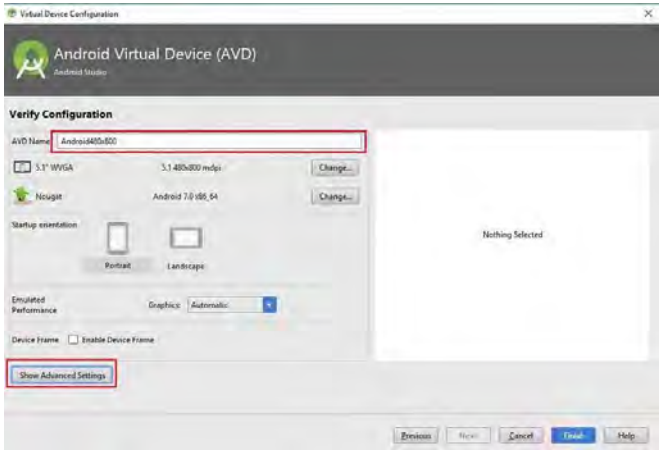
3. En este nuevo diálogo debemos seleccionar que crearemos un dispositivo virtual de tipo “Phone” y por ejemplo elegiremos uno genérico de 5.1 pulgadas:



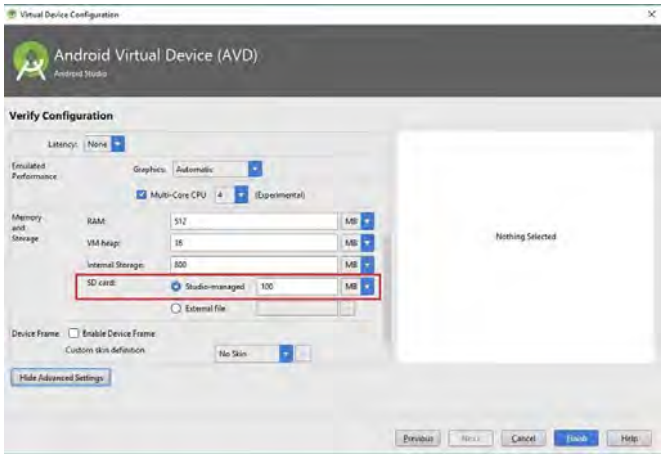
4. El siguiente diálogo seleccionamos la imagen de máquina virtual que disponemos:



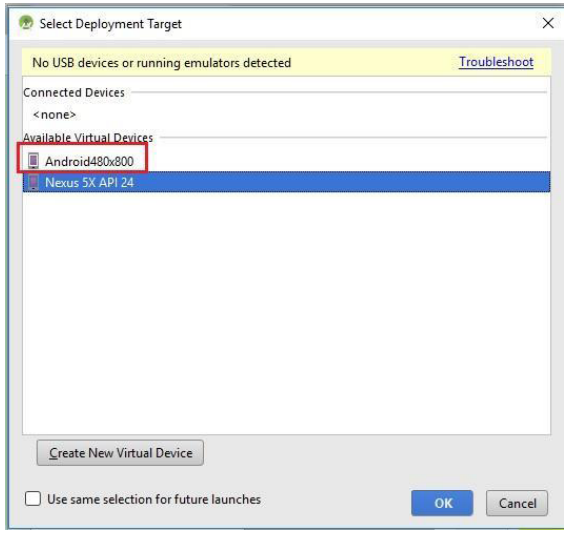
5. En el nuevo diálogo asignamos un nombre al AVD, por ejemplo: Android480x800:



6. Presionamos el botón “Show Advanced Settings” Controlamos que tenga configurado la propiedad de SD card con un valor de 100 o más:



Finalmente ya tenemos configurado nuestra nueva máquina virtual que permite almacenar datos en una tarjeta sd. Cuando ejecutemos nuevamente un proyecto tenemos que seleccionar esta nueva máquina virtual para que arranque:



Anexo 3 Solucionario Capitulo 3

Solución problema 3.1.3.1.

LoginActivity.java

```
package com.juan.inicio;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.app.ProgressDialog;
import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import com.google.android.gms.auth.api.Auth;
import com.google.android.gms.auth.api.signin.GoogleSignInAccount;
import com.google.android.gms.auth.api.signin.GoogleSignInOptions;
import com.google.android.gms.auth.api.signin.GoogleSignInResult;
import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.SignInButton;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.common.api.ResultCallback;
import com.google.android.gms.common.api.Status;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;

public class LoginActivity extends AppCompatActivity implements
GoogleApiClient.OnConnectionFailedListener{


    private EditText etUser;
    private EditText etPassword;
    private Button btnLogin;
    private Button btnRegistrar;

    private SignInButton btnSignIn;
    private Button btnSignOut;
    private Button btnRevoke;
    //CallbackManager callbackManager;

    private GoogleApiClient apiClient;
    private static final int RC_SIGN_IN = 1001;

    private ProgressDialog progressDialog;
    private static final String TAG = "MainActivity";

    //Llamamos al autenticador
    FirebaseAuth auth;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_login);

    //AccessToken accessToken = AccessToken.getCurrentAccessToken();
    //boolean isLoggedIn = accessToken != null && !accessToken.isExpired();

    //Ahora Instanciamos
    auth=FirebaseAuth.getInstance();

    etUser = (EditText) findViewById(R.id.etUser);
    etPassword = (EditText) findViewById(R.id.etPassword);
    btnLogin = (Button)findViewById(R.id.btnLogin);
    btnRegistrar = (Button)findViewById(R.id.btnRegistrar);

    btnLogin.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            String userE=etUser.getText().toString();
            String passE=etPassword.getText().toString();
            //Ahora validamos por si uno de los campos esta vacio
            if(TextUtils.isEmpty(userE)){
                //por si falta correo
                Toast.makeText(LoginActivity.this,"Inserte correo",Toast.LENGTH_SHORT).show();
                return;
            }
            if(TextUtils.isEmpty(passE)){
                //por si falta password
                Toast.makeText(LoginActivity.this,"Inserte contraseña",Toast.LENGTH_SHORT).show();
                return;
            }

            //Ahora usamos el Auth para que se logee una vez registrado
            auth.signInWithEmailAndPassword(userE,passE).
                    //Le pasamos la clase registro
                    addOnCompleteListener(LoginActivity.this, new OnCompleteListener<AuthResult>()
{
                        @Override
                        public void onComplete(@NonNull Task<AuthResult> task) {

                            if(!task.isSuccessful()){
                                Toast.makeText(LoginActivity.this,"A ocurrido un
error",Toast.LENGTH_SHORT).show();
                                return;
                            }

                            Toast.makeText(LoginActivity.this,"Bienvenido",Toast.LENGTH_SHORT).show();

                            Intent i = new Intent(LoginActivity.this,MapsActivity.class);
```

```
                startActivity(i);
            }
        });
    }
});

btnRegistrar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent i = new Intent(LoginActivity.this,Registro.class);
        startActivity(i);
    }
});

btnSignIn = (SignInButton)findViewById(R.id.sign_in_button);
btnSignOut = (Button)findViewById(R.id.sign_out_button);
btnRevoke = (Button)findViewById(R.id.revoke_button);

//Google API Client

GoogleSignInOptions gso =
    new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
        .requestEmail()
        .build();

apiClient = new GoogleApiClient.Builder(this)
    .enableAutoManage(this, this)
    .addApi(Auth.GOOGLE_SIGN_IN_API, gso)
    .build();

//Personalización del botón de login

btnSignIn.setSize(SignInButton.SIZE_STANDARD);
btnSignIn.setColorScheme(SignInButton.COLOR_LIGHT);
btnSignIn.setScopes(gso.getScopeArray());

//Eventos de los botones

btnSignIn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent signInIntent = Auth.GoogleSignInApi.getSignInIntent(apiClient);
        startActivityForResult(signInIntent, RC_SIGN_IN);
    }
});

btnSignOut.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Auth.GoogleSignInApi.signOut(apiClient).setResultCallback(
```

```
                new ResultCallback<Status>() {
                  @Override
                  public void onResult(Status status) {
                    updateUI(false);
                  }
                });
        }
    });

    btnRevoke.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Auth.GoogleSignInApi.revokeAccess(apiClient).setResultCallback(
                new ResultCallback<Status>() {
                  @Override
                  public void onResult(Status status) {
                    updateUI(false);
                  }
                });
      }
    });

    updateUI(false);

}

private void updateUI(boolean signedIn) {
    if (signedIn) {
      btnSignIn.setVisibility(View.GONE);
      btnSignOut.setVisibility(View.VISIBLE);
      btnRevoke.setVisibility(View.VISIBLE);
    } else {

      btnSignIn.setVisibility(View.VISIBLE);
      btnSignOut.setVisibility(View.GONE);
      btnRevoke.setVisibility(View.GONE);
    }
}

@Override
public void onConnectionFailed(@NonNull ConnectionResult connectionResult) {

}

@Override
public void onPointerCaptureChanged(boolean hasCapture) {

}

@Override
```

```
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (requestCode == RC_SIGN_IN) {
            GoogleSignInResult result =
                    Auth.GoogleSignInApi.getSignInResultFromIntent(data);

            handleSignInResult(result);
        }

        //Toast.makeText(this, "No existe el producto con dicho código", Toast.LENGTH_SHORT).show();
        Intent i = new Intent(this, MapsActivity.class );
        startActivity(i);

    }

    private void handleSignInResult(GoogleSignInResult result) {
        if (result.isSuccess()) {
            //Usuario logueado --> Mostramos sus datos
            GoogleSignInAccount acct = result.getSignInAccount();
            updateUI(true);
        } else {
            //Usuario no logueado --> Lo mostramos como "Desconectado"
            updateUI(false);
        }
    }
}
```

