Religación Press
Ideas desde el Sur Global

Instituto Nelson Torres

FUNDAMENTOS DE PROGRAMACIÓN CON JAVA

RAMIRO GUSTAVO AGUIRRE INGA
JUAN DAVID CHIMARRO AMAGUAÑA
DELIA JAZMÍN VILATUÑA CATAGÑA
ALBA ELIZABETH FLORES RUIZ
EDISON SANTIAGO LUCERO LUCERO

| Colección Programación |

Fundamentos de Programación con Java

Ramiro Gustavo Aguirre Inga, Juan David Chimarro Amaguaña,
Delia Jazmín Vilatuña Catagña, Alba Elizabeth Flores Ruiz,
Edison Santiago Lucero Lucero

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

Fundamentos de Programación con Java

Fundamentals of Java Programming
Fundamentos de programação com Java

Primera Edición:	2023
Editorial:	Religación Press
Materia Dewey:	005 - Programación. programas. datos de computadores
Clasificación Thema:	UMA - Técnicas de programación
	UMB - Algoritmos y estructuras de datos
BISCA:	COM051000 COMPUTERS / Programming / General
Público objetivo:	Profesional/Académico
Colección:	Programación
Soporte:	Digital
Formato:	Epub (.epub)/PDF (.pdf)
Publicado:	2023-12-06
ISBN:	978-9942-642-36-3

Citar como (APA 7)

Aguirre Inga, R.G., Chimarro Amaguaña, J.D., Vilatuña Catagña,D.J., Flores Ruiz, A.E., y Lucero Lucero, E.S. (2023). *Fundamentos de Programación con Java*. Religación Press. https://doi.org/10.46652/ReligacionPress.113

Patrocinio:

Instituto Superior Tecnológico Nelson Torres.

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

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/as

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.

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).

Delia Jazmín Vilatuña Catagña
Instituto Tecnológico Superior Nelson Torres | Cayambe | Ecuador
https://orcid.org/0009-0005-9158-5052
delia.vilatuna@intsuperior.edu.ec
Docente investigador, con una Maestría en Tecnologías de la Información, Ing. en Sistemas de información, y tecnólogo en sistemas de información, con manejo de diferentes plataformas tecnológicas, se desempeña como docente investigador en el Instituto Superior Tecnológico Nelson Torres, impartiendo asignaturas en las carreras de desarrollo de software y administración.

Alba Elizabeth Flores Ruiz
Instituto Tecnológico Superior Nelson Torres | Cayambe | Ecuador
https://orcid.org/0009-0008-4074-051X
alba.flores@intsuperior.edu.ec
Licenciada en Sistemas Computacionales, con una maestría en Tecnología Educativa y Competencias Digitales, 5 años de experiencia como Docente en Educación Superior, cuenta con 2 artículos científicos publicados.

Edison Santiago Lucero Lucero
Instituto Tecnológico Superior Nelson Torres | Cayambe | Ecuador
https://orcid.org/0009-0007-5758-312X
edison.lucero@intsuperior.edu.ec
Ingeniero en Electrónica y Redes de comunicación. Magister en Educación. En constante aprendizaje y con facilidad de adaptación ante la constante innovación en tecnología.

Resumen

Este libro es una guía para estudiantes que desean iniciarse en la programación de software utilizando Java. Los cuatro capítulos abarcan desde los conceptos básicos de programación y tipos de datos hasta las estructuras de control y datos más complejas, proporcionando una base sólida para la resolución de problemas. Los capítulos profundizan en las estructuras de control selectivas y repetitivas, el entorno integrado de desarrollo y el lenguaje de programación, y la implementación de sentencias de control 'case' y bucles 'for' y 'while'. El libro concluye destacando la importancia de las estructuras de datos en la creación de programas sólidos y escalables.

Palabras claves: java, programación, datos, software.

Abstract

This book is a guide for students who want to get started in software programming using Java. The four chapters cover everything from basic programming concepts and data types to more complex control and data structures, providing a solid foundation for problem solving. Chapters delve into selective and repetitive control structures, the integrated development environment and programming language, and the implementation of 'case' control statements and 'for' and 'while' loops. The book concludes by highlighting the importance of data structures in creating robust and scalable programs.

Keywords: java, programming, data, software.

Resumo

Este livro é um guia para estudantes que desejam começar a programar software usando Java. Os quatro capítulos abrangem desde conceitos básicos de programação e tipos de dados até estruturas de controle e dados mais complexas, fornecendo uma base sólida para a solução de problemas. Os capítulos se aprofundam em estruturas de controle seletivas e repetitivas, no ambiente de desenvolvimento integrado e na linguagem de programação, além da implementação de instruções de controle 'case' e loops 'for' e 'while'. O livro conclui destacando a importância das estruturas de dados na criação de programas robustos e escalonáveis.

Palavras-chave: java, programação, dados, software.

Contenido

Figuras

| Colección Programación |

Fundamentos de Programación con Java

Prólogo

El presente texto aborda los fundamentos esenciales de la programación en el contexto de Java, brindando un enfoque gradual y exhaustivo en el aprendizaje de las estructuras de control y el manejo de datos. A lo largo de los cuatro capítulos, se abordan temas como la introducción a la programación por computadora, las estructuras de control selectivas y repetitivas, así como el uso de arreglos y matrices. Cada capítulo contiene ejemplos de aplicación y autoevaluaciones para garantizar la comprensión y el dominio de los conceptos presentados.

Capítulo 1: Introducción a la programación y estructuras básicas

Este capítulo cubre una introducción a la programación por computadora, incluyendo lenguajes de programación, el ciclo de vida del software y la arquitectura de una computadora. También se abordan las características de un algoritmo, herramientas de un algoritmo, y ejemplos de aplicación. Se presentan la estructura secuencial, las variables y constantes, los tipos de datos simples y complejos, el dimensionamiento (arreglos-arrays) y las expresiones.

Capítulo 2: Estructuras de control selectivas y repetitivas

Se inicia con una explicación de la estructura de control selectiva, que incluye la estructura condicional "Si-Entonces", estructuras de control selectivas anidadas y ejemplos de aplicación. Posteriormente, se trata la estructura de control de selección

múltiple y las estructuras repetitivas "Mientras" y "Para", con sus correspondientes ejemplos de aplicación. El capítulo concluye con una sección de autoevaluación.

Capítulo 3: El lenguaje y su entorno integrado de desarrollo

Este capítulo se centra en el lenguaje y su entorno integrado de desarrollo, presentando una introducción a ambos. Se describen las clases y las funciones, incluyendo las funciones vacías y estáticas, así como la entrada/salida de datos. Además, se abordan los conceptos de variables o identificadores, tipos de variables y operadores. Se presentan también las sentencias de control de decisión "if" y se incluyen ejemplos de aplicación. El capítulo termina con una conclusión y una sección de autoevaluación.

Capítulo 4: Sentencias de control y estructuras de datos

Este capítulo cubre las sentencias de control "case", el bucle "for" y el bucle "while" junto con ejemplos de aplicación. Se explican las estructuras de datos, en particular la definición de vectores y matrices. El capítulo concluye con una sección de conclusión y autoevaluación.

Introducción

En un mundo impulsado por la tecnología, la capacidad de traducir problemas en soluciones de software es una habilidad fundamental. Este objetivo se centra en dotarte de las herramientas necesarias para desarrollar algoritmos básicos utilizando una combinación de lógica de programación, estructuras de control y una sintaxis adecuada en un lenguaje de programación. Al finalizar este objetivo, serás capaz de abordar problemas sencillos y diseñar soluciones algorítmicas efectivas y eficientes.

Comprensión de la Lógica de Programación: Aprenderás a analizar problemas y descomponerlos en pasos lógicos y secuenciales. Esta habilidad te permitirá identificar patrones y relaciones clave en los problemas, lo que es esencial para diseñar algoritmos.

Utilización de Estructuras de Control: Te familiarizarás con las estructuras de control básicas, como las decisiones (if-else) y los bucles (for, while), para controlar el flujo de ejecución de un programa. Saber cuándo y cómo aplicar estas estructuras es esencial para crear algoritmos funcionales.

Aprenderás la sintaxis adecuada en un lenguaje de programación seleccionado. La sintaxis es la regla y el orden en el que debes escribir las instrucciones para que el programa sea comprensible para la computadora.

Introducción a la lógica de programación y su importancia en el desarrollo de software.

Desarrollo de algoritmos: pasos para descomponer un problema en tareas lógicas.

Estructuras de control: if-else para toma de decisiones y bucles (for, while) para iteraciones.

- Manejo de variables y tipos de datos básicos.
- Práctica en la creación de algoritmos sencillos para problemas cotidianos.
- Debugging y resolución de errores básicos en el código.

La metodología de este objetivo se basa en un enfoque práctico. A medida que avances, encontrarás ejemplos de problemas y algoritmos, junto con ejercicios para aplicar lo aprendido. La práctica constante es esencial para dominar la lógica de programación y las estructuras de control.

Al completar este objetivo, serás capaz de analizar problemas simples, diseñar algoritmos que los resuelvan, implementar esos algoritmos utilizando estructuras de control y expresarlos correctamente en el lenguaje de programación elegido. Estarás en el camino para convertirte en un solucionador de problemas hábil y un programador competente en entornos básicos.

- Una breve descripción general del contenido del texto
- Una explicación detallada de los objetivos que se persiguen
- Una explicación de los contenidos y de las actividades

- Una explicación de los recursos que se necesitan para llevar a cabo las actividades propuestas.
- Una descripción de la metodología que se empleará
- Una explicación de la estructura y organización
- Una explicación sobre cómo los participantes pueden obtener ayuda si tienen alguna duda.
- Una guía metodológica suele incluir varias unidades. Estas pueden incluir una descripción de la metodología, una lista de herramientas y recursos necesarios para seguir el proceso, una descripción de la estructura de los pasos del proceso, una sección de recomendaciones para realizar cada paso, ejemplos de cómo se aplica la metodología a diferentes situaciones, y una sección de conclusiones para ayudar a la gente a entender mejor el proceso.

Objetivos

Aplicar de forma autónoma la lógica de programación para el desarrollo de algoritmos sencillos, utilizando estructuras de control y una sintaxis adecuada en lenguaje de programación.

Proceso de enseñanza para el logro de competencias

- Desarrolla aplicaciones de software mediante lenguajes de programación que se encuentren a la vanguardia de la tecnología, a fin de atender las necesidades empresariales del sector económico y productivo.

- Analiza los problemas planteados y los soluciona a través de pseudocódigos y diagramas de flujo.
- Usa herramientas que permita realizar pruebas de escritorio comprobando que el resultado sea correcto.
- Realiza sentencias e instrucciones de programación usando un lenguaje orientado a objetos, solventando problemas planteados.
- Desarrolla programas aplicando estructuras de control dentro den un lenguaje de programación.

Capítulo 1
Introducción a la programación por computadora

1.1. Introducción a la programación por computadora.

La disciplina de la programación informática se origina en la década de los 50, con el propósito de abordar cuestiones particulares mediante la utilización de computadoras. Por ejemplo, resolver problemáticas como el cálculo del IVA dentro de una factura, la administración de los fondos de reserva de los empleados o la estimación del décimo tercer salario, entre otras (Pérez Narváez, 2017).

1.1.1 Lenguajes de Programación, Ciclo de Vida de Software, Arquitectura de una computadora

A medida que la tecnología, especialmente la electrónica, avanza, comienza a surgir una nueva generación de computadoras con una mayor capacidad de procesamiento y almacenamiento. Como resultado de este progreso, se observa un aumento en la disponibilidad de lenguajes de programación que son cada vez más accesibles y potentes para la creación de programas. Entre estos lenguajes de programación se encuentran ejemplos como C, C++, Java, C#, VB, PHP, Pascal, Python, JavaScript, entre otros (García Santillán, 2014).

1.1.2 Características de un Algoritmo,

Los algoritmos exhiben atributos distintivos que son fundamentales para su desarrollo. A continuación, se destacan los más significativos:

Deben ser simples, transparentes y precisos en su formulación.

Deben poseer un enfoque lógico, asegurando que las instrucciones sean coherentes y conduzcan a resultados esperados.

Operan de manera secuencial y sistemática, lo que significa que las instrucciones se ejecutan en un orden establecido y predecible.

Cuentan con un punto de partida y un punto de conclusión claramente definidos, lo que garantiza que el proceso tenga un inicio y un fin determinados (Celi, 2023).

1.1.3 Fases para creación de Algoritmos

Las etapas de desarrollo de un algoritmo constan de tres fases claramente definidas:

Análisis. Durante esta fase, se lleva a cabo una evaluación minuciosa para definir de manera precisa el problema que se busca resolver. Se identifican los datos que constituirán la entrada del algoritmo, así como aquellos que deberán generarse como salida.

Diseño. Durante esta fase, se lleva a cabo la creación concreta del algoritmo. Se genera una secuencia de pasos lógicos y se define el conjunto de instrucciones requeridas para abordar eficazmente el problema en cuestión.

Prueba. La última fase involucra la verificación del algoritmo. Se pone a prueba su funcionalidad, observando si produce el

resultado esperado para una variedad de entradas posibles. Esto asegura que el algoritmo funcione de manera correcta y coherente en diferentes situaciones.

Todas estas etapas tienen una función fundamental en el proceso de desarrollo de un algoritmo, asegurando su correcta formulación y su capacidad para resolver problemas de manera efectiva (Sánchez, 2007).

1.1.4 Herramientas de un Algoritmo

PSeInt

PSeInt se ha diseñado con el objetivo principal de brindar apoyo a estudiantes en las primeras etapas de su aprendizaje en la creación de programas y algoritmos informáticos. El pseudocódigo se utiliza ampliamente como una introducción inicial, permitiendo a los estudiantes familiarizarse con conceptos fundamentales como el uso de estructuras de control, expresiones, variables y estructuras de datos. Todo esto se logra sin la necesidad de lidiar con las complejidades que conlleva la sintaxis de un lenguaje de programación real. El software tiene como finalidad simplificar la tarea de los principiantes al proporcionar una serie de herramientas y asistencias para la creación de algoritmos en este pseudolenguaje, además de ofrecer recursos adicionales que facilitan la detección de errores y la comprensión de la lógica subyacente en los algoritmos (Sachez et al., 2020).

Cuando se crea un algoritmo, la intención es ejecutarlo en una computadora. No obstante, para que la computadora pueda comprender los pasos necesarios para llevar a cabo el algoritmo, es necesario comunicar estos pasos a través de un conjunto de instrucciones y reglas que la computadora pueda entender. Estas instrucciones se agrupan en lo que se conoce como un lenguaje de programación. Posteriormente, estas instrucciones son procesadas (compiladas) por el compilador del programa para que la computadora pueda ejecutar el algoritmo de manera eficiente (Cruz-Barragán et al., 2019).

Un algoritmo que ha sido codificado utilizando un lenguaje de programación se denomina programa. Sin embargo, antes de adentrarse en el aprendizaje de un lenguaje de programación en sí, es esencial adquirir conocimientos en metodología de programación. Esto implica aprender la estrategia fundamental para abordar problemas mediante la creación de programas (Beúnes Cañete et al., 2019).

1.1.5 Ejemplos de aplicación

Crear un algoritmo el cual permita escribir en pantalla "Hola mundo Ramiro"

Figura 1 HolaMundoPSeInt

```
1  Proceso holaMundo
2      Escribir "Hola Mundo Ramiro";
3  FinProceso
```

1.2 Estructura secuencial

Cada algoritmo escrito en el pseudocódigo en PSeInt sigue la una estructura general la cual se muestra a continuación:

Comienza con la palabra reservada "Proceso," seguida por el nombre del programa. Luego, se presenta una serie de instrucciones organizadas en una secuencia y finaliza con la expresión "FinProceso." Esta secuencia de instrucciones se compone de una lista de directivas, cada una de las cuales termina con un punto y coma, como se ilustra en la Figura 2 (Cruz-Barragán et al., 2019).

Figura 2. Estructura General de un algoritmo

```
1  Proceso estructuraGeneral
2      accion ←1;
3      accion ←2;
4      accion ←3;
5      accion ←n;
6  FinProceso
```

1.2.1 Ejemplos de aplicación

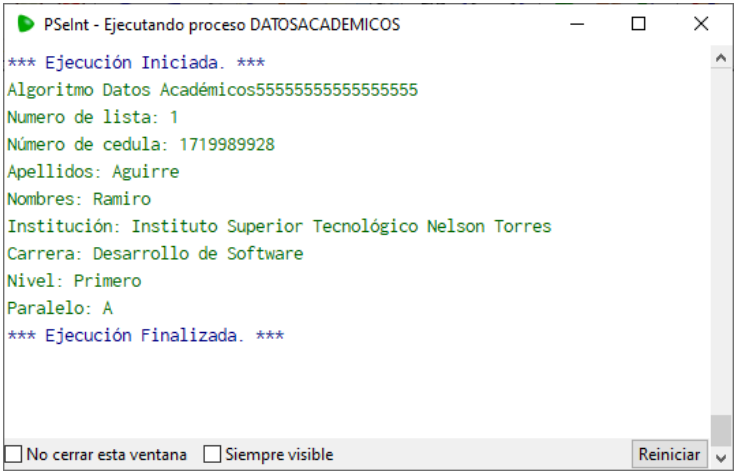
Crear un algoritmo el cual permita mostrar en pantalla los datos académicos de un estudiante tomando en cuenta los siguientes datos: número de lista, cedula, apellidos nombres, institución, carrera, nivel y paralelo.

Figura 3. Algoritmo datos Académicos

```
Proceso datosAcademicos
    Escribir "Algoritmo Datos Académicos55555555555555555";
    Definir cedula  Como Caracter;
    cedula ← "1719757278" ;
    Escribir "Numero de lista: 1";
    Escribir "Número de cedula: 1719989928";
    Escribir "Apellidos: Aguirre";
    Escribir "Nombres: Ramiro";
    Escribir "Institución: Instituto Superior Tecnológico Nelson Torres";
    Escribir "Carrera: Desarrollo de Software";
    Escribir "Nivel: Primero";
    Escribir "Paralelo: A";
FinProceso
```

Para ejecutar el algoritmo debe ir al menú ejecuta  o presionar la tecla de función F9. Esto permite que nuestro algoritmo se ejecute, a continuación se muestra un ejemplo en la siguiente imagen.

Figura 4. Ejecución del Algoritmo datos académicos



1.3 Variables y Constantes.

Los identificadores, que son los nombres asignados a las variables, deben estar formados únicamente por letras, números y/o guiones bajos (_), siendo obligatorio que inicien con una letra siempre.

1.3.1 Tipos de datos simples

Los tipos de datos simples se pueden clasificar en las siguientes categorías:

- Numérico: Este tipo de dato abarca números, tanto enteros como decimales. Los números decimales se representan utilizando el punto como separador. Ejemplos de datos numéricos son: 15, 25, 5, -15.5, 3.23.

- Lógico: Los datos de tipo lógico solo pueden tener dos valores posibles: TRUE (verdadero) o FALSE (falso), a menudo representados como CERO o UNO. Estos valores se utilizan para evaluar condiciones lógicas en un programa.

- Carácter: Los datos de tipo carácter representan caracteres individuales o cadenas de caracteres. Estas cadenas se encierran entre comillas, que pueden ser simples o dobles. Ejemplos de datos de tipo carácter son: 'Ramiro', "Hola mundo Ramiro", '150985', 'TRUE', 'ENTRE OTROS', "!#$%6789(/&HDFGB", entre otros.

Cada uno de estos tipos de datos tiene sus propias características y se utiliza según las necesidades específicas del programa.

Los tipos de datos simples se asignan automáticamente cuando se crean las variables. Estos tipos se derivan de las acciones de lectura (LEER) y asignación (<-). Por ejemplo, cuando realizamos la asignación "numeroUno <- 5;", estamos indicando de manera implícita que la variable "numeroUno" será de tipo numérico. Una vez que se determina el tipo de dato, este debe mantenerse constante durante toda la ejecución del proceso. Cualquier intento de cambiar este tipo de dato resultaría en la interrupción del proceso (García-Rodríguez & Dugarte-Peña, 2022).

1.3.2 Tipos de datos complejos (Estructurados)

Los arreglos o vectores son estructuras de datos homogéneas, esto significa que todos sus componentes pertenecen a un mismo tipo de dato. Estos permiten almacenar un conjunto definido de datos bajo un único nombre de identificación y, para poder acceder a estos datos utilizando uno o varios subíndices el cual generalmente inicia en cero. Los arreglos pueden adoptar diversas formas, como vectores unidimensionales o matrices multidimensionales. Su capacidad para organizar y gestionar datos de manera estructurada es fundamental en la programación y se utiliza para simplificar el acceso y la manipulación de información en programas informáticos.

Para poder usar un vector, es esencial llevar a cabo su proceso de dimensionamiento, esto implica que, al crear un arreglo,

es necesario definir sus límites o rangos de subíndices. Este paso es crucial, ya que determina la cantidad de elementos que se almacenarán en el arreglo y cómo se accederá a cada uno de ellos (Beúnes Cañete et al., 2019).

Dimensionamiento de Vectores, Arreglos, Arrays.

El comando "Dimensión" posibilita la definición del arreglo, especificando su dimensión.

Dimension <identificador> (<maxl>,...,<maxN>);

El dimensionamiento se refiere al proceso de definir o establecer el tamaño o la capacidad de un arreglo. Esto implica especificar cuántos elementos o posiciones contendrá el arreglo y cómo se pueden acceder a esos elementos a través de los índices. Por ejemplo, si tienes un arreglo unidimensional de tamaño 10, significa que puede contener 10 elementos, numerados del 0 al 9.

El dimensionamiento es una parte fundamental al trabajar con arreglos porque determina cuánta información se puede almacenar y cómo se organiza esa información en la memoria. El tamaño del arreglo generalmente se establece cuando se declara o se crea por primera vez y, en algunos lenguajes de programación, puede ser estático (no cambia durante la ejecución) o dinámico (puede cambiar en tiempo de ejecución).

Asegurarse de dimensionar correctamente los arreglos es esencial para evitar errores de desbordamiento de memoria o acceso a datos incorrectos.

Dimension <identificadorX> (<max5>,...,<maxX>),..., <identificadorY> (<maxV1>,...,<maxXY>) (García Perdomo et al., 2022).

1.3.3 Expresiones

Operadores

PSeInt incluye un conjunto elemental de operadores que pueden emplearse para formar expresiones, tanto simples como más elaboradas. Las tablas a continuación detallan todos los operadores presentes en este lenguaje simplificado:

Figura 5. Operadores PSeInt

Operador	Significado	Ejemplo
Relacionales		
>	Mayor que	3>2
<	Menor que	'ABC'<'abc'
=	Igual que	4=3
<=	Menor o igual que	'a'<='b'
>=	Mayor o igual que	4>=5
<>	Distinto que	Var1<>var2
Lógicos		
& ó Y	Conjunción (y).	(7>4) & (2=1) //falso
| ó O	Disyunción (o).	(1=1 | 2=1) //verdadero
~ ó NO	Negación (no).	~(2<5) //falso
Algebraicos		
+	Suma	total <- cant1 + cant2
-	Resta	stock <- disp – venta
*	Multiplicación	area <- base * altura
/	División	porc <- 100 * parte / total
^	Potenciación	sup <- 3.41 * radio ^ 2
% ó MOD	Módulo (resto de la división entera)	resto <- num MOD div

La prioridad de los operadores matemáticos en este contexto coincide con la del álgebra, pero se puede ajustar utilizando paréntesis según sea necesario. Con respecto a los operadores &

y |, se sigue una estrategia de "cortocircuito" en su evaluación. Esto significa que cuando dos expresiones están conectadas por el operador | y la primera resulta en Falso, o si están unidas mediante el operador | y la primera es Verdadero, la evaluación se interrumpe y la segunda expresión no se calcula, ya que no tendría impacto en el resultado (Educativo, 2016).

Funciones

Dentro del pseudocódigo, las funciones se utilizan de manera análoga a otros lenguajes. Se escribe el nombre de la función seguido de los argumentos necesarios, los cuales se colocan entre paréntesis (por ejemplo, exp(x), tan(x), mayusculas(S), minusculas(S), entre otras). Estas funciones pueden emplearse en cualquier expresión, y cuando la expresión se evalúa, la función se reemplaza por su resultado correspondiente. Hasta el momento, todas las funciones disponibles son de naturaleza matemática, lo que significa que generan un resultado de tipo numérico. Cada función toma un único parámetro numérico. A continuación, se enlistan las funciones integradas disponibles: (Huari Evangelista & José Novara, 2014).

Figura 6. Funciones PSeInt

Función	*Significado*
RC(X) o RAIZ(X)	Raíz Cuadrada de X
ABS(X)	Valor Absoluto de X
LN(X)	Logaritmo Natural de X
EXP(X)	Función Exponencial de X
SEN(X)	Seno de X
COS(X)	Coseno de X
TAN(X)	Tangente de X
ASEN(X)	Arcoseno de X
ACOS(X)	Arcocoseno de X
ATAN(X)	Arcotangente de X
TRUNC(X)	Parte entera de X
REDON(X)	Entero más cercano a X
AZAR(X)	Entero aleatorio en el rango [0;x-1]
ALEATORIO(A,B)	Entero aleatorio en el rango [A;B]
LONGITUD(S)	Cantidad de caracteres de la cadena S
MAYUSCULAS(S)	Retorna una copia de la cadena S con todos sus caracteres en mayúsculas
MINUSCULAS(S)	Retorna una copia de la cadena S con todos sus caracteres en minúsculas
SUBCADENA(S,X,Y)	Retorna una nueva cadena que consiste en la parte de la cadena S que va desde la posición X hasta la posición Y (incluyendo ambos extremos). Las posiciones utilizan la misma base que los arreglos, por lo que la primer letra será la 0 o la 1 de acuerdo al perfil del lenguaje utilizado.
CONCATENAR(S1,S2)	Retorna una nueva cadena resulta de unir las cadenas S1 y S2.
CONVERTIRANUMERO(X)	Recibe una cadena de caracteres que contiene un número y devuelve una variable numérica con el mismo.
CONVERTIRATEXTO(S)	Recibe un real y devuelve una variable numérica con la representación como cadena de caracteres de dicho real.

Expresiones Coloquiales

Si la opción de "Condicionales en lenguaje coloquial" está activada en las configuraciones del lenguaje, es posible utilizar algunas expresiones adicionales para formar condiciones, es decir, expresiones lógicas. En la tabla que sigue, se presentan ejemplos de expresiones, asumiendo que X e Y son variables definidas. También se proporciona la expresión equivalente en el lenguaje formal correspondiente o significado: (Del Prado & Lamas, 2014).

Figura 7. Expresiones Coloquiales PSeInt

Expresión	*Significado*
X ES Y	X=Y
X ES IGUAL A Y	X=Y
X ES DISTINTO DE Y	X<>Y
X ES MAYOR QUE Y	X>Y
X ES MENOR QUE Y	X<Y
X ES MAYOR O IGUAL A Y	X>=Y
X ES MENOR O IGUAL A Y	X<=Y
X ES CERO	X=0
X ES POSITIVO	X>0
X ES NEGATIVO	X<0
X ES PAR	X MOD 2 = 0
X ES IMPAR	X MOD 2 = 1
X ES MULTIPLO DE Y	X MOD Y = 0
X ES DIVISIBLE POR Y	X MOD Y = 0

1.4 Conclusión

Hemos explorado una serie de conceptos fundamentales en el ámbito del desarrollo de algoritmos y la programación. Comenzamos por comprender la importancia de la programación como un medio para resolver problemas y llevar a cabo tareas de manera eficiente mediante el diseño de algoritmos. Aprendimos cómo los algoritmos son estructuras lógicas secuenciales que se componen de pasos definidos, y cómo estos pasos pueden traducirse en programas utilizando lenguajes de programación.

Exploramos la utilidad del pseudocódigo, especialmente a través de herramientas como PSeInt, como un enfoque introductorio para crear algoritmos y comprender conceptos clave, como estructuras de control y expresiones. La estructura de un algoritmo en pseudocódigo de forma general, junto con la importancia de la metodología de programación y el uso de arreglos, se destacó como elementos cruciales en el proceso de desarrollo de programas.