LoginActivity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".LoginActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <EditText
            android:id="@+id/etUser"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:ems="10"
```

```
    android:hint="Ingrese su correo"
    android:inputType="textEmailAddress" />

  <EditText
    android:id="@+id/etPassword"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:ems="10"
    android:hint="Ingrese su contraseña"
    android:inputType="textPassword" />

  <Button
    android:id="@+id/btnLogin"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:text="Ingresar" />

  <Button
    android:id="@+id/btnRegistrar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Registrar" />

  <com.google.android.gms.common.SignInButton
    android:id="@+id/sign_in_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center" />

  <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="40dp"
    android:layout_gravity="center_horizontal"
    android:orientation="horizontal">

    <Button
      android:id="@+id/sign_out_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center_horizontal"
      android:text="@string/sign_out" />

    <Button
      android:id="@+id/revoke_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center_horizontal"
      android:text="@string/revocar" />
```

```
    </LinearLayout>


  </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Registro.java

```
package com.juan.inicio;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.app.ProgressDialog;
import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;

public class Registro extends AppCompatActivity {

  private EditText etUser;
  private EditText etPassword;
  private Button btnRegistrar;
  FirebaseAuth auth;

  private ProgressDialog progressDialog;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_registro);

    progressDialog=new ProgressDialog(this);

    //Instanciamos igualmente
    auth=FirebaseAuth.getInstance();

    etUser=(EditText) findViewById(R.id.etUser);
```

```
    etPassword=(EditText) findViewById(R.id.etPassword);
    btnRegistrar=(Button) findViewById(R.id.btnRegistrar);

    btnRegistrar.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
        String userE=etUser.getText().toString();
        String passE=etPassword.getText().toString();
        //Ahora validamos por si uno de los campos esta vacio
        if(TextUtils.isEmpty(userE)){
          //por si falta correo
          Toast.makeText(Registro.this,"Inserte correo",Toast.LENGTH_SHORT).show();
          return;
        }
        if(TextUtils.isEmpty(passE)){
          //por si falta password
          Toast.makeText(Registro.this,"Inserte contraseña",Toast.LENGTH_SHORT).show();
          return;
        }

        progressDialog.setMessage("En proceso");
        progressDialog.show();

        //Ahora usamos el metodo
        auth.createUserWithEmailAndPassword(userE,passE).
            //Le pasamos la clase registro
                addOnCompleteListener(Registro.this, new OnCompleteListener<AuthResult>() {
              @Override
              public void onComplete(@NonNull Task<AuthResult> task) {
                //Task=Tareas devuelve si la tarea si se cumple
                //En este caso si se cumplio
                Toast.makeText(Registro.this, "Usuario registrado exitosamente",
Toast.LENGTH_SHORT).show();
                //Si no logra registrarse
                if(!task.isSuccessful()){
                  Toast.makeText(Registro.this,"Usuario no se ha podido
registrar",Toast.LENGTH_SHORT).show();
                  return;
                }
                Intent i = new Intent(Registro.this,LoginActivity.class);
                startActivity(i);
              }
            });
      }
    });

  }
}
```

Registro.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".Registro">

  <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <EditText
      android:id="@+id/etUser"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_marginTop="20dp"
      android:ems="10"
      android:hint="Ingrese su correo"
      android:inputType="textEmailAddress" />

    <EditText
      android:id="@+id/etPassword"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_marginTop="20dp"
      android:ems="10"
      android:hint="Ingrese su contraseña"
      android:inputType="textPassword" />


    <Button
      android:id="@+id/btnRegistrar"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="Registrar" />

  </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Depencias Project

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.

buildscript {
  repositories {
    google()
```

```
        maven {
            url 'https://maven.google.com/'
            name 'Google'
        }
        jcenter()

    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.4.1'
        classpath 'com.google.gms:google-services:4.3.3'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        maven {
            url 'https://maven.google.com/'
            name 'Google'
        }
        jcenter()

    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Dependencias Module

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.2"
    defaultConfig {
        applicationId "com.juan.inicio"
        minSdkVersion 16
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
```

```
        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
    }
  }
}

dependencies {
  implementation fileTree(dir: 'libs', include: ['*.jar'])
  implementation 'androidx.appcompat:appcompat:1.2.0'
  implementation 'com.google.android.gms:play-services-maps:17.0.0'
  implementation 'androidx.constraintlayout:constraintlayout:2.0.1'
  testImplementation 'junit:junit:4.12'
  androidTestImplementation 'androidx.test:runner:1.3.0'
  androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'

  implementation 'com.google.firebase:firebase-auth:19.3.2'
  implementation 'com.google.android.gms:play-services-auth:18.1.0'

}
apply plugin: 'com.google.gms.google-services'
```

Solución problema 3.2.1.1.

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="net.sgoliver.android.firebasertdb.MainActivity">

  <TextView android:id="@+id/lblCielo0"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Cielo: " />

  <TextView android:id="@+id/lblCielo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toRightOf="@id/lblCielo0"/>

  <TextView android:id="@+id/lblTemperatura0"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/lblCielo0"
    android:layout_marginTop="5dp"
```

```
    android:text="Temperatura: " />

  <TextView android:id="@+id/lblTemperatura"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/lblCielo0"
    android:layout_marginTop="5dp"
    android:layout_toRightOf="@id/lblTemperatura0"/>

  <TextView android:id="@+id/lblHumedad0"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="5dp"
    android:layout_below="@id/lblTemperatura0"
    android:text="Humedad: " />

  <TextView android:id="@+id/lblHumedad"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="5dp"
    android:layout_below="@id/lblTemperatura0"
    android:layout_toRightOf="@id/lblHumedad0"/>

  <Button android:id="@+id/btnEliminarListener"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/lblHumedad0"
    android:text="Eliminar Listener" />

</RelativeLayout>
```

MainActivity.java

```
package net.sgoliver.android.firebasertdb;

import android.support.v4.widget.TextViewCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.ButtonBarLayout;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

public class MainActivity extends AppCompatActivity {
```

```
private static final String TAGLOG = "firebase-db";

private TextView lblCielo;
private TextView lblTemperatura;
private TextView lblHumedad;
private Button btnEliminarListener;

private DatabaseReference dbCielo;
private DatabaseReference dbPrediccion;
private ValueEventListener eventListener;

@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);

  lblCielo = (TextView)findViewById(R.id.lblCielo);
  lblTemperatura = (TextView)findViewById(R.id.lblTemperatura);
  lblHumedad = (TextView)findViewById(R.id.lblHumedad);
  btnEliminarListener = (Button)findViewById(R.id.btnEliminarListener);

  /*
  dbCielo =
    FirebaseDatabase.getInstance().getReference()
      .child("prediccion-hoy")
        .child("cielo");

  dbCielo.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
      String valor = dataSnapshot.getValue().toString();
      lblCielo.setText(valor);
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
      Log.e(TAGLOG, "Error!", databaseError.toException());
    }
  });
  */

  dbPrediccion =
    FirebaseDatabase.getInstance().getReference()
      .child("prediccion-hoy");

  eventListener = new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
```

```
            //Opción 1
            //lblCielo.setText(dataSnapshot.child("cielo").getValue().toString());
            //lblTemperatura.setText(dataSnapshot.child("temperatura").getValue().toString());
            //lblHumedad.setText(dataSnapshot.child("humedad").getValue().toString());

            //Opcion 2
            Prediccion pred = dataSnapshot.getValue(Prediccion.class);
            lblCielo.setText(pred.getCielo());
            lblTemperatura.setText(pred.getTemperatura() + "ºC");
            lblHumedad.setText(pred.getHumedad() + "%");

            Log.e(TAGLOG, "onDataChange:" + dataSnapshot.getValue().toString());
        }

        @Override
        public void onCancelled(DatabaseError databaseError) {
            Log.e(TAGLOG, "Error!", databaseError.toException());
        }
    };

    dbPrediccion.addValueEventListener(eventListener);

    btnEliminarListener.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            dbPrediccion.removeEventListener(eventListener);
        }
    });
  }
}
```

Prediccion.java

```
package net.sgoliver.android.firebasertdb;

public class Prediccion {
  private String cielo;
  private long temperatura;
  private double humedad;

  public Prediccion() {
    //Es obligatorio incluir constructor por defecto
  }

  public Prediccion(String cielo, long temperatura, double humedad)
  {
    this.cielo = cielo;
    this.temperatura = temperatura;
    this.humedad = humedad;
  }
```

```
    public String getCielo() {
        return cielo;
    }

    public void setCielo(String cielo) {
        this.cielo = cielo;
    }

    public long getTemperatura() {
        return temperatura;
    }

    public void setTemperatura(long temperatura) {
        this.temperatura = temperatura;
    }

    public double getHumedad() {
        return humedad;
    }

    public void setHumedad(double humedad) {
        this.humedad = humedad;
    }

    @Override
    public String toString() {
        return "Prediccion{" +
                "cielo='" + cielo + '\'' +
                ", temperatura=" + temperatura +
                ", humedad=" + humedad +
                '}';
    }
}
```

Build.gradle (Project)

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.

buildscript {
    repositories {
        jcenter()
        maven {
            url 'https://maven.google.com/'
            name 'Google'
        }
        google()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.4.1'
        classpath 'com.google.gms:google-services:3.0.0'
```