Aprendimos sobre la jerarquía y los operadores matemáticos, y cómo estos operadores se pueden combinar para formar expresiones. También exploramos la incorporación de funciones matemáticas en el desarrollo de algoritmos, permitiendo cálculos complejos y análisis más profundos.

Finalmente, comprendimos que la programación no solo implica escribir código, sino también comprender la lógica subyacente en la resolución de problemas. La capacidad de desglosar problemas en pasos lógicos, tomar decisiones y diseñar soluciones eficaces es esencial para cualquier programador.

En conjunto, esta exploración nos ha proporcionado una base sólida para comprender los principios de programación de manera básica, al construir algoritmos y dentro del proceso de desarrollo de algoritmos o programas. Estos conocimientos son valiosos no solo para quienes buscan ingresar al mundo de la informática, sino también para cualquiera que desee abordar problemas de manera lógica y creativa en diversos contextos.

Autoevaluación Capitulo 1

1. ¿Qué importancia tiene la programación en la resolución de problemas y tareas eficientes?

a) Poca importancia

b) Importante

c) No tiene relevancia

2. ¿Cómo se definen los algoritmos y qué característica fundamental poseen?

a) Son cálculos matemáticos

b) Son secuencias lógicas de pasos

c) Son lenguajes de programación

3. ¿Qué es el pseudocódigo y cómo puede facilitar la comprensión y creación de algoritmos?

a) Un lenguaje de programación avanzado

b) Un lenguaje natural

c) Una representación intermedia entre el lenguaje humano y de programación

4. ¿Qué es PSeInt y cómo se utiliza para apoyar a los principiantes en programación?

a) Un lenguaje de programación de alto nivel

b) Un compilador de código C

c) Un software que ayuda a desarrollar algoritmos en pseudocódigo

5. Describe la estructura general de un algoritmo en pseudocódigo.

a) Inicia con "Inicio" y termina con "Fin"

b) Comienza con "Proceso" seguido del nombre del programa, luego sigue una secuencia de instrucciones y finaliza con "FinProceso"

c) No tiene una estructura definida

6. ¿Por qué es esencial comprender la metodología de programación antes de aprender un lenguaje de programación?

a) No es necesario comprenderla

b) Ayuda a escribir código más rápido

c) Establece una estrategia para resolver problemas antes de abordar la sintaxis de un lenguaje

7. ¿Qué son los arreglos y cómo se dimensionan en PSeInt?

a) Son estructuras de datos con elementos de diferentes tipos

b) Son secuencias de instrucciones en un algoritmo

c) Son estructuras de datos homogéneas y se dimensionan con la instrucción "Dimensión"

8. Enumera y describe los tres tipos de datos básicos disponibles en PSeInt.

a) Numérico, Carácter y Booleano

b) Texto, Entero y Real

c) Numérico, Lógico y Carácter

9. ¿Cómo se establece la jerarquía de operadores matemáticos en PSeInt? ¿Qué es el "cortocircuito" en relación con los operadores & y |?

a) No hay jerarquía de operadores en PSeInt

b) Se establece mediante el uso de paréntesis

c) Sigue la jerarquía del álgebra y se puede alterar con paréntesis; el "cortocircuito" implica que en expresiones con & y |, si la primera parte determina el resultado, la segunda no se evalúa

10. ¿Qué son las funciones en PSeInt y cómo se utilizan en las expresiones?

a) Son estructuras de control

b) Son operaciones predefinidas que procesan valores y devuelven resultados

c) Son nombres de variables

11.¿Qué aspectos fundamentales se deben comprender para desarrollar programas y algoritmos eficaces?

a) No es necesario comprender nada

b) La estructura de las bases de datos

c) La estructura de algoritmos, la metodología de programación, los tipos de datos, operadores y funciones

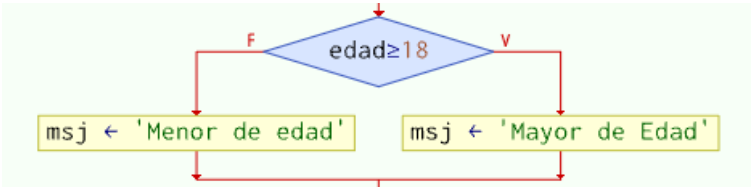
Capítulo 2
Estructura de control selectiva

2.1 Estructura de control selectiva

Estructura Condicional "Si-Entonces"

La secuencia de pasos llevados a cabo por la estructura "Si-Entonces-SiNo" se encuentra influenciada por el valor de una condición lógica (Educativo, 2016).

Figura 8. Diagrama de Estructura "Si-Entonces"



Cuando se activa esta estructura, se realiza una evaluación de la condición y se ejecutan las instrucciones correspondientes dependiendo de si la condición es verdadera o falsa. Si la condición es verdadera, se ejecutan las instrucciones que siguen a "Entonces", y si la condición es falsa, se ejecutan las directivas que continúan después de "SiNo". La condición debe expresarse como una expresión lógica, cuya evaluación produce un resultado de Verdadero o Falso (García-Rodríguez & Dugarte-Peña, 2022).

Figura 9. Pseudocódigo Estructura "Si-Entonces"

```
Si edad≥18 Entonces
    msj  ← "Mayor de Edad";
SiNo
    msj ←  "Menor de edad";
FinSi
```

Es fundamental que la cláusula "Entonces" esté siempre presente, aunque la cláusula "SiNo" podría no ser requerida. En tal situación, si la condición es falsa, no se realiza ninguna ejecución de instrucciones y el flujo del programa continúa con la instrucción subsiguiente.

2.1.1 Ejemplos de aplicación.

Crear un algoritmo el cual permita el ingreso de la edad de un estudiante y determinar si esta corresponde a una persona "Mayor o Menor de Edad". Hay que considerar que para ser mayor de edad esta debe ser igual o mayor a 18 años.

Figura 10. Algoritmo Mayor Menor de Edad.

```
1  Proceso mayorMenorEdad
2      Definir edad Como Entero;
3      Definir msj Como Caracter;
4      Escribir "Algoritmo - Mayor y Menor de Edad";
5      Escribir "Algoritmo Mayor de edad";
6      Escribir "Ingrese su edad";
7      leer edad;
8      Si edad≥18 Entonces
9          msj  ← "Mayor de Edad";
10     SiNo
11         msj ←  "Menor de edad";
12     FinSi
13     Escribir edad, " Corresponde a: ",msj;
14 FinProceso
```

Figura 11. Ejecución del proceso Mayor Menor de Edad 1.

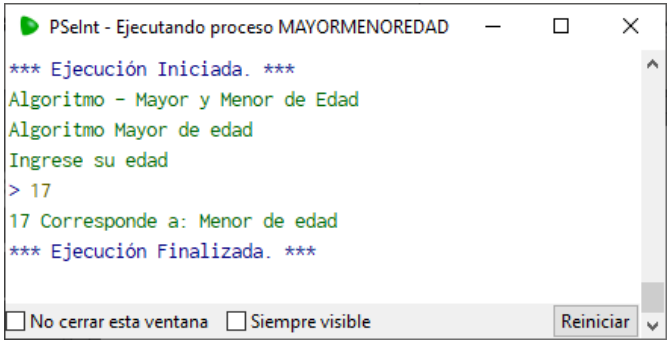
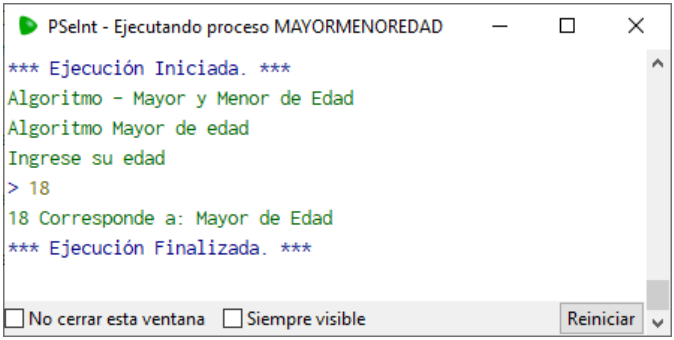


Figura 12. Ejecución del proceso Mayor Menor de Edad 2.



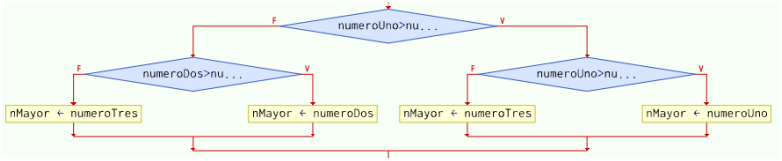
2.1.2 Estructura de control selectiva anidada.

Existe situaciones en las cuales hay que validar comparar mas de una condición por lo que al momento de programar existe la posibilidad de anidar la estructura lo cual consiste en poner una estructura Si dentro de otra (Del Prado & Lamas, 2014).

Figura 13. Ejemplo Si Anidado.

```
Si numeroUno > numeroDos Entonces
    si numeroUno > numeroTres Entonces
        nMayor ← numeroUno;
    SiNo
        nMayor ← numeroTres;
    FinSi
SiNo
    si numeroDos > numeroTres Entonces
        nMayor ← numeroDos;
    SiNo
        nMayor ← numeroTres;
    FinSi
FinSi
```

Figura 14. Si Entonces anidada.



2.1.3 Ejemplo de aplicación

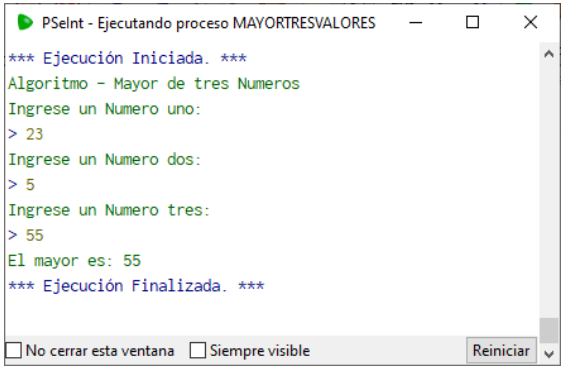
A continuación, se presenta un ejemplo:

Crear un algoritmo el cual le permita el ingreso de tres valores numéricos enteros y determinar el mayor de ellos.

Figura 15. Mayor de Tres Valores.

```
Proceso mayorTresValores
    Definir numeroUno, numeroDos, numeroTres, nMayor Como Entero;
    Definir msj Como Caracter;
    Escribir "Algoritmo - Mayor de tres Numeros ";
    Escribir "Ingrese un Numero uno:";
    Leer numeroUno;
    Escribir "Ingrese un Numero dos:";
    Leer numeroDos;
    Escribir "Ingrese un Numero tres:";
    Leer numeroTres;
    Si numeroUno > numeroDos Entonces
        si numeroUno > numeroTres Entonces
            nMayor ← numeroUno;
        SiNo
            nMayor ← numeroTres;
        FinSi
    SiNo
        si numeroDos > numeroTres Entonces
            nMayor ← numeroDos;
        SiNo
            nMayor ← numeroTres;
        FinSi
    FinSi
    Escribir "El mayor es: ", nMayor;
```

Figura 16. Ejecución algoritmo Mayor de tres números.



En el ejemplo se puede evidenciar como al momento de comparar si el numeroUno es mayor que el numeroDos, es necesario también compararlo con el numeroTres, para poder determinar

cual es el mayor de los tres. Es ahí cuando es necesario usar un "Si-Entonces anidado"

2.2 Estructura de control de Selección Múltiple

La serie de pasos realizados por una instrucción "Segun" se encuentra determinada por el valor de una variable numérica.

Figura 17. Estructura Segun – diagrama.

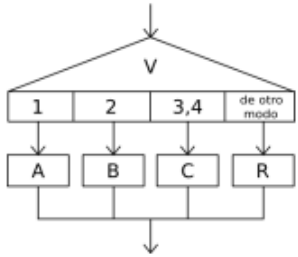


Figura 18. Estructura Según.

```
Segun <variable> Hacer
      <número1>: <instrucciones>
      <número2>,<número3>: <instrucciones>
      <...>
      De Otro Modo: <instrucciones>
FinSegun
```

Esta instrucción brinda la oportunidad de ejecutar diversas acciones en función del valor almacenado en una variable numérica. Al llevarse a cabo, se procede a evaluar el contenido de dicha variable y se ejecuta la secuencia de instrucciones relacionada con ese valor específico.

Cada opción consiste en una lista de números separados por comas, seguidos de dos puntos y un conjunto de instrucciones asociadas. Si el valor de la variable coincide con alguno de los números en la lista de una opción que contiene múltiples números, se ejecutarán las instrucciones correspondientes a esa opción.

Cada opción consiste en una lista de números separados por comas, seguidos de dos puntos y un conjunto de instrucciones asociadas. Si el valor de la variable coincide con alguno de los números en la lista de una opción que contiene múltiples números, se ejecutarán las instrucciones correspondientes a esa opción (Ronald & Ayquipa, n.d.).

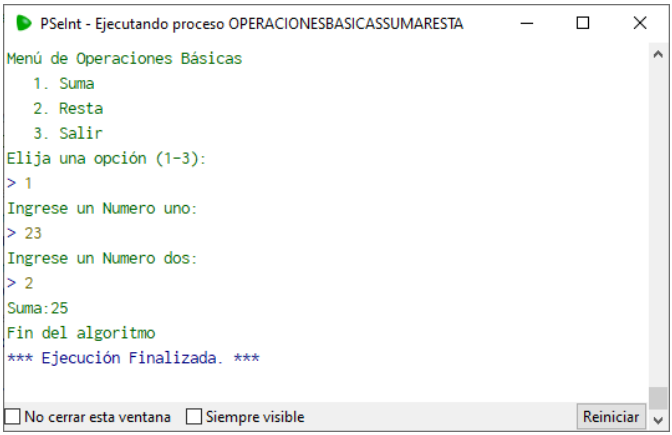
2.2.1 Ejemplos de aplicación.

Crear un algoritmo el cual permita seleccionar entre tres opciones 1 para suma, 2 para resta y 3 para salir. Si se selecciona la opción 1 el algoritmo debe permitir la suma de dos valores. Si se selecciona la opción 2 el algoritmo debe permitir la resta de dos valores. Si se selecciona la opción 3 el algoritmo debe permitir terminar el algoritmo enviando un mensaje que diga “Fin del algoritmo”.

Figura 19. Algoritmo Operaciones Básicas.

```
Algoritmo operacionesBasicas
    Definir numeroUno, numeroDos, OP Como Entero;
    // mostrar menu
    Limpiar Pantalla;
    Escribir "Menú de Operaciones Básicas";
    Escribir "   1. Suma";
    Escribir "   2. Resta";
    Escribir "   3. Salir";
    // ingresar una opcion
    Escribir "Elija una opción (1-3): ";
    Leer OP;
    //Leer los valores de las variables
    Escribir "Ingrese un Numero uno:";
    Leer numeroUno;
    Escribir "Ingrese un Numero dos:";
    Leer numeroDos;
    // procesar esa opción
    Segun OP Hacer
        1:
            Escribir "Suma:", (numeroUno + numeroDos);
        2:
            Escribir "Resta:", (numeroUno - numeroDos);
        3:
            Escribir "Gracias, vuelva pronto";
        De otro modo:
            Escribir "Opción no válida ",OP;
    FinSegun
    Escribir "Fin del algoritmo";
    Esperar Tecla;
FinAlgoritmo
```

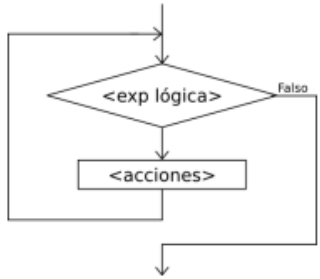
Figura 20. Ejecución del algoritmo Menú Operaciones Básicas.



2.3 Estructura repetitiva Mientras y Para

La instrucción Mientras.

Figura 21. Instrucción Mientras.



Desencadena una serie de instrucciones mientras una condición se mantenga verdadera.

Figura 22. Sintaxis Mientras.

```
Mientras expresion_logica Hacer
    secuencia_de_acciones
FinMientras
```

Al utilizar esta instrucción, primero se evalúa la condición. Si la condición es verdadera, las instrucciones dentro del cuerpo del ciclo se ejecutan una vez. Luego de finalizar la ejecución del cuerpo del ciclo, la condición se vuelve a evaluar. Si la condición sigue siendo verdadera, el ciclo se repite y las instrucciones se ejecutan nuevamente. Este proceso se repite continuamente mientras la condición siga siendo verdadera.

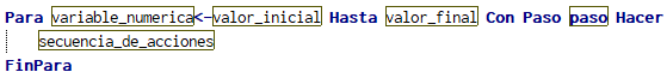
Es relevante notar que las instrucciones dentro del cuerpo del ciclo podrían no ejecutarse si, al evaluar la condición por primera vez, esta resulta ser falsa.

En caso de que la condición permanezca siempre verdadera, esta instrucción resultará en un bucle infinito. Para prevenirlo, es necesario que las instrucciones dentro del ciclo incluyan acciones que alteren las variables asociadas a la condición. De esta manera, la condición eventualmente se volverá falsa y el ciclo se detendrá. (Iván García Santillán, 2014).

Bucles "Para"

La instrucción "Para" realiza una serie de instrucciones un número específico de veces.

Figura 23. Sintaxis Para.



Dentro del bloque, la variable "<nombreVariable>" se inicializa con el valor "<valorInicial>" y luego se ejecutan las instrucciones en el cuerpo del ciclo. A continuación, se incrementa la variable "<nombreVariable>" en "<pasoIncremento>" unidades y se verifica si el valor contenido en "<nombreVariable>" ha alcanzado o superado "<valorFinal>". Si esta afirmación es falsa, el proceso se repite hasta que "<nombreVariable>" sea mayor que "<valorFinal>". Si se omite la cláusula "Con Paso <pasoIncremen-

to>", la variable "<nombreVariable>" se incrementará en 1 (Duque, 2021).

2.3.1 Ejemplos de aplicación.

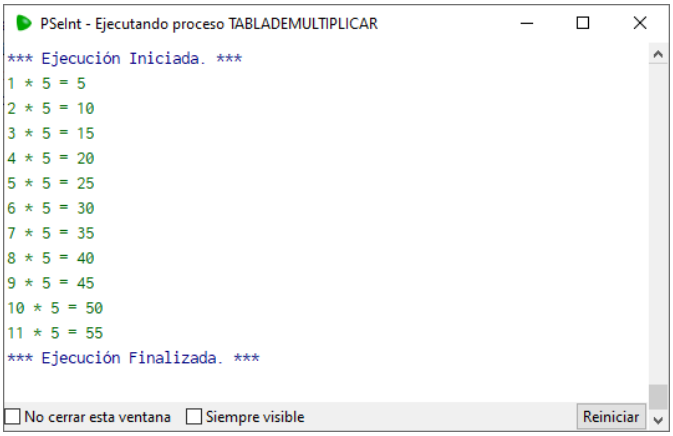
Usando la sentencia MIENTRAS, crear un algoritmo que permita imprimir en pantalla la tabla de multiplicar del 5 desde iniciando en 1 hasta llegar a 10, al ejecutar se deberá mostrar de la siguiente manera:

1 * 5 = 5
2 * 5 = 10
3 * 5 = 15
4 * 5 = 20
5 * 5 = 25
6 * 5 = 30
7 * 5 = 35
8 * 5 = 40
9 * 5 = 45
10 * 5 = 50

Figura 24. Algoritmo tabla de multiplicar Mientras.

```
1  Proceso tablaDeMultiplicarMientras
2      Definir  i Como Entero;
3      i ← 0;
4      Mientras i≤10 Hacer
5          i ← i+1;
6          Escribir i," * 5 = ",i*5;
7      FinMientras
8  FinProceso
```

Figura 25. Ejecución de la Tabla de multiplicar – Mientras.



Usando la sentencia PARA, crear un algoritmo que permita imprimir en pantalla la tabla de multiplicar del 5 desde iniciando en 1 hasta llegar a 10.

Figura 26. Algoritmo Tabla de Multiplicar Para

```
1  Proceso tablaDeMultiplicarPara
2      Definir  i Como Entero;
3      Para i←1 Hasta 10 Con Paso 1 Hacer
4          Escribir i," * 5 = ",i*5;
5      FinPara
6  FinProceso
```

Figura 27. Diagrama de Flujo Tabla de Multiplicar.

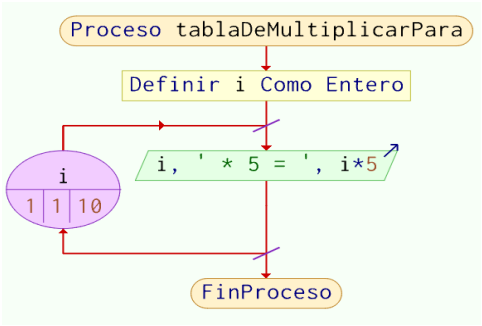
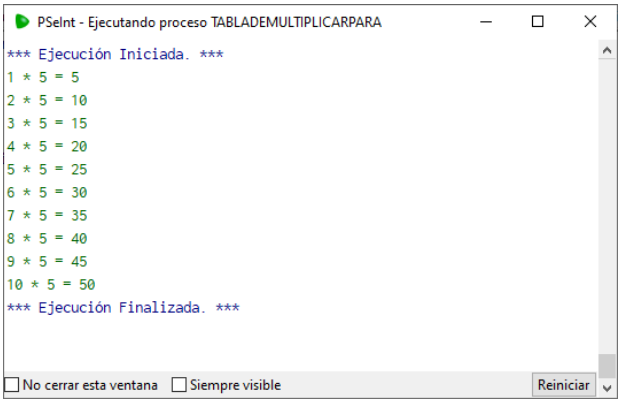


Figura 28. Ejecución de la Tabla de multiplicar – Para.



2.4 Conclusión

En este capítulo, hemos examinado los principios esenciales vinculados al manejo del flujo de ejecución y las estructuras de control en el ámbito de la programación. Hemos adquirido una comprensión profunda de cómo las sentencias condicionales, como "Si-Entonces" y "Segun", posibilitan que un programa tome elecciones basadas en condiciones lógicas. Asimismo, hemos explorado cómo los bucles "Mientras" y "Para" habilitan la repetición eficaz de una secuencia de comandos.

La instrucción "Si-Entonces" nos ha permitido ejecutar acciones alternativas según el resultado de una condición. Por otro lado, la instrucción "Segun" nos ha mostrado cómo realizar múltiples opciones basadas en el valor de una variable. Asimismo, hemos visto cómo los bucles "Mientras" y "Para" nos permiten controlar la repetición de instrucciones, ya sea mientras se cumpla una condición o un número predeterminado de veces.

Es esencial comprender estas estructuras de control, ya que forman la base para la creación de programas eficientes y soluciones lógicas a problemas complejos. Además, hemos aprendido a evitar trampas comunes, como los bucles infinitos, al asegurarnos de que las condiciones se modifiquen en el cuerpo del bucle.

En el siguiente capítulo, ampliaremos nuestro conocimiento al explorar la manipulación de datos y la interacción con el usuario. Estos conceptos nos permitirán crear programas más interactivos y versátiles. ¡Continuemos avanzando en nuestro viaje de aprendizaje de la programación!

Recuerda que la comprensión sólida de las estructuras de control es esencial para construir algoritmos y programas que funcionen de manera eficiente y cumplan con los requisitos del problema. Con estos conocimientos, estamos listos para enfrentar desafíos más complejos en el mundo de la programación.

Autoevaluación Capitulo 2

1. ¿Qué determina la secuencia de instrucciones ejecutadas en la instrucción "Si-Entonces-SiNo"?

a) Una variable numérica
b) Una variable lógica
c) Una expresión matemática

1. **¿Qué se evalúa en la instrucción «Mientras» antes de ejecutar el cuerpo del ciclo?**

a) Una condición numérica
b) Una condición lógica
c) Una expresión matemática

2. **La instrucción "Segun" se utiliza para realizar:**

a) Repetición de instrucciones
b) Toma de decisiones
c) Evaluación de expresiones matemáticas

3. **¿Qué ocurre en la instrucción «Para» después de ejecutar el cuerpo del ciclo?**

a) Se incrementa la variable y se verifica la condición
b) Se evalúa la condición y se incrementa la variable
c) Se decrementa la variable y se verifica la condición

4. **En la instrucción "Si-Entonces-SiNo", ¿qué sucede si la condición es falsa y no se incluye la cláusula "SiNo"?**

a) Se ejecuta la secuencia de instrucciones siguiente
b) No se ejecuta ninguna instrucción
c) Se repite el ciclo hasta que la condición sea verdadera

5. **¿Qué papel juega la cláusula «Con Paso <paso>» en la instrucción «Para»?**

   a) Incrementa la variable en 1 automáticamente

   b) Indica cuánto se incrementa la variable en cada repetición

   c) Define el valor inicial de la variable

6. **¿Cuál es el propósito de la instrucción «Segun» en términos de control de flujo?**

   a) Evaluar una condición

   b) Repetir una secuencia de instrucciones

   c) Realizar diferentes acciones basadas en el valor de una variable

7. **En la instrucción "Mientras", ¿qué sucede si la condición es falsa desde el principio?**

   a) Se ejecuta el cuerpo del ciclo al menos una vez

   b) No se ejecuta ninguna instrucción

   c) Se repite el ciclo indefinidamente

8. **¿Qué se debe hacer para evitar un ciclo infinito en la instrucción «Para»?**

   a) No incluir la cláusula "Con Paso <paso>"

   b) No modificar la variable dentro del ciclo

   c) Asegurarse de que las instrucciones dentro del ciclo modifiquen la variable para cambiar la condición

9. **¿Cuál es el propósito principal de las estructuras de control en programación?**

   a) Incrementar la velocidad del programa

   b) Facilitar la depuración del código

   c) Controlar el flujo de ejecución y tomar decisiones en función de condiciones

Capítulo 3
El lenguaje y su entorno integrado de desarrollo

3.1 El lenguaje y su entorno integrado de desarrollo.

3.1.1 Introducción al lenguaje y a su entorno de desarrollo.

El paradigma de la Programación Orientada a Objetos.

Los Lenguajes Orientados a Objetos se basan en la representación de conceptos a través de objetos. Estos objetos representan entidades del mundo real y poseen atributos cruciales para abordar las situaciones en las que se encuentran. Por ejemplo, una persona puede tener atributos como nombre, fecha de nacimiento y nivel de educación, mientras que un vehículo podría contar con atributos como propietario, número de matrícula y fecha de registro (Rodríguez Barrera, 2019).

Lograr programas exentos de errores conlleva una importancia considerable hacia la facilidad de depuración dentro del entorno o lenguaje que empleemos. Una vez que los programas han sido construidos, se torna imperativo mantenerlos, lo que a su vez podría requerir ajustar o ampliar las funcionalidades de los objetos. La aptitud para conservar un programa se vincula directamente con la estructura sólida de sus tipos, la encapsulación de los conceptos pertinentes, la viabilidad de la reutilización y la magnitud de software reutilizado, así como la comprensibilidad del mismo. Todas estas cualidades resultan más alcanzables al utilizar lenguajes orientados a objetos, en especial Java (Fernández, 2014).

Modularidad: se refiere a la cualidad mediante la cual un programa informático se configura a partir de segmentos independientes a los que denominamos módulos. La modularidad resulta esencial para la escalabilidad y el entendimiento de los programas, además de generar ahorro en esfuerzo y tiempo durante el proceso de desarrollo. En Java, las entidades modulares principales son las clases, que pueden ser integradas junto a las interfaces, formando unidades modulares de mayor envergadura conocidas como paquetes (de programación & Miguel Toro Bonilla, 2022).

Encapsulación: representa la habilidad de ocultar los pormenores de la implementación. En particular, esto abarca los detalles de implementación del estado de un objeto y las particularidades de la implementación de los métodos. En Java, esta capacidad se logra mediante la distinción entre interfaces y clases, así como mediante la declaración de atributos privados en las clases, lo cual será discutido con más detalle más adelante. Los usuarios de un objeto, es decir, otros objetos que interactúan con él, lo hacen a través del contrato presentado (Villalobos & Casallas Robby, n.d.).