```
    // NOTE: Do not place your application dependencies here; they belong
    // in the individual module build.gradle files
  }
}

allprojects {
  repositories {
    jcenter()
    maven {
      url 'https://maven.google.com/'
      name 'Google'
    }
  }
}

task clean(type: Delete) {
  delete rootProject.buildDir
}
```

Build.gradle (Module)

```
apply plugin: 'com.android.application'

android {
  compileSdkVersion 25
  defaultConfig {
    applicationId "net.sgoliver.android.firebasertdb"
    minSdkVersion 15
    targetSdkVersion 25
    versionCode 1
    versionName "1.0"
    testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
  }
  buildTypes {
    release {
      minifyEnabled false
      proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
    }
  }
}

dependencies {
  compile fileTree(dir: 'libs', include: ['*.jar'])
  androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
    exclude group: 'com.android.support', module: 'support-annotations'
  })
  compile 'com.android.support:appcompat-v7:25.0.0'
  compile 'com.google.firebase:firebase-database:9.6.1'
  testCompile 'junit:junit:4.12'
}
```

```
apply plugin: 'com.google.gms.google-services'
```

Solución problema 3.2.2.2.
Accedido el 29 de septiembre del 2020:
https://www.geeksforgeeks.org/how-to-populate-recyclerview-with-firebase-data-using-firebaseui-in-android-studio/

Build.Gradle (Project)

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.

buildscript {
    repositories {
        google()
        maven {
            url 'https://maven.google.com/'
            name 'Google'
        }
        jcenter()

    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.4.1'
        classpath 'com.google.gms:google-services:4.3.3'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        maven {
            url 'https://maven.google.com/'
            name 'Google'
        }
        jcenter()

    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Build.Gradel (Module)

```
apply plugin: 'com.android.application'
```

```
android {
    compileSdkVersion 29
    defaultConfig {
        applicationId "com.juan.realtimedatabase"
        minSdkVersion 16
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.1'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test:runner:1.3.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
    implementation 'com.firebaseui:firebase-ui-database:6.2.1'
    implementation 'com.google.firebase:firebase-database:19.3.1'
    implementation 'com.google.android.material:material:1.1.0'
    implementation 'com.google.firebase:firebase-core:17.0.0'
}


apply plugin: 'com.google.gms.google-services'
```

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/lstPredicciones"
            android:layout_width="match_parent"
            android:layout_height="match_parent">
        </androidx.recyclerview.widget.RecyclerView>

</RelativeLayout>
```

Item_lista.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="wrap_content">

  <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
      android:id="@+id/lblFecha"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textAppearance="@style/TextAppearance.AppCompat.Medium" />

    <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="horizontal">

      <TextView
        android:id="@+id/lblCielo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="5dp" />

      <TextView
        android:id="@+id/lblTemperatura"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="5dp" />

      <TextView
        android:id="@+id/lblHumedad"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    </LinearLayout>
  </LinearLayout>

</LinearLayout>
```

Prediccion.java

```
package com.juan.realtimedatabase;

public class Prediccion {
```

```
private String cielo;
private long temperatura;
private double humedad;
private String fecha;

public Prediccion() {
   //Es obligatorio incluir constructor por defecto
}

public Prediccion(String fecha, String cielo, long temperatura, double humedad)
{
   this.fecha = fecha;
   this.cielo = cielo;
   this.temperatura = temperatura;
   this.humedad = humedad;
}

public String getFecha() {
   return fecha;
}

public void setFecha(String fecha) {
   this.fecha = fecha;
}

public String getCielo() {
   return cielo;
}

public void setCielo(String cielo) {
   this.cielo = cielo;
}

public long getTemperatura() {
   return temperatura;
}

public void setTemperatura(long temperatura) {
   this.temperatura = temperatura;
}

public double getHumedad() {
   return humedad;
}

public void setHumedad(double humedad) {
   this.humedad = humedad;
}

@Override
```

```
    public String toString() {
        return "Prediccion{" +
                "fecha='" + fecha + '\'' +
                ", cielo='" + cielo + '\'' +
                ", temperatura=" + temperatura +
                ", humedad=" + humedad +
                '}';
    }
}
```

PrediccionHolder.java

```
package com.juan.realtimedatabase;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;

import com.firebase.ui.database.FirebaseRecyclerAdapter;
import com.firebase.ui.database.FirebaseRecyclerOptions;

import java.util.List;

public class PrediccionHolder extends FirebaseRecyclerAdapter<
        Prediccion, PrediccionHolder.prediccionViewholder> {

    public PrediccionHolder(
            @NonNull FirebaseRecyclerOptions<Prediccion> options)
    {
        super(options);
    }

    // Function to bind the view in Card view(here
    // "person.xml") iwih data in
    // model class(here "person.class")
    @Override
    protected void
    onBindViewHolder(@NonNull prediccionViewholder holder,
                     int position, @NonNull Prediccion model)
    {

        // Add fecha from model class (here
        // "Prediccion.class")to appropriate view in Card
        // view (here "item_lista.xml")
        holder.fecha.setText(model.getFecha());

        // Add cielo from model class (here
```

```
        // "Prediccion.class")to appropriate view in Card
        // view (here "item_lista.xml")
        holder.cielo.setText(model.getCielo());

        // Add humedad from model class (here
        // "Prediccion.class")to appropriate view in Card
        // view (here "item_lista.xml")
        holder.humedad.setText("Hum: "+String.valueOf(model.getHumedad())+" %");
        holder.temperatura.setText("Temp: "+String.valueOf(model.getTemperatura())+" °C");
    }

    // Function to tell the class about the Card view (here
    // "person.xml")in
    // which the data will be shown
    @NonNull
    @Override
    public prediccionViewholder
    onCreateViewHolder(@NonNull ViewGroup parent,
                       int viewType)
    {
        View view
                = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.item_lista, parent, false);
        return new PrediccionHolder.prediccionViewholder(view);
    }

    // Sub Class to create references of the views in Crad
    // view (here "person.xml")
    class prediccionViewholder
            extends RecyclerView.ViewHolder {
        TextView fecha, cielo, humedad, temperatura;
        public prediccionViewholder(@NonNull View itemView)
        {
            super(itemView);

            fecha = itemView.findViewById(R.id.lblFecha);
            cielo = itemView.findViewById(R.id.lblCielo);
            humedad = itemView.findViewById(R.id.lblHumedad);
            temperatura = itemView.findViewById(R.id.lblTemperatura);
        }
    }

}
```

Main_Activity.java

```
package com.juan.realtimedatabase;

import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
```

```
import android.os.Bundle;
import com.firebase.ui.database.FirebaseRecyclerOptions;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;


public class Predicciones extends AppCompatActivity {

  private static final String TAGLOG = "firebase-db";

  //FirebaseRecyclerAdapter mAdapter;

  private RecyclerView recyclerView;
  PrediccionHolder
      adapter; // Create Object of the Adapter class
  DatabaseReference mbase; // Create object of the
  // Firebase Realtime Database


  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_predicciones);

    //Ejemplo con FirebaseUI

    // Create a instance of the database and get
    // its reference
    mbase = FirebaseDatabase.getInstance().getReference().child("predicciones");

    recyclerView = findViewById(R.id.lstPredicciones);

    // To display the Recycler view linearly
    recyclerView.setLayoutManager(
        new LinearLayoutManager(this));

    // It is a class provide by the FirebaseUI to make a
    // query in the database to fetch appropriate data
    FirebaseRecyclerOptions<Prediccion> options
        = new FirebaseRecyclerOptions.Builder<Prediccion>()
        .setQuery(mbase, Prediccion.class)
        .build();
    // Connecting object of required Adapter class to
    // the Adapter class itself
    adapter = new PrediccionHolder(options);
    // Connecting Adapter class with the Recycler view*/
    recyclerView.setAdapter(adapter);


  }
```

```
    // Function to tell the app to start getting
    // data from database on starting of the activity
    @Override protected void onStart()
    {
        super.onStart();
        adapter.startListening();
    }

    // Function to tell the app to stop getting
    // data from database on stoping of the activity
    @Override protected void onStop()
    {
        super.onStop();
        adapter.stopListening();
    }


}
```

Solución problema 3.2.4.1.

Build.Gradle (Project)

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.

buildscript {
    repositories {
        google()
        maven {
            url 'https://maven.google.com/'
            name 'Google'
        }
        jcenter()

    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.4.1'
        classpath 'com.google.gms:google-services:4.3.3'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        maven {
            url 'https://maven.google.com/'
            name 'Google'
```

```
        }
        jcenter()

    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Build.Gradle (Module)

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.2"
    defaultConfig {
        applicationId "com.juan.realtime4"
        minSdkVersion 16
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.1'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test:runner:1.3.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
    implementation 'com.firebaseui:firebase-ui-database:6.2.1'
    implementation 'com.google.firebase:firebase-database:19.3.1'
    implementation 'com.google.android.material:material:1.1.0'
    implementation 'com.google.firebase:firebase-core:17.0.0'
}

apply plugin: 'com.google.gms.google-services'
```

Activity_Main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <Button android:id="@+id/btnEscribirSimple"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Escribir dato simple"/>

        <Button android:id="@+id/btnEscribirMap"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Escribir Map"/>

        <Button android:id="@+id/btnEscribirList"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Escribir List"/>

        <Button android:id="@+id/btnEscribirObjeto"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Escribir Objeto Java"/>

        <Button android:id="@+id/btnUpdateChildren"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="actualizaciones simultaneas"/>

        <Button android:id="@+id/btnInsertarPush"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Insertar Push"/>

        <Button android:id="@+id/btnEliminar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Eliminar"/>