Reutilización: Después de crear e implementar una clase de objetos, otros desarrolladores pueden utilizarla sin necesidad de conocer los aspectos internos de su implementación. Existen varias técnicas de reutilización disponibles, como la herencia y la delegación, las cuales serán analizadas en profundidad en secciones posteriores.

Claridad de comprensión o legibilidad: hace referencia a la facilidad con la que se puede comprender un programa. La legibilidad de un programa mejora cuando está organizado en módulos adecuados y se ha aplicado la encapsulación de manera adecuada (Toro Bonilla, 2022).

Objetos.

Los objetos poseen una identidad definida, propiedades particulares, un estado específico y una funcionalidad inherente.

Cada objeto ostenta una identidad que lo singulariza y lo diferencia de los demás objetos del mismo tipo. Aunque varios objetos puedan compartir un estado similar, cada uno retiene su propia identidad; en tales situaciones, se describe que los objetos son equivalentes, pero no idénticos (Peñarrieta, 2011).

- Las propiedades representan las características visibles de un objeto desde su entorno externo. Pueden abarcar distintos tipos, como números enteros, reales, texto, booleanos, entre otros.

- El estado refleja el valor actual de sus propiedades en un instante determinado.

- La funcionalidad de un objeto se presenta a través de un conjunto de métodos. Estos métodos interactúan con el estado del objeto (pudiendo consultar o alterar las propiedades) y constituyen el medio de comunicación entre el objeto y su entorno exterior.

3.1.2 Clases

Las clases representan las entidades fundamentales en la Programación Orientada a Objetos que posibilitan la especificación de los aspectos internos de un objeto (a través de los atributos), la obtención de las propiedades de los objetos basándose en los atributos y la implementación de las capacidades proporcionadas por los objetos (mediante los métodos). Cada clase establece un tipo distinto que puede ser empleado para declarar variables de esa clase y crear objetos correspondientes (Mart & De Guevara, n.d.).

A continuación se muestra un ejemplo de la estructura básica de una clase.

Figura 29. Estructura básica de una clase.

```
public class conceptosBasicos {


}
```

En el ejemplo se define una clase llamada conceptosBasicos, la cual esta vacía.

Por lo general, al construir una clase, comenzamos con un diseño del tipo. Al diseñar un nuevo tipo, es esencial basarse en los tipos ya existentes. Se requiere tomar decisiones sobre qué tipos existentes se extenderán o usarán. Se dice que el nuevo tipo utiliza aquellos tipos con los que declara sus propiedades. Asimismo, es posible diseñar el nuevo tipo ampliando algunos de los tipos disponibles. Se establece una relación de extensión entre

tipos cuando uno de ellos incorpora funcionalidad adicional, es decir, nuevas propiedades y métodos, al otro.

Para esta guía se va a tomar en cuenta únicamente los métodos que nos permitirá resolver problemas de bajo nivel de complejidad. A continuación se presenta un ejemplo.

Figura 30. Clase con método hola Mundo.

```
public class conceptosBasicos {

    public static void holaMundo(){
        System.out.println("Hola Mundo");
    }
}
```

En el ejemplo se define un método tipo estático vacío el cual se llama holaMundo.

Entorno de desarrollo

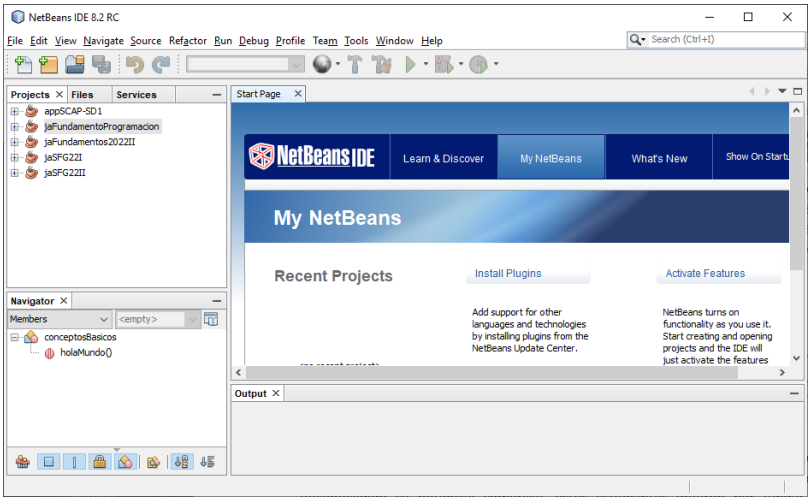
Java no dispone de su propio entorno de desarrollo integrado (IDE). Por ello, es posible utilizar desde herramientas sencillas como un bloc de notas hasta entornos de desarrollo avanzados como NetBeans. NetBeans se destaca como un entorno de desarrollo de gran potencia que permite la creación de aplicaciones complejas con características como interacción web, diagramas UML, acceso a bases de datos, desarrollo de aplicaciones web, aplicaciones móviles y hasta implementaciones de Inteligencia Artificial (IA). Debido a su eficacia, nos enfocaremos en la programación de aplicaciones dentro de este IDE (Toro Bonilla, 2022).

NetBeans 8.2 RC

NetBeans, que tuvo sus inicios en 1996 como un proyecto estudiantil en la República Checa bajo el nombre original de Xelfi, fue desarrollado en colaboración con la Facultad de Matemáticas y Física de la Universidad Carolina en Praga. El objetivo principal era crear un entorno de desarrollo integrado (IDE) para Java que se inspirara en las características de Delphi. Xelfi marcó un hito al convertirse en el primer IDE escrito en Java y lanzó su primer prelanzamiento en 1997. La iniciativa atrajo suficiente atención, lo que motivó a los estudiantes a continuar con el proyecto incluso después de su graduación. Reconociendo su potencial comercial, estos estudiantes optaron por fundar una compañía en torno a este proyecto, utilizando espacios web proporcionados por amigos y familiares. La mayoría de ellos aún están involucrados en el desarrollo y evolución de NetBeans (Peñarrieta, 2011).

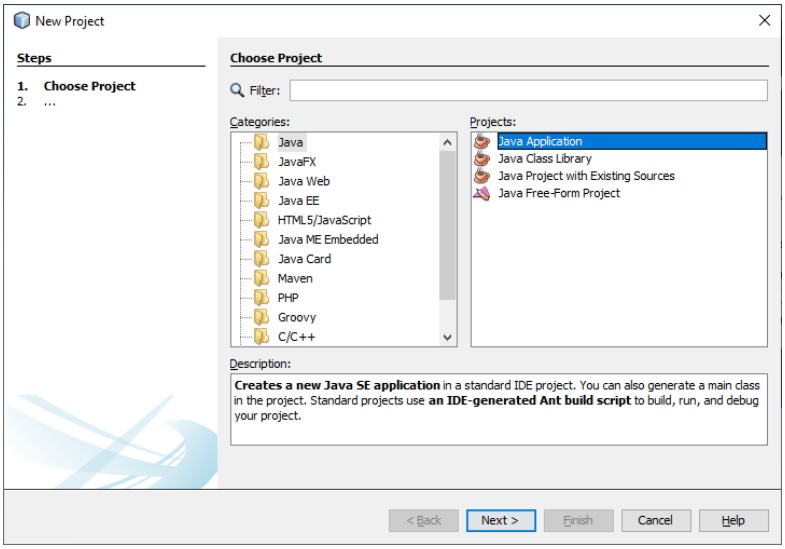
La versión con la que se realiza la guía es la versión NetBeans 8.2 RC. A continuación se presenta una captura:

Figura 31. IDE NetBeans 8.2 RC.



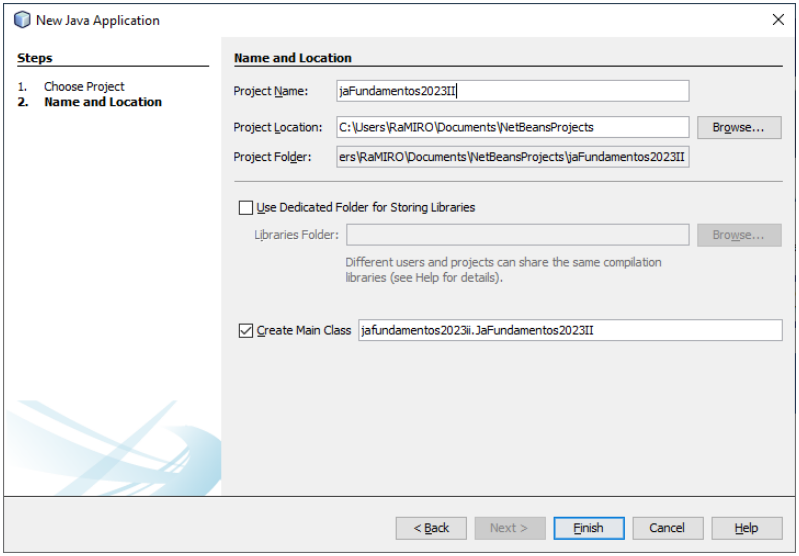
Para iniciar a trabajar dentro del IDE, es necesario crear un nuevo proyecto para lo cual debemos ir a *file/new Project*, para esto se iniciara el *wizard*.

Figura 32. New Project.



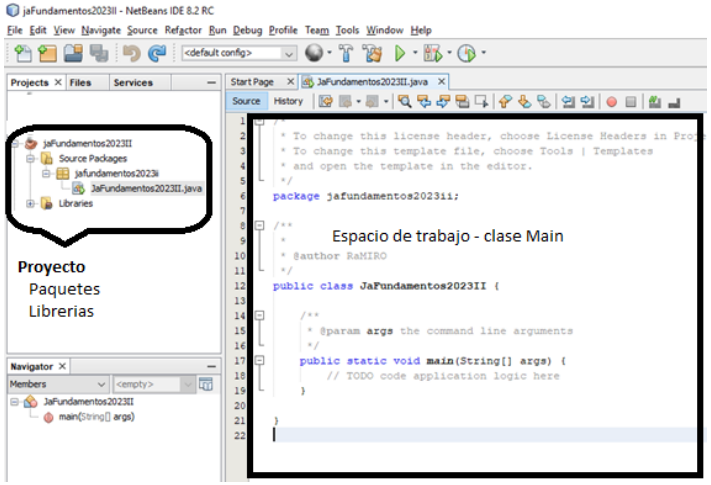
En esta ventana nos ubicamos en la categoría Java y elegimos *Java Aplication* y a continuación *Next*.

Figura 33. New Java Application–Nombre y Ubicación.



Aquí debemos en primera instancia escribir el nombre del proyecto y no menos importante especificar la ubicación de este, luego hay que dar clic en *Finish.*

Figura 34. Proyecto–Clase Main.



Se debe crear tres paquetes llamados: modelo, vista y controlador. Los cuales nos permiten organizar nuestro código. Para esto debemos dar clic derecho sobre Source Packages/new/java Package

Figura 35. Menú nuevo Package.

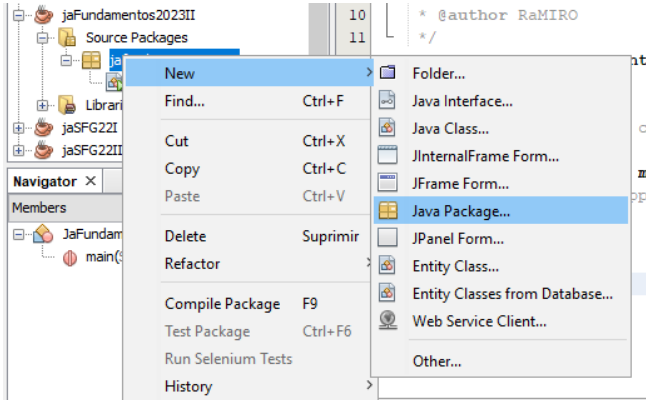
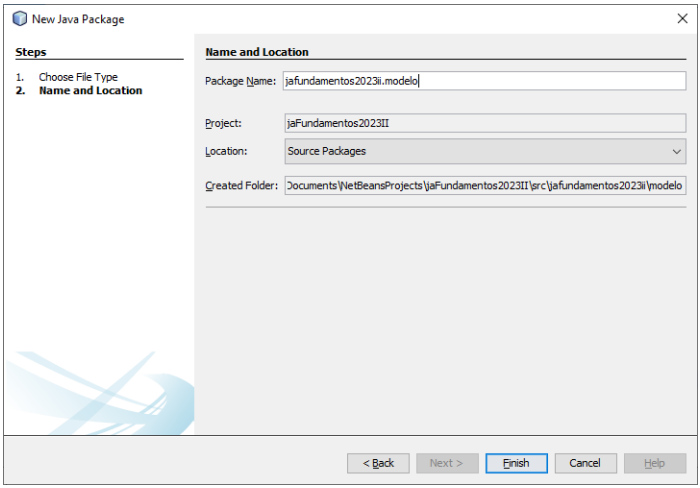
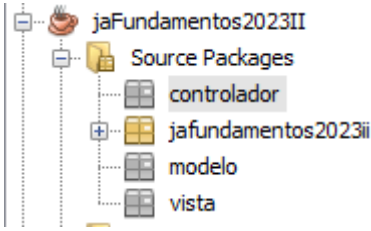


Figura 36. Nuevo Package – nombre y ubicación.



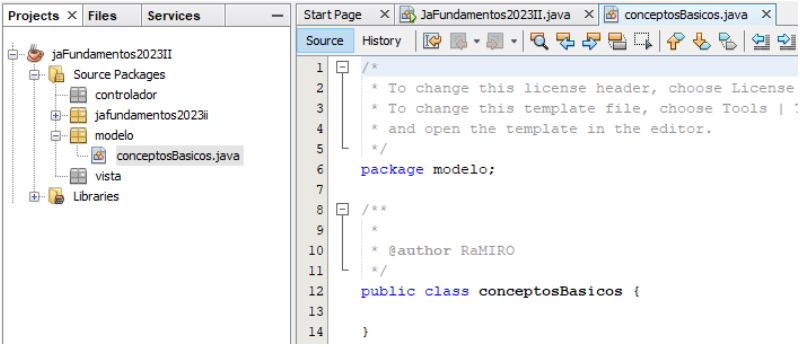
De esta manera se debe crear los paquetes los cuales deben quedar de la siguiente forma:

Figura 37. Proyecto jaFundamentos2023I.