    </LinearLayout>
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

Prediccion.java

```
package com.juan.realtime4;

public class Prediccion {
  private String cielo;
  private long temperatura;
  private double humedad;
  private String fecha;

  public Prediccion() {
    //Es obligatorio incluir constructor por defecto
  }

  public Prediccion(String fecha, String cielo, long temperatura, double humedad)
  {
    this.fecha = fecha;
    this.cielo = cielo;
    this.temperatura = temperatura;
    this.humedad = humedad;
  }

  public String getFecha() {
    return fecha;
  }

  public void setFecha(String fecha) {
    this.fecha = fecha;
  }

  public String getCielo() {
    return cielo;
  }

  public void setCielo(String cielo) {
    this.cielo = cielo;
  }

  public long getTemperatura() {
    return temperatura;
  }

  public void setTemperatura(long temperatura) {
    this.temperatura = temperatura;
  }
```

```
    public double getHumedad() {
        return humedad;
    }

    public void setHumedad(double humedad) {
        this.humedad = humedad;
    }

    @Override
    public String toString() {
        return "Prediccion{" +
                "fecha='" + fecha + '\'' +
                ", cielo='" + cielo + '\'' +
                ", temperatura=" + temperatura +
                ", humedad=" + humedad +
                '}';
    }
}
```

Main_Activity.java

```
package com.juan.realtime4;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;

import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;

import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

public class MainActivity extends AppCompatActivity {


    private static final String TAGLOG = "firebase-db";

    private Button btnEscribirSimple;
    private Button btnEscribirMap;
    private Button btnEscribirList;
    private Button btnEscribirObjeto;
    private Button btnUpdateChildren;
    private Button btnInsertarPush;
    private Button btnEliminar;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);

  btnEscribirSimple = (Button)findViewById(R.id.btnEscribirSimple);
  btnEscribirSimple.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
      DatabaseReference dbRef =
          FirebaseDatabase.getInstance().getReference()
              .child("dias-semana");

      dbRef.child("dia7").setValue("domingo");
    }
  });

  btnEscribirMap = (Button)findViewById(R.id.btnEscribirMap);
  btnEscribirMap.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
      DatabaseReference dbRef =
          FirebaseDatabase.getInstance().getReference()
              .child("dias-semana");

      Map<String, String> domingo = new HashMap<>();
      domingo.put("periodo-1", "domingo-mañana");
      domingo.put("periodo-2", "domingo-tarde");
      domingo.put("periodo-3", "domingo-noche");

      dbRef.child("dia7").setValue(domingo);
    }
  });

  btnEscribirList = (Button)findViewById(R.id.btnEscribirList);
  btnEscribirList.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
      DatabaseReference dbRef =
          FirebaseDatabase.getInstance().getReference()
              .child("dias-semana");

      List<String> domingo = new LinkedList<>();
      domingo.add("mañana");
      domingo.add("tarde");
      domingo.add("noche");

      dbRef.child("dia7").setValue(domingo);
    }
```

```
});

btnEscribirObjeto = (Button)findViewById(R.id.btnEscribirObjeto);
btnEscribirObjeto.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View view) {
    DatabaseReference dbRef =
        FirebaseDatabase.getInstance().getReference()
            .child("predicciones");

    Prediccion pred =
        new Prediccion("01/12/2016", "Despejado", 29, 35);

    dbRef.child("20161201").setValue(pred);
  }
});

btnUpdateChildren = (Button)findViewById(R.id.btnUpdateChildren);
btnUpdateChildren.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View view) {

    //EJEMPLO 1 (Sin evento onComplete)
    /*
    DatabaseReference dbRef =
        FirebaseDatabase.getInstance().getReference()
            .child("predicciones");

    Map<String, Object> actualizacion = new HashMap<>();
    actualizacion.put("/temperatura", 29);
        actualizacion.put("/humedad", 34);

    dbRef.child("20161120")
        .updateChildren(actualizacion);
    */

    //EJEMPLO 2 (Con evento onComplete)
    DatabaseReference dbRef =
        FirebaseDatabase.getInstance().getReference();

    Map<String, Object> actualizacion2 = new HashMap<>();
    actualizacion2.put("/20161120/temperatura", 25);
    actualizacion2.put("/20161121/humedad", 31);

    dbRef.child("predicciones")
        .updateChildren(actualizacion2, new DatabaseReference.CompletionListener(){
          public void onComplete(DatabaseError error, DatabaseReference ref) {
            if(error == null)
              Log.i(TAGLOG, "Operación OK");
            else
```

```
                    Log.e(TAGLOG, "Error: " + error.getMessage());
                }
            });
        }
    });

    btnInsertarPush = (Button)findViewById(R.id.btnInsertarPush);
    btnInsertarPush.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            DatabaseReference dbPredicciones =
                    FirebaseDatabase.getInstance().getReference()
                            .child("predicciones");

            Prediccion p1 = new Prediccion("02/12/2016", "Soleado", 29, 35);
            Prediccion p2 = new Prediccion("03/12/2016", "Nublado", 23, 52);

            dbPredicciones.push().setValue(p1);
            dbPredicciones.push().setValue(p2);
        }
    });

    btnEliminar = (Button)findViewById(R.id.btnEliminar);
    btnEliminar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            DatabaseReference dbRef =
                    FirebaseDatabase.getInstance().getReference();

            //Alternativa 1
            dbRef.child("dias-semana").child("dia6").setValue(null);

            //Alternativa 2
            dbRef.child("dias-semana").child("dia7").removeValue();
        }
    });

  }
}
```

Anexo 4 Solucionario Capitulo 4
Solución problema 4.1.2.1.
Main Activity

```
package net.sgoliver.android.localizacion;

import android.Manifest;
import android.app.Activity;
import android.content.Intent;
```

```
import android.content.IntentSender;
import android.content.pm.PackageManager;
import android.location.Location;
import android.support.annotation.Nullable;
import android.support.v4.app.ActivityCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;
import android.widget.ToggleButton;

import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.common.api.PendingResult;
import com.google.android.gms.common.api.ResultCallback;
import com.google.android.gms.common.api.Status;
import com.google.android.gms.location.LocationListener;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationServices;
import com.google.android.gms.location.LocationSettingsRequest;
import com.google.android.gms.location.LocationSettingsResult;
import com.google.android.gms.location.LocationSettingsStatusCodes;

public class MainActivity extends AppCompatActivity
    implements GoogleApiClient.OnConnectionFailedListener,
    GoogleApiClient.ConnectionCallbacks,
    LocationListener {

  private static final String LOGTAG = "android-localizacion";

  private static final int PETICION_PERMISO_LOCALIZACION = 101;
  private static final int PETICION_CONFIG_UBICACION = 201;

  private GoogleApiClient apiClient;

  private TextView lblLatitud;
  private TextView lblLongitud;
  private ToggleButton btnActualizar;

  private LocationRequest locRequest;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    lblLatitud = (TextView) findViewById(R.id.lblLatitud);
    lblLongitud = (TextView) findViewById(R.id.lblLongitud);
    btnActualizar = (ToggleButton) findViewById(R.id.btnActualizar);
```

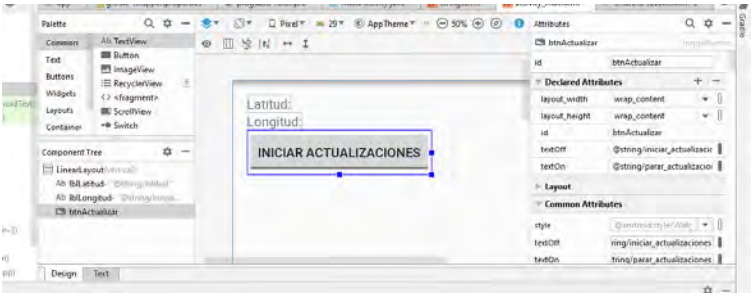
```
    btnActualizar.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
        toggleLocationUpdates(btnActualizar.isChecked());
      }
    });

    //Construcción cliente API Google
    apiClient = new GoogleApiClient.Builder(this)
        .enableAutoManage(this, this)
        .addConnectionCallbacks(this)
        .addApi(LocationServices.API)
        .build();
}

private void toggleLocationUpdates(boolean enable) {
    if (enable) {
      enableLocationUpdates();
    } else {
      disableLocationUpdates();
    }
}

private void enableLocationUpdates() {

    locRequest = new LocationRequest();
    locRequest.setInterval(2000);
    locRequest.setFastestInterval(1000);
    locRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);

    LocationSettingsRequest locSettingsRequest =
        new LocationSettingsRequest.Builder()
            .addLocationRequest(locRequest)
            .build();

    PendingResult<LocationSettingsResult> result =
        LocationServices.SettingsApi.checkLocationSettings(
            apiClient, locSettingsRequest);

    result.setResultCallback(new ResultCallback<LocationSettingsResult>() {
      @Override
      public void onResult(LocationSettingsResult locationSettingsResult) {
        final Status status = locationSettingsResult.getStatus();
        switch (status.getStatusCode()) {
          case LocationSettingsStatusCodes.SUCCESS:

            Log.i(LOGTAG, "Configuración correcta");
            startLocationUpdates();

            break;
```

```
            case LocationSettingsStatusCodes.RESOLUTION_REQUIRED:
              try {
                Log.i(LOGTAG, "Se requiere actuación del usuario");
                status.startResolutionForResult(MainActivity.this, PETICION_CONFIG_UBICACION);
              } catch (IntentSender.SendIntentException e) {
                btnActualizar.setChecked(false);
                Log.i(LOGTAG, "Error al intentar solucionar configuración de ubicación");
              }

              break;
            case LocationSettingsStatusCodes.SETTINGS_CHANGE_UNAVAILABLE:
              Log.i(LOGTAG, "No se puede cumplir la configuración de ubicación necesaria");
              btnActualizar.setChecked(false);
              break;
          }
        }
      });
  }

  private void disableLocationUpdates() {

    LocationServices.FusedLocationApi.removeLocationUpdates(
        apiClient, this);

  }

  private void startLocationUpdates() {
    if (ActivityCompat.checkSelfPermission(MainActivity.this,
        Manifest.permission.ACCESS_FINE_LOCATION) ==
PackageManager.PERMISSION_GRANTED) {

      //Ojo: estamos suponiendo que ya tenemos concedido el permiso.
      //Sería recomendable implementar la posible petición en caso de no tenerlo.

      Log.i(LOGTAG, "Inicio de recepción de ubicaciones");

      LocationServices.FusedLocationApi.requestLocationUpdates(
          apiClient, locRequest, MainActivity.this);
    }
  }

  @Override
  public void onConnectionFailed(ConnectionResult result) {
    //Se ha producido un error que no se puede resolver automáticamente
    //y la conexión con los Google Play Services no se ha establecido.

    Log.e(LOGTAG, "Error grave al conectar con Google Play Services");
  }

  @Override
```

```
  public void onConnected(@Nullable Bundle bundle) {
    //Conectado correctamente a Google Play Services

    if (ActivityCompat.checkSelfPermission(this,
        Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {

      ActivityCompat.requestPermissions(this,
          new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
          PETICION_PERMISO_LOCALIZACION);
    } else {

      Location lastLocation =
          LocationServices.FusedLocationApi.getLastLocation(apiClient);

      updateUI(lastLocation);
    }
  }

  @Override
  public void onConnectionSuspended(int i) {
    //Se ha interrumpido la conexión con Google Play Services

    Log.e(LOGTAG, "Se ha interrumpido la conexión con Google Play Services");
  }

  private void updateUI(Location loc) {
    if (loc != null) {
      lblLatitud.setText("Latitud: " + String.valueOf(loc.getLatitude()));
      lblLongitud.setText("Longitud: " + String.valueOf(loc.getLongitude()));
    } else {
      lblLatitud.setText("Latitud: (desconocida)");
      lblLongitud.setText("Longitud: (desconocida)");
    }
  }

  @Override
  public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    if (requestCode == PETICION_PERMISO_LOCALIZACION) {
      if (grantResults.length == 1
          && grantResults[0] == PackageManager.PERMISSION_GRANTED) {

        //Permiso concedido

        @SuppressWarnings("MissingPermission")
        Location lastLocation =
            LocationServices.FusedLocationApi.getLastLocation(apiClient);

        updateUI(lastLocation);
```

```
        } else {
            //Permiso denegado:
            //Deberíamos deshabilitar toda la funcionalidad relativa a la localización.

            Log.e(LOGTAG, "Permiso denegado");
        }
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
        case PETICION_CONFIG_UBICACION:
            switch (resultCode) {
                case Activity.RESULT_OK:
                    startLocationUpdates();
                    break;
                case Activity.RESULT_CANCELED:
                    Log.i(LOGTAG, "El usuario no ha realizado los cambios de configuración necesarios");
                    btnActualizar.setChecked(false);
                    break;
            }
            break;
    }
}

@Override
public void onLocationChanged(Location location) {

    Log.i(LOGTAG, "Recibida nueva ubicación!");

    //Mostramos la nueva ubicación recibida
    updateUI(location);
}
}
```

Activity XML



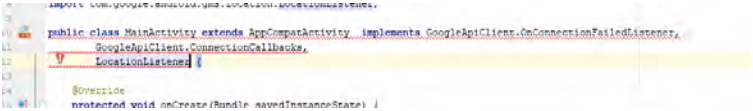

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:orientation="vertical"
    tools:context="net.sgoliver.android.localizacion.MainActivity">

    <TextView android:id="@+id/lblLatitud"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/latitud"/>

    <TextView android:id="@+id/lblLongitud"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/longitud"/>

    <ToggleButton android:id="@+id/btnActualizar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textOn="@string/parar_actualizaciones"
        android:textOff="@string/iniciar_actualizaciones" />

</LinearLayout>
```

Gradle Module, dependencys

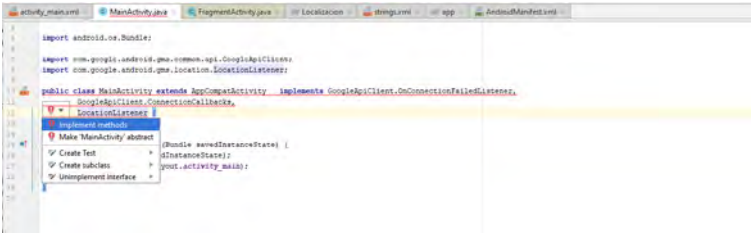
```
implementation 'com.google.android.gms:play-services-location:9.4.0'
```

Realizar lo siguiente al momento de importar estos paquetes

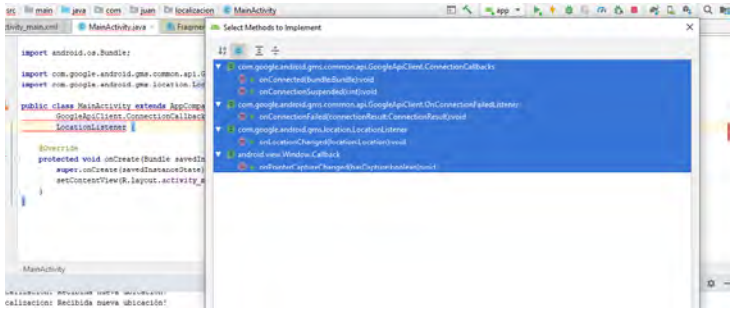
1)



2)



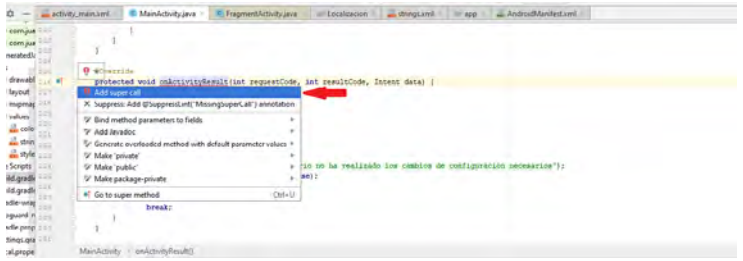
3)



4)

```
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
        case PETICION_CONFIG_UBICACION:
            switch (resultCode) {
                case Activity.RESULT_OK:
                    startLocationUpdates();
                    break;
                case Activity.RESULT_CANCELED:
                    Log.i(LOGTAG,  msg: "El usuario no ha realizado los cambios de configuración necesarios");
                    btnActualizar.setChecked(false);
                    break;
            }
            break;
    }
}
```

5)



Solución problema 4.3.1.

Se desarrollará una aplicación móvil que permita mostrar una lista de equipos de fútbol, sus datos se encontran almacenados en una base de datos Realtime Database y los escudos de cada equipo se encontrarán almacenados en el Cloud Storage de Firebase. A continuación se muestra todo el código fuente.

Build.Gradle (Project)

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.

buildscript {
    repositories {
        google()
        jcenter()

    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.4.1'
        classpath 'com.google.gms:google-services:4.3.3'
        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()

    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Build.Gradle (Module)

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.2"
    defaultConfig {
        applicationId "com.juan.cloudstorage"
        minSdkVersion 16
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.1'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test:runner:1.3.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'

    implementation 'com.google.firebase:firebase-storage:19.2.0'
    implementation 'androidx.navigation:navigation-fragment:2.2.1'
    implementation 'androidx.navigation:navigation-ui:2.2.1'
    implementation 'com.google.android.material:material:1.1.0'
    implementation 'com.google.firebase:firebase-database:17.0.0'
    implementation 'com.google.firebase:firebase-core:17.0.0'
    implementation 'com.squareup.picasso:picasso:2.71828'
    implementation 'com.android.support:cardview-v7:29.0.0'
    implementation 'com.android.support:gridlayout-v7:29.0.0'

}

apply plugin: 'com.google.gms.google-services'  // Google Play services Gradle plugin
```

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <com.google.android.material.appbar.AppBarLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:theme="@style/AppBarOverlay">

      <androidx.appcompat.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="@color/colorPrimary"
        app:popupTheme="@style/PopupOverlay"
        android:theme="@style/AppTheme.AppBarOverlay"/>

    </com.google.android.material.appbar.AppBarLayout>

    <include layout="@layout/content_equipos" />

</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

En el **Res** crear un directorio con el nombre de **navigation** y dentro de este directorio crear el archivo **nav_graph.xml**

nav_graph.xml

```
<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/nav_graph"
    app:startDestination="@id/FirstFragment">

    <fragment
      android:id="@+id/FirstFragment"
      android:name="com.juan.cloudstorage.EquiposFragment"
      android:label="@string/first_fragment_label"
      tools:layout="@layout/fragment_equipos"/>
</navigation>
```

values\strings.xml

```
<resources>
    <string name="app_name">CloudStorage</string>
    <string name="first_fragment_label">First Fragment</string>
</resources>
```

values\styles.xml

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="AppTheme.NoActionBar">
        <item name="windowActionBar">false</item>
        <item name="windowNoTitle">true</item>
    </style>

    <style name="AppTheme.AppBarOverlay" parent="ThemeOverlay.AppCompat.Dark.ActionBar"
/>