Para crear una nueva clase damos clic sobre el paquete de modelo seleccionamos *new/java Class*… ponemos el nombre conceptosBasicos y damos clic en *Finish*.

Figura 38. Clase conceptosBasicos.



3.1.3 Funciones Vacías, Funciones estáticos, Entrada/ Salida de Datos

Métodos

Operaciones o acciones ofrecidas por un objeto derivado de una clase. Un método constituye una abstracción de una operación ejecutable sobre un objeto. Dentro de una clase, es posible declarar múltiples métodos que ejecuten diversas operaciones con los objetos. Un método se compone de un conjunto de instrucciones incorporadas en una clase, las cuales desempeñan una tarea específica y pueden ser convocadas por su nombre correspondiente (Sánchez, 2007).

Métodos de tipo void

La función principal de los métodos de tipo void reside en que son métodos que no producen ningún tipo de resultado o retorno.

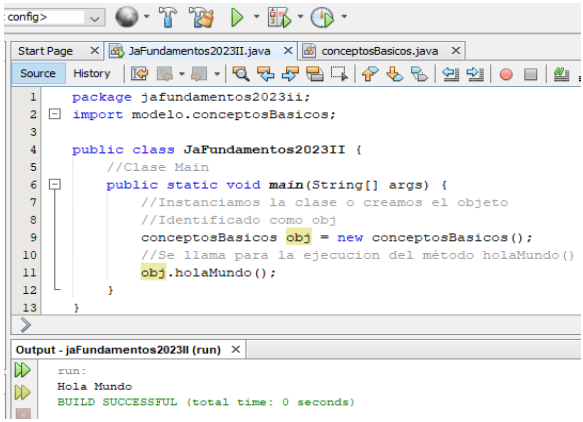
Figura 39. ConceptosBasicos.holaMundo().

```
package modelo;

/**
 *
 * @author RaMIRO
 */
public class conceptosBasicos {
    public void holaMundo(){
        System.out.println("Hola Mundo");
    }
}
```

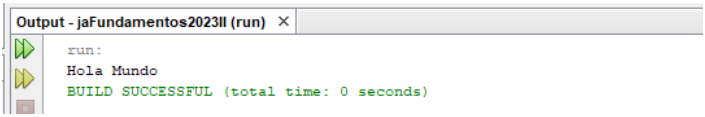
En el ejemplo se declara un método tipo void llamado holaMundo(), el cual imprime en pantalla "Hola Mundo", un factor a considerar es que para hacer uso de la clase conceptosBasicos se debe importar la clase a través del comando import seguido del nombre del paquete y a continuación el nombre de la clase. Para ejecutar este método es necesario ir a la clase Main y crear un objeto el cual permita llamar a los métodos que se encuentran dentro de la clase conceptosBasicos tal como se muestra a continuación:

Figura 40. Ejecución del método holaMundo desde un objeto.



Es importante mencionar que para ejecutar la clase debemos ir al menú *Run/Run Project* o presionar la tecla *F6 o dar clic en el icono*  . Para visualizar los resultados, se debe hacer en el apartado denominado *Output.*

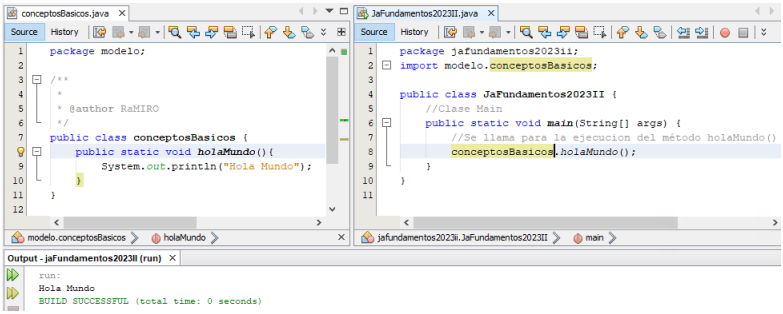
Figura 41. Output holaMundo.



Static

Los elementos estáticos, también conocidos como miembros de clase, son aquellos que están asociados a la clase en sí misma en lugar de pertenecer a una instancia específica de la clase. Un ejemplo ilustrativo de esta definición es el mismo método holaMundo(), el cual lo podemos definir como static para poder llamarlo sin tener la necesidad de instanciar la clase o crear un objeto de la clase (Perez, n.d.).

Figura 42. Ejecución del método tipo static holaMundo.



En la figura podemos ver en primera instancia que al método holaMundo() se lo define como de tipo static, esto permite llamarlo directamente desde la clase Main con el nombre de la clase seguido por el nombre del método. Esto se puede realizar debido a que el método ahora le no le pertenece al objeto y si a la clase. Al ejecutar podemos evidenciar en el output que se ejecuta de forma esperada.

3.2 Variables o Identificadores

Las variables, también conocidas como identificadores, son términos esenciales que permiten hacer referencia a los distintos elementos que conforman el código. En esencia, actúan como nombres para denotar clases, interfaces, métodos, atributos, variables, parámetros y paquetes. Para designar a estos componentes, es fundamental adherirse a ciertas normativas para que puedan ser interpretados adecuadamente por el compilador (Cairo, n.d.).

La construcción de los identificadores involucra una secuencia de letras, dígitos o los símbolos "_" y "$". No obstante, es crucial considerar las siguientes directrices:

- No deben coincidir con las palabras reservadas de Java.
- Deben dar inicio con una letra, "_", o "$", si bien se aconseja evitar estos dos últimos.
- No tienen restricción en cuanto a longitud.

- Es case sensitive esto significa que distingue el nombre de las variables entre mayúsculas y minúsculas. Por ejemplo, el identificador “aux” se diferencia de “AUX” o “Aux”.

Convenciones de Codificación

Las convenciones de codificación representan un conjunto de normativas diseñadas para un lenguaje de programación particular, con el propósito de recomendar estilos de programación, prácticas recomendables y enfoques para mantener la apariencia del código fuente. Estas pautas son recomendadas a los desarrolladores de software para mejorar la legibilidad del código y simplificar el mantenimiento del software (Bohada Jaime et al., 2019).

camelCase

El término “CamelCase” se deriva de la semejanza de las mayúsculas a lo largo de una palabra con las jorobas de un camello. En el contexto de la convención CamelCase, se distinguen dos tipos principales:

- UpperCamelCase: En esta variante, la primera letra de cada palabra está en mayúscula. Ejemplo: NombreVariable, NombreEmpleado.
- lowerCamelCase: Similar a la anterior, pero con la primera letra en minúscula. Ejemplo: nombreVariable, nombreEmpleado.

Este estilo de escritura es ampliamente utilizado, siendo aplicable en elementos como los hashtags (#hashTags), nombres de compañías como LaTri, así como en variables en varios lenguajes de programación como PHP, Java y C#. En la presente guía se trabaja con la nomenclatura lowerCamelCase, tal como ya se evidencia en ejemplos anteriores tales como: conceptosBasicos, holaMundo, menuCiclos, mayorDosNumeros, entre otros.

Comentarios

Dentro de la programación, los comentarios representan una categoría especial de delimitadores que tienen la finalidad de explicar o aclarar ciertas instrucciones presentes en el código. Estos comentarios son proporcionados por el programador para contribuir a la comprensión y el mantenimiento del código, lo que resulta especialmente útil en su proceso de prueba y revisión. La meta es lograr que el código sea legible tanto para otros programadores como para el propio autor en momentos posteriores. Un aspecto importante es que los comentarios son obviados por el compilador, no afectan la ejecución del programa (Superior Abierta Distancia et al., n.d.).

En Java, los comentarios son de dos tipos:

Comentarios de línea: Estos comentarios se marcan utilizando el símbolo //. Son útiles para brindar aclaraciones sobre una sola línea de código.

Bloques de comentarios: Los bloques de comentarios se inician con /* y concluyen con */. Son útiles para proporcionar ex-

plicaciones más extensas o para bloquear secciones de código, lo cual puede ser útil al realizar pruebas o al realizar cambios en el código.

En ambos casos, los comentarios son herramientas esenciales para mejorar la comprensión y el mantenimiento del código fuente. A continuación, se muestra un ejemplo de los dos tipos de comentarios (Hernández Bejarano, 2021).

Figura 43. Ejemplo comentarios.

```
//comentario de linea
//La siguiente linea permite imprimir en consola
//Hola Mundo
System.out.println("Hola Mundo");
/* comentario
multi
linea*/
```

3.2.1 Tipos de variables

Tipos de datos primitivos y estructurados en Java

En el entorno de programación Java, existe un conjunto reducido de tipos de datos primitivos que desempeñan un papel fundamental. Estos tipos son esenciales para gestionar información básica, como números y valores booleanos (“verdadero” o “falso”). Además de estos tipos primitivos, Java también ofrece tipos estructurados u objetos que permiten manipular conjuntos más complejos de información. A continuación, se detallan los tipos de datos primitivos y estructurados en Java:(SENA, n.d.).

Tipos de datos primitivos:

Los tipos de datos primitivos se utilizan para manejar información básica y están divididos en dos categorías: numéricos y no numéricos.

Tipos numéricos enteros:

1. byte: Almacena valores enteros en el rango de -128 a 127.
2. short: Almacena valores enteros en el rango de -32,768 a 32,767.
3. int: Almacena valores enteros en el rango de -2,147,483,648 a 2,147,483,647.
4. long: Almacena valores enteros en el rango de -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807.

Tipos numéricos en punto flotante:

5. float: Almacena números reales de precisión simple en el rango de aproximadamente 1.4x10^-45 a 3.4028235x10^38.
6. double: Almacena números reales de precisión doble en el rango de aproximadamente 4.9x10^-324 a 1.7976931348623157x10^308.

Otros tipos primitivos:

7. boolean: Almacena valores booleanos (“verdadero” o “falso”).

8. char: Almacena caracteres individuales utilizando codificación UTF-16 de Unicode.

Tipos de datos estructurados

Aparte de los tipos primitivos, Java también ofrece tipos estructurados que pueden contener múltiples valores de tipos primitivos. Algunos ejemplos incluyen:

Cadenas de caracteres (String): Permite almacenar secuencias de caracteres y ofrece métodos para operar con ellas.

Vectores (arrays): Colecciones de elementos del mismo tipo, numerados a partir de 0. Pueden trabajar en una o más dimensiones.

Tipos definidos por el usuario: Además de los tipos proporcionados por Java, se pueden crear clases personalizadas para manejar operaciones y almacenar información específica.

Tipos envoltorio (wrapper): Son clases equivalentes a los tipos primitivos, que agregan métodos y propiedades útiles. Ejemplos de estos tipos son Byte, Short, Integer, Long, Float, Double, Boolean y Character (Hernández-Cruz et al., 2021).

3.2.2 Ejemplos de aplicación

```
Integer idCargoEmpleado;
String detalles;
```

3.3 Operadores

Operadores de Asignación y Aritméticos en Java

Operador de Asignación.

El operador de asignación en Java permite asignar un valor a una variable. Su estructura es simple, utilizando el símbolo igual (=):

variable = valor;

A través de este operador, podemos asignar valores a variables de diferentes tipos, como enteros, cadenas, números decimales y valores booleanos: (Borboa-Acosta et al., 2019)

int numeroUno = 3;

String NombreDeInstitucion = “Nelson Torres”;

Double altura = 1.70;

Boolean opcion = true;

Operadores Aritméticos.

Los operadores aritméticos en Java desempeñan un papel fundamental al permitir la ejecución de operaciones matemáticas esenciales, incluyendo sumas, restas, multiplicaciones, divisiones y la obtención del residuo en una división. Estos operadores se aplican entre dos literales o variables, y el resultado obtenido puede asignarse a una variable o integrarse en una expresión (Llerena et al., 2019).

Los operadores aritméticos en Java son los siguientes.

+ : Suma (Se usa además para concatenar de caracteres).

- : Resta.

* : Multiplicación.

/ : División.

% : Resto.

Ejemplos de uso de operadores aritméticos:

```
int suma = 10 + 5; // suma = 15

int resta = 15–5; // resta = 10

int multiplicacion = 5 * 2; // multiplicacion = 10

int division = 25 / 5; // division = 5

int resto = 8 % 3; // resto = 2
```

Los operadores aritméticos también pueden aplicarse en combinación con variables:

```
int variableUno = 5;

int sumaVariable = variableUno + 15; // sumaVariable = 20

int restaVariable = variableUno–2; // restaVariable = 3
```

Además, los operadores aritméticos pueden ser utilizados en expresiones condicionales:

```
if (edad > resultado +5) {
    // ...
}
```

Los operadores de asignación y aritméticos en Java son esenciales para realizar asignaciones de valores y llevar a cabo operaciones matemáticas en programas. Estos operadores forman parte de las herramientas fundamentales para la manipulación de datos y el control de flujo en la programación.

3.4 Sentencias de control de decisión if.

La sentencia if-else en Java representa una estructura de control que posibilita la evaluación de una expresión lógica, también conocida como condición. Dependiendo de si esta condición resulta en verdadero o falso, se ejecutará uno de los dos bloques de sentencias disponibles. En resumen, esta estructura de control habilita la toma de decisiones en el flujo de ejecución de un programa (Uruchurtu Moreno et al., 2020).

La estructura básica de una sentencia if-else es la siguiente:

```
if (condición) {
    // Código si la condición es VERDADERA
} else {
    // Código si la condición es FALSA
}
```

La condición se refiere a una expresión lógica que se analiza para determinar si es cierta (true) o falsa (false). Cuando la condición es cierta, se procede a ejecutar el conjunto de instrucciones contenido en las llaves {} que están asociadas al primer bloque. En caso de que la condición resulte ser falsa, se ejecutan

las instrucciones ubicadas dentro del segundo conjunto de llaves {} que siguen a la palabra clave "else", en caso de que exista. Esto permite controlar el flujo del programa y ejecutar distintas secciones de código dependiendo de si la condición es verificada como verdadera o falsa.

A continuación se muestra un ejemplo sencillo:

```
int numeroUno = 5;

if (numeroUno > 0) {

    System.out.println("NÚMERO ES POSITIVO: " + numeroUno);

} else {
   System.out.println("NÚMERO NO ES POSITIVO: " + numeroUno);
}
```

3.4.1 Ejemplos de aplicación

En este ejemplo, si el valor de número es mayor que 0, se imprimirá "El número es positivo."; de lo contrario, se imprimirá "El número no es positivo.".

También es posible encadenar múltiples sentencias if-else para evaluar varias condiciones en secuencia. Por ejemplo:

```
int edadEmpleado = 18;

if (edadEmpleado <= 18) {

    System.out.println("Empleado menor de edad.");

} else if (edad >= 18 && edad < 65) {
    System.out.println("Mayor de Edad.");
} else {
    System.out.println("Adulto Mayor.");
}
```

En este caso, se evalúa la edad y se imprimen mensajes diferentes según la condición que se cumpla primero. La estructura if-else if-else, en Java, posibilita la evaluación secuencial de múltiples condiciones. Cuando una de estas condiciones se evalúa como verdadera, se ejecuta el conjunto de instrucciones asociado y se omiten los bloques restantes. Esta estructura es útil cuando se necesita elegir entre varias opciones mutuamente excluyentes y ejecutar el bloque de código correspondiente a la primera condición que sea verdadera, descartando las demás.

3.5 Conclusión

En el Capítulo 3, hemos explorado conceptos fundamentales de la programación en Java. Comenzamos comprendiendo la importancia de los lenguajes de programación y cómo Java se destaca como una opción popular y poderosa para desarrolladores en todo el mundo.

Luego, profundizamos en la estructura básica de un programa Java, que involucra clases, métodos y variables. Aprendimos cómo declarar variables y asignarles valores, y cómo utilizar los operadores aritméticos para realizar cálculos matemáticos simples.

Exploramos también las sentencias condicionales, en particular la sentencia if-else, que nos permite tomar decisiones en nuestros programas según condiciones lógicas. Vimos cómo encadenar múltiples condiciones para manejar situaciones más complejas.

Además, abordamos conceptos relacionados con los tipos de datos en Java, incluyendo tipos primitivos, tipos envoltorios y tipos definidos por el usuario. Comprendimos la importancia de elegir el tipo de dato adecuado según nuestras necesidades y cómo declarar variables de estos tipos.

Finalmente, introdujimos los operadores de asignación y aritméticos, que son fundamentales para realizar cálculos y asignar valores en nuestros programas Java.

A medida que avanzamos en nuestro viaje de aprendizaje de Java, es esencial comprender estos conceptos fundamentales, ya que forman la base sobre la cual construimos aplicaciones más complejas y poderosas. En el próximo capítulo, exploraremos conceptos avanzados de programación orientada a objetos, que son esenciales para desarrollar aplicaciones Java robustas y eficientes. ¡Sigamos adelante en nuestro viaje de aprendizaje de Java.

Autoevaluación Capitulo 3

1. ¿Cuál es el operador utilizado para asignar un valor a una variable en Java?

a) ==

b) =

c) :=

d) =>

2. ¿Cuál es el operador utilizado para realizar una suma en Java?

a) +

b) *

c) /

d) -

2. **¿Cuál de los siguientes tipos de datos en Java permite almacenar valores enteros más largos?**

a) Byte

b) Short

c) Integer

d) Long

3. **¿Qué hace la sentencia if en Java?**

a) Realiza una multiplicación

b) Evalúa una expresión lógica y ejecuta un bloque de código si es verdadera

c) Divide dos números

d) Asigna un valor a una variable

4. **¿Cuál es la representación de caracteres individuales en Java?**

a) String

b) Char

c) Character

d) CharSeq

5. **¿Cuál es el tipo de dato que representa valores booleanos en Java?**
    a) Int
    b) Boolean
    c) Bool
    d) TrueFalse
6. **¿Cuál de las siguientes opciones es un tipo primitivo en Java?**
    a) Integer
    b) String
    c) Float
    d) Double

Respuesta: c) Float

7. **¿Cuál de los siguientes operadores aritméticos se utiliza para la división en Java?**
    a) +
    b) *
    c) /
    d) -
8. **¿Qué es el operador de asignación += en Java?**
    a) Un operador de suma
    b) Un operador de asignación de suma
    c) Un operador de comparación
    d) Un operador de asignación de igualdad
9. **¿Qué tipo de operadores permiten encadenar varias condiciones y ejecutar un bloque de código según la condición que se cumpla?**
    a) Operadores lógicos
    b) Operadores de comparación
    c) Operadores aritméticos
    d) Operadores de asignación

Capítulo 4
Sentencias de control case

La sentencia switch-case es utilizada para tomar decisiones basadas en el valor de una expresión. Proporciona una manera más eficiente y organizada de manejar múltiples opciones en lugar de usar una serie de sentencias if-else (Ochoa & Bedregal-Alpaca, 2022).

Aquí hay una descripción general de cómo funciona switch-case:

Expresión de control

Se evalúa una expresión, generalmente una variable o una expresión que produce un valor. Este valor se utiliza para determinar cuál de los casos se ejecutará.

Casos

Dentro de la estructura switch, tienes varios bloques case. Cada bloque case contiene un valor o una expresión constante que se compara con la expresión de control.

Ejecución del código

Cuando se detecta un bloque "case" cuyo valor coincide con la expresión de control, se ejecuta el conjunto de instrucciones que se encuentra dentro de ese bloque "case".

Break

Después de ejecutar el código en un bloque case, generalmente se usa la palabra clave break para salir del switch. Esto evita que se ejecuten los casos restantes.

Default (opcional).

Puedes incluir un bloque default al final del switch. Este bloque se ejecutará si ninguno de los casos coincide con la expresión de control. Es opcional, pero a menudo se usa para manejar cualquier otro caso que no se haya especificado.

Ejemplo de aplicación:

```
int diaDeLaSemana = 3;
msj = "El número "+ diaDeLaSemana + " corresponde a: ";
switch (diaDeLaSemana) {
case 1:
    msj = msj + " LUNES";
    break;
case 2:
    msj = msj + " MARTES";
    break;
case 3:
    msj = msj + " MIÉRCOLES";
    break;
case 4:
    msj = msj + " JUEVES";
    break;
case 5:
    msj = msj + " VIERNES";
    break;
default:
    msj = msj + " FIN DE SEMANA ";
}
    msj = msj + "";
    System.out.println(msj);
```

En este ejemplo, el valor de diaDeLaSemana es 3, por lo que se ejecuta el código dentro del bloque case 3. El resultado será "EL número 3 corresponde a: MIÉRCOLES". Si diaDeLaSemana fuera un valor que no coincide con ninguno de los casos, se ejecutaría el bloque default.

Recuerda que es importante usar break después de cada bloque case para evitar la ejecución accidental de otros casos. El bloque default es opcional y puede omitirse si no se necesita.

4.1.1 Ejemplos de aplicación.

Crear un método el cual le permita crear un menú que permita acceder a los métodos implementados dentro de la clase denominada ciclos:

Figura 44. Método menuCiclos.

```
public static void menuCiclos(){
    Scanner t = new Scanner(System.in);
    Integer op;
    System.out.println("Munu Ciclo");
    System.out.println("Elige una opcion");
    System.out.println("1. Numeros del 1 al 10");
    System.out.println("2. Palabra Invertida");
    System.out.println("3. Factorial");
    System.out.println("4. Factorial del 1 hasta N");
    System.out.println("5. Serie");
    System.out.println("6. Salir");
    op = t.nextInt();
    switch(op){
        case 1:
            numerosUnoDiez();
            break;
        case 2:
            ciclosWhile.palabraInversa();
            break;
        case 6:
            System.out.println("Salio del menú - ciclo");
        default:
            System.out.println("Opcion no válida.");

    }
}
```

El método menuCiclos() que proporcionaste es un método estático en Java que muestra un menú de opciones al usuario y realiza diferentes acciones según la opción seleccionada. Veamos una descripción técnica del método:

1. Se importa la clase Scanner para permitir la entrada de datos desde la consola.

2. Se declara una variable t de tipo Scanner para leer la entrada del usuario.

3. Se declara una variable op de tipo Integer que se utilizará para almacenar la opción seleccionada por el usuario en el menú.

4. Se muestra un encabezado que indica que es un menú con la línea System.out.println("Menú Ciclo");.

5. Se muestran las opciones disponibles para el usuario, enumeradas del 1 al 6, cada una acompañada de una descripción breve, utilizando varias llamadas a System.out.println().

6. Se utiliza t.nextInt() para leer la opción ingresada por el usuario desde la entrada estándar y se almacena en la variable op.

7. Se utiliza una estructura switch-case para tomar decisiones basadas en el valor de op. Dependiendo de la opción seleccionada por el usuario, se ejecutará un bloque de código específico.

- Si op es igual a 1, se llama a un método llamado numerosUnoDiez().
- Si op es igual a 2, se llama a un método palabraInversa() de la clase ciclosWhile.
- Si op es igual a 6, se muestra un mensaje indicando que el usuario salió del menú-ciclo.

En el caso predeterminado (ninguna de las opciones anteriores coincide), se muestra un mensaje que indica que la opción no es válida.

Este método crea un menú interactivo que muestra las opciones al usuario, lee la opción seleccionada y ejecuta una acción específica según la opción elegida. Es importante destacar que este método asume que existen métodos llamados numerosUnoDiez() y palabraInversa() que realizarán las acciones correspondientes a las opciones 1 y 2 respectivamente. Además, el bucle do-while que podría haber rodeado a este método no está presente aquí, por lo que el menú se ejecutará una sola vez y luego finalizará.

4.2 Sentencia Bucle for().

La sentencia de control "for" en Java se emplea para establecer bucles o ciclos que ejecutan un bloque de código un número determinado de veces. Esta es una de las estructuras de control de repetición más comunes en Java y otros lenguajes de programación. La sintaxis general de un bucle "for" en Java es la siguiente:

```
for (inicio; condición; incremento) {
    // Código que ejecuta la iteración
}
```

inicio

Esta parte se ejecuta una vez al principio del bucle y generalmente se utiliza para inicializar una variable de control que se usará en el bucle.

Condición

Es una expresión booleana que se evalúa antes de cada iteración. Si la condición es verdadera (true), el bucle continúa ejecutándose; si es falsa (false), el bucle se detiene y se sale.

Incremento

Esta parte se ejecuta al final de cada iteración y generalmente se utiliza para modificar la variable de control de modo que se acerque al valor que hará que la condición sea false. Esto se conoce como "actualizar" la variable de control.

El flujo de un bucle for típico es el siguiente:

1. Se ejecuta la parte de inicialización una sola vez al comienzo del bucle.
2. Se verifica la condición. Si es true, el bloque de código dentro del bucle se ejecuta. Si es false, el bucle se detiene y se sale.

3. Después de que el bloque de código dentro del bucle se ejecuta, se ejecuta la parte de actualización.
4. Se repite el paso 2. Si la condición sigue siendo true, el bloque de código se ejecuta nuevamente, y así sucesivamente.

El bucle for es especialmente útil cuando se sabe de antemano cuántas veces se debe ejecutar un bloque de código. También se utiliza para recorrer colecciones de datos, como matrices o listas, donde la longitud de la colección se utiliza como condición de parada.

Ejemplo de un bucle for que imprime números del 1 al 15:

```
for ( int i = 1 ; i <= 5; i++) {
    System.out.println( i );
}
```

Este bucle se ejecutará cinco veces, imprimiendo los números del 1 al 5 en la consola.

4.2.1 Ejemplos de aplicación.

El método que proporcionaste se llama `numerosUnoDiez()` y tiene el propósito de imprimir los números del 1 al 20 que son pares. Aquí está un análisis técnico del método:

```
public static void numerosUnoDiez() {
    for (int i = 1; i <= 20; i++) {
            if (i % 2 == 1) {
                    continue;
```

```
            }
            System.out.println("Número: " + i);
        }
    }
```

Analicemos este método paso a paso:

1. Se declara un método llamado `numerosUnoDiez()` con un modificador `public` y `static`. Esto significa que el método es accesible desde cualquier parte del programa sin necesidad de crear una instancia de la clase que lo contiene.

2. Dentro del método, se inicializa un bucle `for` con la variable de control `i` inicializada en 1. El bucle se ejecutará mientras `i` sea menor o igual a 20.

3. Dentro del bucle `for`, hay una estructura condicional `if` que verifica si `i` es impar. Esto se hace comprobando si el residuo de la división de `i` entre 2 (`i % 2`) es igual a 1 (es decir, si `i` es impar).

4. Si `i` es impar, se encuentra la declaración `continue`, lo que significa que se salta el resto de las instrucciones en la iteración actual y se pasa a la siguiente iteración del bucle `for`.

5. Si `i` no es impar (es decir, es par), se ejecuta la instrucción `System.out.println("Número: " + i);`. Esto imprime en la consola el número par actual.

6. El bucle `for` se repite para cada valor de `i` del 1 al 20. Sin embargo, solo se imprimirán los números pares debido a la instrucción `continue` que salta las iteraciones para números impares.

Este método utiliza un bucle `for` para iterar a través de los números del 1 al 20 y verifica si cada número es par o impar. Solo se imprimen en la consola los números pares encontrados, mientras que los impares se omiten utilizando la declaración `continue`.

4.3 Sentencia Bucle while()

La sentencia while en Java se utiliza para crear un bucle que ejecuta un conjunto de instrucciones mientras una condición especificada sea verdadera. En otras palabras, el código dentro del bloque de while se repetirá continuamente hasta que la condición especificada sea falsa. (Fernández, 2014)

Aquí está la sintaxis básica de la sentencia while en Java:

```
while (condición) {
    // Bloque de Código que se ejecuta mientras la condición
    sea verdadera
}
```

Condición.- Es una expresión booleana que se evalúa antes de cada iteración del bucle. Si la condición es verdadera, el código dentro del bucle se ejecuta. Si la condición es falsa desde el principio, el código dentro del bucle nunca se ejecutará.

Ejemplo de uso de while:

```
int cont = 1;
while (cont <= 5) {
    System.out.println("Iteración " + cont);
    cont++;
}
```

En este ejemplo, el bucle while imprimirá "Iteración 1" a "Iteración 5" en la consola porque la condición contador <= 5 es verdadera durante esas iteraciones. Una vez que contador alcanza el valor 6, la condición se vuelve falsa y el bucle se detiene.