    <style name="AppTheme.PopupOverlay" parent="ThemeOverlay.AppCompat.Light" />

    <style name="NoActionBar" parent="Theme.AppCompat.Light.NoActionBar">
        <item name="windowActionBar">false</item>
        <item name="windowNoTitle">true</item>
    </style>

    <style name="AppBarOverlay" parent="ThemeOverlay.AppCompat.Dark.ActionBar" />

    <style name="PopupOverlay" parent="ThemeOverlay.AppCompat.Light" />

</resources>
```

En el **Res\Layout** crear el archivo **content_equipos.xml**

content_equipos.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior">

    <fragment
        android:id="@+id/nav_host_fragment"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:defaultNavHost="true"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
```

```
    app:layout_constraintTop_toTopOf="parent"
    app:navGraph="@navigation/nav_graph" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

En el **Res\Layout** crear el archivo **fragment_equipos.xml**
fragment_equipos.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:layout_marginLeft="10dp"
  android:layout_marginTop="10dp"
  android:layout_marginRight="10dp"
  android:layout_marginBottom="10dp"
  tools:context=".EquiposFragment">

  <androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recyclerEquipos"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />

</RelativeLayout>
```

En el **Res\Layout** crear el archivo **item_equipos.xml**
item_equipos.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  android:layout_width="match_parent"
  android:layout_height="wrap_content">

  <androidx.cardview.widget.CardView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:layout_marginBottom="10dp"
    android:foreground="?selectableItemBackground"
    app:cardCornerRadius="8dp"
    app:cardElevation="6dp"
    app:layout_columnWeight="1"
    app:layout_rowWeight="1">

    <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
```

```
        android:orientation="horizontal"
        android:padding="10dp">

        <ImageView
          android:id="@+id/imgEquipo"
          android:layout_width="10dp"
          android:layout_height="80dp"
          android:layout_weight="1"
          app:srcCompat="@drawable/logo_equipo" />

        <LinearLayout
          android:layout_width="100dp"
          android:layout_height="wrap_content"
          android:layout_weight="1"
          android:gravity="center_horizontal|center_vertical"
          android:orientation="vertical">

          <TextView
            android:id="@+id/txtNEquipo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center_horizontal|center_vertical"
            android:text="Nombre Equipo"
            android:textStyle="bold" />

          <TextView
            android:id="@+id/txtDEquipo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center_horizontal|center_vertical"
            android:text="Descripción" />
        </LinearLayout>

      </LinearLayout>
    </androidx.cardview.widget.CardView>

</LinearLayout>
```

En el **Res\drawable** insertar una imagen **logo_equipo.png**
logo_equipo.png

En la carpeta Java crear la clase:
EquipoModel.java

```
package com.juan.cloudstorage;

public class EquipoModel {
  private String idEquipo;
  private String nombre;
  private String descripcion;
  private String logoEquipoURL;

  public EquipoModel() {
  }

  public EquipoModel(String idEquipo, String nombre, String descripcion, String logoEquipoURL) {
    this.idEquipo = idEquipo;
    this.nombre = nombre;
    this.descripcion = descripcion;
    this.logoEquipoURL = logoEquipoURL;
  }

  public String getIdEquipo() {
    return idEquipo;
  }

  public void setIdEquipo(String idEquipo) {
    this.idEquipo = idEquipo;
  }

  public String getNombre() {
    return nombre;
  }

  public void setNombre(String nombre) {
    this.nombre = nombre;
  }

  public String getDescripcion() {
    return descripcion;
  }

  public void setDescripcion(String descripcion) {
    this.descripcion = descripcion;
  }

  public String getLogoEquipoURL() {
    return logoEquipoURL;
  }

  public void setLogoEquipoURL(String logoEquipoURL) {
    this.logoEquipoURL = logoEquipoURL;
  }
}
```

En la carpeta Java crear la clase:
EquipoRecyclerViewAdapter.java

```
package com.juan.cloudstorage;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;

import com.squareup.picasso.Picasso;

import java.util.List;

public class EquipoRecyclerViewAdapter extends
RecyclerView.Adapter<EquipoRecyclerViewAdapter.ViewHolder> {
  public static class ViewHolder extends RecyclerView.ViewHolder{
    private TextView nombreEquipo,descripcionEquipo;
    private ImageView logoEquipo;

    public ViewHolder(@NonNull View itemView) {
      super(itemView);
      nombreEquipo = (TextView) itemView.findViewById(R.id.txtNEquipo);
      descripcionEquipo = (TextView) itemView.findViewById(R.id.txtDEquipo);
      logoEquipo = (ImageView) itemView.findViewById(R.id.imgEquipo);
    }
  }

  public List<EquipoModel> equiposLista;

  public EquipoRecyclerViewAdapter(List<EquipoModel> equiposLista) {
    this.equiposLista = equiposLista;
  }

  @Override
  public EquipoRecyclerViewAdapter.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent,
int viewType) {
    View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_equipo,parent,false);
    EquipoRecyclerViewAdapter.ViewHolder viewHolder = new
EquipoRecyclerViewAdapter.ViewHolder(view);
    return viewHolder;
  }

  @Override
  public void onBindViewHolder(@NonNull EquipoRecyclerViewAdapter.ViewHolder holder, int
position) {
```

```
        holder.nombreEquipo.setText(equiposLista.get(position).getNombre());
        holder.descripcionEquipo.setText(equiposLista.get(position).getDescripcion());
        //holder.logoEquipo.setImageResource();

Picasso.get().load(equiposLista.get(position).getLogoEquipoURI()).error(R.drawable.logo_equipo).into(
holder.logoEquipo);
    }

    @Override
    public int getItemCount() {
        return equiposLista.size();
    }
}
```

En la carpeta Java crear la clase:
EquiposFragment.java

```
package com.juan.cloudstorage;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import androidx.annotation.NonNull;
import androidx.fragment.app.Fragment;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

import java.util.ArrayList;
import java.util.List;

public class EquiposFragment extends Fragment {
    View vista;
    private RecyclerView recyclerViewEquipo;
    private EquipoRecyclerViewAdapter adapterEquipo;

    FirebaseDatabase database = FirebaseDatabase.getInstance();
    private DatabaseReference mDatabase;
```

```
@Override
public View onCreateView(
    LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState
) {
  // Inflate the layout for this fragment
  vista=inflater.inflate(R.layout.fragment_equipos, container, false);

  recyclerViewEquipo=(RecyclerView)vista.findViewById(R.id.recyclerEquipos);
  recyclerViewEquipo.setLayoutManager(new LinearLayoutManager(this.getContext()));
  listarEquipos();


  return vista;
}

private void listarEquipos(){
  mDatabase = database.getReference("equipo");
  mDatabase.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
      List<EquipoModel> equipos = new ArrayList<>();
      for(DataSnapshot equipoSnapshot: dataSnapshot.getChildren()){
        if(equipoSnapshot.exists()){
          EquipoModel equipoDB = new EquipoModel();
          String idEquipo = equipoSnapshot.child("idEquipo").getValue().toString();
          String nombre = equipoSnapshot.child("nombre").getValue().toString();
          String descripcion = equipoSnapshot.child("descripcion").getValue().toString();
          String logoEquipoURL = equipoSnapshot.child("logoEquipoURL").getValue().toString();
          equipoDB.setIdEquipo(idEquipo);
          equipoDB.setNombre(nombre);
          equipoDB.setDescripcion(descripcion);
          equipoDB.setLogoEquipoURL(logoEquipoURL);
          equipos.add(equipoDB);
        }
      }
      adapterEquipo = new EquipoRecyclerViewAdapter(equipos);
      recyclerViewEquipo.setAdapter(adapterEquipo);
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {

    }
  });
}


public void onViewCreated(@NonNull View view, Bundle savedInstanceState) {
  super.onViewCreated(view, savedInstanceState);
```

```
    }
}
```

MainActivity.java

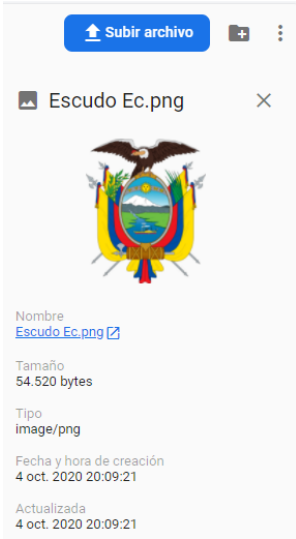
```
package com.juan.cloudstorage;

import androidx.appcompat.app.ActionBar;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;

import android.os.Bundle;
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        this.setTitle("Equipos");

        ActionBar actionBar = getSupportActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.juan.cloudstorage">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity"
            android:theme="@style/NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Estructura de la base de datos en el Realtime Database

Subir una imagen en Cloud Storage y obtener el URL haciendo clic donde se muestra



Anexo 5 Rúbrica de evaluación

Cuestionario general

1. La definición de Aplicación Móvil
2. Los sistemas operativos para dispositivos móviles
3. Al ser aplicaciones residentes en los dispositivos móviles. ¿Cuál de las siguientes opciones es una ventaja?
4. ¿Qué permite a los programadores los kits de desarrollo de software (SDK)?
5. ¿Que caracteriza a una aplicación nativa?
6. Ejemplos de aplicaciones híbridas.
7. La interfaz visual de nuestro programa para Android se almacena en un archivo XML, Que contenido nos permite visualizar
8. ¿Cuales son emuladores para Android?
9. En Android Studio, cual es el método que se usa para enlazar las variables de los objetos definidos en el archivo XML con las variables de la clase en JAVA.
10. ¿Cual control me permite mostrar una lista de String y nos permite seleccionar uno de ellos?
11. En Android cual es la función del método setOnItemClicListener de la clase ListView
12. ¿Cual clase en Android permite mostrar una ventana emergente temporal que informa al usuario mediante un mensaje que aparece en la pantalla por un lapso pequeño de tiempo?
13. ¿Cuál es el tipo de EditText que nos ofrece Android para utilizar en nuestras aplicaciones?
14. ¿Cuál es la clase en Android que me permite lanzar la ventana de un segundo Activity?
15. ¿Qué método de la clase Intent me permite enviar datos de la primera ventana a la segunda para que a partir de estos proceda a efectuar una acción?
16. ¿Cuál es el archivo de manifiesto describe información esencial como el nombre del paquete de la aplicación, los componentes de la aplicación, los permisos que necesita la aplicación para acceder a las partes protegidas del sistema o a otras aplicaciones?
17. ¿Cuál es la clase que encapsula todo lo relacionado a pintar píxeles,

líneas, rectángulos?

18. En la clase Paint llamamos al método setARGB para definir el color del pincel el primer parámetro indica el valor de transparencia. ¿Cuál de las siguientes configuraciones indicamos que no hay transparencia?
19. ¿Cuál es método de la clase Canvas donde indicamos la cantidad de rojo, verde a azul?
20. ¿Cuál es método de la clase Canvas que permite mostrar un archivo jpg, png etc?
21. ¿Cuál es el permiso que debemos configurar en el archivo "AndroidManifest.xml" para que nuestra aplicación pueda acceder a Internet?
22. La plataforma de Android nos da varias facilidades para el almacenamiento permanente de datos dentro de la aplicación. ¿Cuál de las siguientes opciones no es uno de estos métodos?
23. Cuando guardamos datos en el archivo de preferencias de la clase SharedPreferences. ¿Cuál es el método que podemos usar para almacenar el siguiente dato ("activo", true)?
24. Para leer los bytes o datos de un archivo de texto, cual es la clase que se debe usar:
25. ¿Cómo se llama la herramienta nativa de Android para almacenar datos en una base de datos?
26. ¿Cuál es la clase auxiliar para gestionar la creación de bases de datos nativa y la gestión de versiones en Android?
27. ¿Cuál es el método en SQLite que permite que la base de datos se abra en modo lectura y escritura en Android?
28. ¿Cuál componente agrupa componentes en filas y columnas?
29. ¿Cuál componente nos permite disponer una cantidad de elementos visuales que superan la cantidad de espacio del visor del celular o Tablet y luego el usuario puede desplazar con el dedo la interfaz creada?
30. ¿Cuál es el componente que permite realizar diseños más simples ya que se puede establecer los componentes visuales uno junto al otro, ya sea horizontal o verticalmente?
31. Si se usa un objeto de la clase MediaPlayer con cual método hacemos referencia al archivo de audio que copiamos en la carpeta raw
32. ¿Cuál es la clase que permite la captura de audio en Android?

33. ¿Para que se utiliza el archivo strings.xml?
34. ¿Cuál es el componente ActionBar?
35. La plataforma de Android se pueden agregar y eliminar elementos en el ListView, al momento de añadir un elemento al ArrayList , Cuál es el método que debemos llamar del ArrayAdapter para que informe al ListView que actualice los datos en pantalla
36. ¿Cuál método no es un tipo de Alerta o Notificación en Android?
37. ¿Cuál tipo de notificación constan de un icono y un texto mostrado en la barra de estado superior, y adicionalmente un mensaje algo más descriptivo, una marca de fecha/hora que podemos consultar desplegando la bandeja del sistema?
38. ¿Cuál tipo de notificación que debe utilizarse para mostrar feedbacks sobre alguna operación realizada por el usuario ya que aparece en pantalla por un corto periodo de tiempo y después desaparece automáticamente, además puede contener un botón de texto de acción?
39. En Android se puede implementar Web Services para enlazarse a un gestor de base de datos, cuál de las siguientes herramientas me ayuda con este tipo de implementación
40. ¿Cuál es la definición correcta de Apache?
41. ¿Cuál es el proveedor de localización geográfica en la que no tenemos que preocuparnos al menos no de forma explícita, de qué queremos utilizar en cada momento ya que se encargará automáticamente de gestionar todas las fuentes de datos disponibles para obtener la información que nuestra aplicación necesita?
42. ¿Cuál es la dependencia que contiene la librería completa de Google Play Services que nos daría acceso a todos los servicios disponibles?
43. Mediante cual método podemos acceder al objeto Location para obtener la latitud de una coordenada geográfica.
44. ¿Cuál método de la localización permite obtener la última posición geográfica conocida?
45. ¿Cuál de los permisos relacionados con la ubicación geográfica en Android permite acceder a datos de localización con una precisión baja, añadiendo la cláusula correspondiente <uses-permission> a nuestro fichero AndroidManifest.xml?
46. Mediante el método setInterval() se puede establecer la periodicidad de actualizaciones de la localización geográfica. ¿Cuál de las

siguientes opciones está establecido, si queremos recibir la nueva posición cada 2 segundos?

47. La precisión en la localización de los datos que queremos recibir se establecerá mediante el método setPriority(). ¿Qué valores define el modo PRIORITY_BALANCED_POWER_ACCURACY?
48. ¿En qué modo la precisión en la localización geográfica de los datos del método setPriority() es más preciso para obtener la ubicación, por lo que utilizará normalmente la señal GPS?
49. Para saber cómo solicitar el inicio de las actualizaciones de localización del dispositivo. ¿Cuál es el método que me ayuda en esta acción?
50. Cuando queremos que de alguna forma el sistema compare los requisitos de nuestra aplicación con la configuración actual de la API de localización se usa el evento que recibe como parámetro un objeto LocationSettingsResult, cuyo método getStatus() contiene el resultado de la comparación. ¿Cuál resultado indica que la configuración actual del dispositivo no es suficiente para nuestra aplicación, pero existe una posible solución por parte del usuario?
51. ¿Qué tarea nos permite realizar el servicio de Google Maps?
52. Actualmente que componente los desarrolladores usan para interactuar con los mapas
53. El método setMapType() permite modificar el tipo de vista mostrada en el mapa cual parámetro nos permite ver en mapa topográfico
54. Para los movimientos básicos en el Mapa se utiliza la clase CameraUpdateFactory. ¿Cuál método establece la latitud, longitud y el zoom para movernos lateralmente por el mapa?
55. ¿Cuál es el evento para responder a los clicks del usuario sobre el mapa?
56. ¿Cuál es el evento que detecta cambios de posición de la cámara en el mapa?
57. En la herramienta de Google Maps cuál método nos permite agregar un marcador básico a un mapa.
58. Si queremos modificar los demás parámetros de la cámara del Mapa con el objeto de tipo CameraPosition. ¿Cuál es el parámetro del ángulo de visión o vista?
59. Mediante cual método del objeto CameraPosition del Mapa podemos conocer con exactitud la posición de la cámara, la orientación

y el nivel de zoom.

60. ¿Qué herramienta web nos permite dar estilos a los mapas?
61. ¿Cuál es el servicio de Google que nos permite almacenar online cualquier tipo de archivo de forma que lo tengamos disponible en cualquier momento y desde cualquier dispositivo?
62. ¿Cuál es la credencial que necesita el Drive API para solicitar la autorización del usuario para que la aplicación pueda acceder a sus datos?
63. Si vamos a usar la referencia a la librería específica de Google Drive perteneciente a los Google Play Services cuál de las configuraciones API debemos añadir a nuestro fichero build.gradle
64. Para realizar el proceso de lectura de un fichero de Google Drive (DriveFile). ¿Cuál es el comando que me permite ejecutar esta acción?
65. En la Api de Google Drive El criterio de búsqueda lo definiremos creando un objeto Query, el cual construiremos a partir de uno o varios filtros, objetos Filter, que representarán cada una de las condiciones que debe cumplir un elemento para formar parte de los resultados de la búsqueda. ¿Cuál de las siguientes condiciones verifica si tiene contenido?
66. ¿Cuál es la definición de Firebase?
67. Firebase permite compilar mejores apps. ¿Cuál herramienta de Firebase me permite autenticar usuarios de forma simple y segura?
68. Con cuál de las siguientes herramientas no es compatible Firebase Authentication
69. ¿Cuál es la dependencia de la biblioteca de Android de Firebase Authentication?
70. ¿Cuál es el método de Firebase Authentication, para crear una cuenta nueva, pasa la dirección de correo electrónico y la contraseña del usuario nuevo?
71. ¿Cuál es el método cuando un usuario acceda a tu app, pasa la dirección de correo electrónico y la contraseña?
72. ¿Cuál es el fichero de configuración por defecto de Firebase que debemos colocar dentro de la carpeta «/app» de nuestro proyecto?
73. ¿Cuál de las siguientes funciones clave de Firebase Cloud Firestore corresponde la siguiente descripción: El modelo de datos de Cloud Firestore admite estructuras de datos flexibles y jerárquicas. Alma-

cena tus datos en documentos, organizados en colecciones. Los documentos pueden contener objetos anidados complejos, además de subcolecciones?

74. Tanto Realtime Database como Cloud Firestore son bases de datos NoSQL.
75. Con respecto al modelo de datos a cuál corresponde la siguiente característica: Los datos complejos y jerárquicos son más fáciles de organizar a escala, con subcolecciones dentro de los documentos.
76. Con respecto a las consultas cuál de las siguientes características corresponde al Cloud Firestore
77. Con respecto a la confiabilidad y rendimiento cuál de las siguientes características corresponde al Realtime Database
78. Con respecto a la escalabilidad cuál de las siguientes características corresponde al Realtime Database
79. Con respecto a la seguridad a cuál corresponde la siguiente característica: Las reglas no se aplican en cascada, a menos que uses un comodín.
80. Con respecto al precio cuál de las siguientes características corresponde al Cloud Firestore
81. ¿Cuál base de datos cuenta con la capacidad de almacenar datos en la nube sin la necesidad de preocuparnos por toda la infraestructura de servidor?
82. Realtime Database de Firebase nos ofrece una base de datos SQL tradicional, con sus tablas y sus registros perfectamente estructurados.
83. Cada nodo de la base de datos de Firebase tendrá un nombre o clave y un valor, o bien solo el nombre en caso de que vaya a servir como nodo padre a otros elementos. ¿A qué tipo de dato corresponde el siguiente ejemplo {«nombre»:»pepe», «edad»:25}?
84. ¿Cuál es el objeto que representa básicamente una rama del árbol JSON del Realtime Database y contendrá toda la información de un nodo determinado, con su clave, su valor, y su listado de nodos hijos?
85. ¿Cuál es el método del Realtime Database de Firebase que nos permite navegar entre los nodos hijo y que recibe como parámetro el nombre del subnodo al que queremos bajar en el árbol?
86. ¿Cuál es el evento en el Realtime Database que se lanza cada vez que

un elemento de la lista (incluidos sus subelementos) sea modificado y recibe como parámetro únicamente la información del elemento que ha sido modificado?

87. Firebase ofrece diferentes métodos de ordenación disponibles. ¿Cuál criterio ordena la información por el valor de cada elemento de la lista de un nodo, en orden ascendente?
88. Hay distintos métodos de filtrado/consulta en el Realtime Database. ¿Cuál método consulta y sólo devuelve los últimos N elementos de la lista ordenada?
89. ¿Cuál método en Realtime Database nos permite actualizar sólo una clave por operación y puede recibir como parámetro un objeto de tipo Map que contenga una serie de pares clave-valor que se insertarán como hijos de la clave sobre la que se está escribiendo?
90. ¿Cuál es el método en Realtime Database nos permite actualizar varias claves al mismo tiempo y recibe como parámetro un objeto de tipo Map con una serie de pares clave-valor donde cada clave es la ruta a una de las referencias que queremos actualizar y el valor será el que queremos asignar a dicha referencia?
91. ¿Cuál de las siguientes funciones clave de Cloud Firestore para Firebase corresponde la siguiente descripción: está diseñado para escalar a exabytes si tu app se vuelve viral? Pasa fácilmente de la fase prototipo a la de producción con la misma infraestructura que respalda a Spotify y Google Fotos.
92. ¿Cuál es la ruta de implementación de Cloud Firestore para Firebase?
93. ¿Cuál es la dependencia para la biblioteca de Android de Cloud Storage para Firebase?
94. ¿Cuál método es para descargar un archivo a Cloud Storage?
95. En Cloud Storage para Firebase cual método me ayuda a actualizar metadatos de archivos
96. Los códigos de error se definen en la clase StorageException como constantes de número entero. ¿Cuál error resultaría si no se configuró ningún depósito para Cloud Storage?
97. ¿Cuál servicio de Firebase permite el envío de mensajes entre un servidor de aplicaciones en la nube y un dispositivo Android, iOS o Web?
98. El token de registro se asigna a nuestra aplicación en el momento de su primera conexión con los servicios de mensajería Firebase Cloud

Messaging, y en condiciones normales se mantiene invariable en el tiempo. ¿Cuál es el método para conocer el token de registro que tenemos asignado?

99. ¿Cuál es el método que se llamará automáticamente cada vez que la aplicación reciba un mensaje de Firebase Cloud Messaging?

100. Firebase Cloud Messaging distingue entre dos tipos de mensajes: de notificación y de datos. ¿Cuál es el espacio límite que puede contener los mensajes de datos?

Evaluación Capitulo I y II

Ítem	Parámetro de evaluación	Descripción	Porcentaje de ponderación
1	Informe del ante Proyecto (Documento Word)	Estructura del documento • Tema • Planteamiento del problema • Formulación del problema • Justificación • Alcance • Objetivo General • Objetivos Específicos	10%
2	Concepto de la Aplicación (Documento Word)	Diseño y Desarrollo del Software • Diagrama de Casos de Uso • Diagrama Entidad – Relación • Diagramas de las Interfaces y Vistas de la Aplicación Móvil • Metodología del Desarrollo de Software	20%

Ítem	Parámetro de evaluación	Descripción	Porcentaje de ponderación
3	Uso de herramientas básicas de Software para el diseño e implementación de las interfaces (Adjuntar la carpeta del Proyecto) (Capturas de las interfaces como anexos en un documento Word)	La herramienta de Software para el DAM es Android Studio deben constar: Activities: • Dependencies • Permissions • Layouts • Buttons • Texto e Imágenes • CheckBox y Radio Button • Spinner • ListView • Action Bar • Menús • Login Notificaciones: • Toast	15%
4	Bases de Datos Cliente / Servidor (Documento Word)	La aplicación móvil debe interactuar con al menos con una base de datos que puede ser implementada en cualquiera de los siguientes gestores: • MySql • SQLite	15%
5	Presentación y Funcionamiento del Proyecto (Google Meet/ Documento Power Point/ Video)	La presentación del proyecto será presentada la semana 8 mediante diapositivas en Power Point y una ponencia por cada integrante del grupo, una demostración práctica funcionamiento de la aplicación y un video demostrativo subido en un canal de YouTube el cual se presentará al final de la exposición. El proyecto debe cumplir con lo siguiente: • Cumplir con al menos el 50% de funcionalidad • Demostración práctica de la fase I del prototipo	25%

Ítem	Parámetro de evaluación	Descripción	Porcentaje de ponderación
6	Archivos del proyecto (Google Drive)	Los archivos del proyecto deben adjuntarse en un enlace generado por el docente. en la carpeta correspondiente al grupo debe subir lo siguiente: • Un archivo de texto donde contenga los enlaces del proyecto de la Aplicación móvil en Android Studio subido en GitHub y el enlace del video demostrativo subido en YouTube. • La carpeta del proyecto en Android Studio • El informe del proyecto en formato Word y PDF • Las diapositivas de la presentación en formato Power Point Adjuntar esta rúbrica con los datos completos de cada integrante del grupo y el tema del proyecto.	15%

Evaluación Capitulo III y IV

Ítem	Parámetro de evaluación	Descripción	Porcentaje de ponderación
1	Informe del Proyecto (Documento Word)	Estructura del documento • Tema en la carátula • Planteamiento del problema • Formulación del problema • Justificación • Alcance • Objetivo General • Objetivos Específicos Diseño y Desarrollo del Software • Diagrama de Casos de Uso • Diagrama Entidad – Relación • Diagramas de las Interfaces y Vistas de la Aplicación Móvil Metodología de Desarrollo • Pasos que se siguió para cumplir los objetivos (implementar una metodología de Desarrollo de Software) Resultados • Capturas de pantalla de las interfaces con la descripción del funcionamiento, captura de la base de datos NoSQL generada en Realtime Database, gráficas de uso de la App obtenidas en Firebase. Conclusiones Recomendaciones Anexos	10%

Ítem	Parámetro de evaluación	Descripción	Porcentaje de ponderación
2	Uso de herramientas básicas de Software para el diseño e implementación de las interfaces (Capturas de las interfaces como anexos en el informe del Proyecto)	La herramienta de Software para el DAM es Android Studio deben constar: Activities: • Dependencies • Permissions • Layouts • Buttons • Texto e Imágenes • CheckBox y Radio Button • Spinner • ListView • Action Bar • Menús • Login Notificaciones: • Toast • Barras de estado • Snackbar	10%
3	Bases de Datos Cliente / Servidor	La aplicación móvil debe interactuar con los siguientes gestores de bases de datos: • Firebase Real Time Database	15%
4	Herramientas Complementarias	La aplicación móvil de contener las siguientes herramientas complementarias: • Google Maps - Localización • Firebase Authentication • Firebase Cloud Messaging	20%
5	Presentación y Funcionamiento del Proyecto (Documento Power Point/ Video)	La presentación del proyecto será presentada la semana 15 a través de diapositivas en Power Point y una exposición por cada integrante del grupo, una demostración práctica funcionamiento de la aplicación y la maqueta si es el caso, pueden mostrar un video explicativo.	25%

Ítem	Parámetro de evaluación	Descripción	Porcentaje de ponderación
6	Archivos del proyecto (Google Drive)	Los archivos del proyecto deben adjuntarse en el enlace generado por el docente. en la carpeta correspondiente al grupo debe subir lo siguiente: • Un archivo de texto donde contenga los enlaces del proyecto de la Aplicación móvil en Android Studio subido en GitHub y el enlace del video demostrativo subido en YouTube. • La carpeta del proyecto en Android Studio • El archivo del respaldo de la base de datos del Realtime Database en formato JSON. • El informe del proyecto en formato Word y PDF • Las diapositivas de la presentación en formato Power Point • Adjuntar esta rúbrica en Word con los datos completos de cada integrante del grupo y el tema del proyecto.	20%

Religación
Press
Ideas desde el Sur Global

Religación Press
Ideas desde el Sur Global

Instituto Nelson Torres

ISBN: 978-9942-642-37-0