Es importante tener en cuenta que si la condición en un bucle while nunca se vuelve falsa, el bucle se ejecutará indefinidamente, lo que podría causar un "bucle infinito". Para evitar esto, es importante asegurarse de que la condición cambie de alguna manera en cada iteración, de lo contrario, el bucle no terminará.

También puedes usar las sentencias break y continue dentro de un bucle while para controlar el flujo del programa. break se usa para salir inmediatamente del bucle, y continue se utiliza para saltar la iteración actual y continuar con la siguiente.

Do while()

La sentencia "do-while" guarda similitudes con "while", aunque presenta una distinción fundamental: en un bucle "do-while", la condición se evalúa después de la ejecución del bloque de código. En consecuencia, el bloque de código se ejecuta al menos

una vez, incluso en el caso de que la condición sea inicialmente falsa.(de programación & Miguel Toro Bonilla, 2022)

Aquí está la sintaxis básica de la sentencia `do-while` en Java:

```
do {
    // Bloque de Código a ejecutar
} while (condición);
```

El bloque de código se ejecutará primero sin importar el valor de la condición.

Después de eso, se evalúa la condición. Si resulta ser verdadera, el bucle se repetirá, y el bloque de código se ejecutará de nuevo. En caso de que sea falsa, el bucle se detendrá y el programa proseguirá después de la sentencia "do-while".

Vamos a relacionar esto con un ejemplo. Tomemos el ejemplo anterior del bucle "while":

```
int cont = 1;
while (cont <= 5) {
    System.out.println("Iteración " + cont);
    cont ++;
}
```

Si quisiéramos lograr el mismo resultado utilizando un bucle `do-while`, se vería así:

```
int cont = 1;
do {
    System.out.println("Iteración " + cont);
```

```
        contador++;
    } while (cont <= 5);
```

En este caso, el resultado sigue siendo el mismo. El bloque de código se ejecuta al menos una vez, y luego la condición "contador <= 5" se verifica. Si la condición es verdadera, el bucle se repite; de lo contrario, se detiene.

La elección entre "while" y "do-while" depende de tus necesidades. Si necesitas que el bloque de código se ejecute al menos una vez sin importar la condición inicial, "do-while" es útil. Si deseas que la condición se verifique antes de ejecutar el bloque de código, entonces "while" es más apropiado.

Espero que esta explicación aclare la relación entre "while" y "do-while". Si tienes más preguntas o necesitas más ejemplos, no dudes en preguntar.

4.3.1 Ejemplos de aplicación.

Figura 45. Método EjemploWhile.

```
public static void ejemplosWhile(){

    System.out.println("Mostrar los valores pares del 1 al 20");
    Integer i =1;
    while(i<=20){
        if(i%2==0){
            System.out.println(""+i);
        }
        i++;
    }
}
```

Nombre del Método: ejemplosWhile

Visibilidad y Modificadores: El método es público ("public") y estático ("static"), lo que significa que puede ser accedido desde cualquier parte del código sin necesidad de crear una instancia de la clase en la que se encuentra.

Retorno: Al ser "void" el método no devuelve ningún valor, ya que su único propósito es imprimir información en la consola.

Parámetros: El método no toma ningún parámetro.

Descripción: Este método tiene la finalidad de mostrar los valores pares en el rango del 1 al 20 utilizando un bucle "while". Veamos cómo funciona:

1. Se imprime en la consola el mensaje "Mostrar los valores pares del 1 al 20" para proporcionar información sobre lo que hará el método.

2. Se declara una variable "i" e inicializa en 1. Esta variable se utilizará como contador para recorrer los números del 1 al 20.

3. Se inicia un bucle "while" que continuará mientras "i" sea menor o igual a 20.

4. Dentro del bucle, se verifica si "i" es par utilizando la expresión "if (i % 2 == 0)". Esto se hace calculando el residuo de la división de "i" por 2. Si el residuo es 0, significa que "i" es par, y se ejecuta el código dentro del bloque "if".

5. Si “i” es par, se imprime en la consola el valor de “i” utilizando “System.out.println(“” + i);”.

6. Luego, se incrementa el valor de “i” en 1 utilizando “i++”.

7. El bucle “while” continúa con la siguiente iteración hasta que “i” sea mayor que 20.

Resultado Esperado: Como resultado, este método imprimirá en la consola los números pares del 1 al 20, uno por uno.

Uso Potencial: Este método podría ser utilizado en un programa más grande cuando necesitas mostrar o realizar alguna operación en una secuencia de números pares dentro de un rango específico utilizando un bucle “while”.

El método “ejemplosWhile” es un ejemplo de cómo utilizar un bucle “while” para mostrar números pares en un rango específico y proporciona información adicional mediante mensajes en la consola.

4.4 Estructura de Datos Arreglos

4.4.1 Definición de Vectores

Un vector o array en Java es una colección ordenada de elementos del mismo tipo de dato. Puedes pensar en él como una lista de valores que están organizados en posiciones numeradas, comenzando desde 0.

Por ejemplo, puedes crear un array de enteros para almacenar números enteros o un array de cadenas para almacenar texto. Los arrays tienen una longitud fija cuando se crean, lo que significa que no pueden cambiar de tamaño después de la creación. (Hernández Bejarano, 2021)

Declaración de un Array en Java:

tipo[] nombreDelArray;

Aquí tienes ejemplos de declaraciones de arrays:

```
int[] numeros;        // Declaración de un array de enteros
String[] nombres;  // Declaración de un array de cadenas
double[] temperaturas;      // Declaración de un array de números de punto flotante
```

Creación de un Array en Java:

Una vez declarado un array, debes crearlo y especificar su longitud (el número de elementos que contendrá). Aquí tienes ejemplos de cómo crear arrays:

```
numeros = new int[5];        // Creación de un array de enteros con longitud 5
nombres = new String[3];   // Creación de un array de cadenas con longitud 3
temperaturas = new double[10];     // Creación de un array de números de punto flotante con longitud 10
```

También puedes declarar y crear un array en una sola línea:

```
int[] numerosCedula = new int[5];
String[] nombresClientes = new String[3];
double[] estaturaEstudiantes = new double[10];
```

Inicialización de un Array en Java:

Se puede usar índices para asignar valores que formaran parte de un array. Los índices comienzan desde 0. Aquí tienes ejemplos de cómo inicializar arrays:

```
numeros[0] = 1; // Asignar el valor 1 al primer elemento del array
numeros[1] = 2; // Asignar el valor 2 al segundo elemento del array
nombres[0] = "Alice"; // Asignar el texto "Alice" al primer elemento del array
nombres[1] = "Bob"; // Asignar el texto "Bob" al segundo elemento del array
```

También es posible inicializar un array cuando lo creas:

```
int[] numeros = {1, 2, 3, 4, 5}; // Inicialización de un array de enteros
String[] colores = {"Rojo", "Verde", "Azul"}; // Inicialización de un array de cadenas
```

Acceso a Elementos de un Array:

Es posible acceder a los elementos de un array utilizando sus índices. Por ejemplo:

```
int primerNumero = numeros[0]; // Acceder al primer elemento del array de números

String segundoColor = colores[1]; // Acceder al segundo elemento del array de colores
```

Iteración a través de un Array:

Para acceder y mostrar en consola todos los elementos de un array, se puede utilizar bucles como "for". A continuación se muestra un ejemplo:

```
for (int i = 0; i < numeros.length; i++) {
    System.out.println(edadesEmpleados[i]);
}
```

Longitud de un Array

Se puede obtener la longitud de un array haciendo uso de la propiedad "length". Por ejemplo:

```
int longitud = numeros.length; // Obtener la longitud del array de números

int longitudColores = colores.length; // Obtener la longitud del array de colores
```

Arrays Multidimensionales:

Además de los arrays unidimensionales, Java también admite arrays multidimensionales. Por ejemplo, puedes tener un array bidimensional (una tabla) o un array tridimensional (una matriz de cubos). La declaración y creación de estos tipos de arrays es un poco más compleja, pero siguen la misma idea básica de almacenar elementos en una estructura organizada. (Caneo Salinas, 2018)

Matrices

Las matrices en programación son estructuras de datos que permiten almacenar valores en una cuadrícula bidimensional, similar a una tabla o una hoja de cálculo. En Java, puedes crear matrices multidimensionales para trabajar con datos organizados en filas y columnas. Aquí tienes información sobre matrices en Java:

Declaración de una Matriz en Java

Una matriz en Java se declara especificando el tipo de datos de sus elementos seguido por corchetes dobles `[][]` para indicar que es bidimensional. Aquí tienes ejemplos de declaraciones de matrices:

```
tipoDeDato[][] nombreDeLaMatriz;
int[][] matrizDeEnteros;
double[][] matrizDeDobles;
String[][] matrizDeCadenas;
```

Creación de una Matriz en Java:

Una vez declarada una matriz, debes crearla y especificar su tamaño, que corresponde a, el número de filas y columnas. A continuación ejemplos de cómo crear matrices:

```
matrizDeEnteros = new int[3][4];          // Matriz de enteros de 3 filas y 4 columnas
matrizDeDobles = new double[2][2];        // Matriz de números de punto flotante de 2x2
matrizDeCadenas = new String[4][3];       // Matriz de cadenas de 4 filas y 3 columnas
```

Inicialización de una Matriz en Java

Para asignar valores a los elementos de una matriz utilizando índices para indicar la fila y la columna. Los índices comienzan desde 0. Aquí se muestra algunos ejemplos de cómo inicializar matrices:

```
matrizDeEnteros[0][0] = 1;          // Asignar el valor 1 a la primera fila y primera columna

matrizDeEnteros[1][2] = 5;          // Asignar el valor 5 a la segunda fila y tercera columna

matrizDeCadenas[2][1] = "Hola";     // Asignar el texto "Hola" a la tercera fila y segunda columna
```

También puedes inicializar una matriz cuando la creas:

int[][] matrizDeEnteros = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

String[][] matrizDeNombres = {{“Alice”, “Bob”}, {“Carol”, “Dave”}};

Acceso a Elementos de una Matriz:

Para acceder a los elementos de una matriz utilizando sus índices de fila y columna. Por ejemplo:

int elemento = matrizDeEnteros[1][2]; // Acceder al elemento en la segunda fila y tercera columna

String nombre = matrizDeNombres[0][1]; // Acceder al elemento en la primera fila y segunda columna

Iteración a través de una Matriz:

Para acceder todos los elementos de una matriz, puedes utilizar bucles anidados `for` para iterar a través de las filas y columnas. Aquí tienes un ejemplo:

```
for (int fila = 0; fila < matrizDeEnteros.length; fila++) {
   for (int columna = 0; columna < matrizDeEnteros[0].length;
   columna++) {
      int elemento = matrizDeEnteros[fila][columna];
      System.out.println(“Fila “ + fila + “, Columna “ + columna
      + “: “ + elemento);
   }
}
```

Longitud de una Matriz:

Para obtener la longitud de una matriz utilizando las propiedades `length` para las filas y las columnas. Por ejemplo:

```
int filas = matrizDeEnteros.length;          // Obtener el número de filas
int columnas = matrizDeEnteros[0].length;    // Obtener el número de columnas
```

Matrices Irregulares:

En Java, las matrices multidimensionales deben tener todas sus filas con la misma cantidad de columnas. Si deseas trabajar con matrices de diferentes longitudes, puedes usar matrices de arrays o ArrayLists.

Matrices en Java:

Ejemplo Completo:

```
public class EjemploMatrices {
    public static void main(String[] args) {
        int[][] matrizEnteros = {{15, 29, 84}, {44, 435, 65}, {277, 28, 19}};
        for (int fila = 0; fila < matriz.length; fila++) {
          for (int columna = 0; columna < matriz[0].length; columna++) {
              int elemento = matriz[fila][columna];
              System.out.print(elemento + " ");
          }
```

```
            System.out.println();
        }
    }
}
```

Este ejemplo crea una matriz de enteros, la inicializa y la recorre utilizando bucles `for`.

4.5 Conclusión

En el Capítulo IV, hemos explorado una serie de conceptos fundamentales en la programación en Java relacionados con la gestión de datos estructurados y el control de flujo. Aprendimos a utilizar switch case para tomar decisiones basadas en un valor específico. Esta estructura es útil cuando se necesita comparar un valor con múltiples casos y ejecutar bloques de código según el caso que coincida.

Luego, nos sumergimos en las estructuras de bucle, comenzando con la declaración "for", que nos proporciona un control preciso sobre la repetición de bloques de código. Exploramos cómo utilizar bucles "for" para realizar tareas repetitivas y cómo gestionar la variable de control.

Continuamos nuestro viaje con los bucles "while" y "do while". Estos bucles nos permiten ejecutar bloques de código repetidamente mientras se cumplan ciertas condiciones. Aprendimos a utilizar estas estructuras de control para crear ciclos personalizables y versátiles.

Además de las estructuras de control, también estudiamos los vectores y las matrices en Java. Los vectores nos permiten almacenar colecciones de datos del mismo tipo, mientras que las matrices nos brindan una forma de organizar datos en una estructura bidimensional. Exploramos cómo declarar, inicializar y acceder a elementos en vectores y matrices.

En este capítulo, hemos adquirido un conocimiento sólido sobre cómo tomar decisiones condicionales, cómo controlar la repetición de código y cómo trabajar con estructuras de datos como vectores y matrices en Java. Estos conceptos son esenciales para la construcción de programas más complejos y funcionales. A medida que avanzamos en nuestro viaje de aprendizaje de Java, podremos aplicar estas herramientas para resolver una variedad de problemas y crear aplicaciones más robustas y eficientes.

Autoevaluación Capitulo 4.

1. ¿Cuál es el propósito principal del comando "switch case" en Java?

- a) Realizar operaciones matemáticas.
- b) Ejecutar código repetitivo.
- c) Tomar decisiones basadas en el valor de una expresión.

2. ¿Qué afirmación sobre el operador "%" en Java es correcta?

- a) Realiza una multiplicación.
- b) Calcula el residuo de la división.
- c) Realiza una comparación de igualdad.

3. ¿En Java, cuál es el propósito de la estructura de control "for"?

- a) Declarar variables.
- b) Ejecutar código una vez.
- c) Crear bucles controlados por una variable de control.

4. ¿Cuál es la principal diferencia entre la estructura "do while" y la estructura "while" en Java?

- a) "do while" no permite bucles.
- b) "do while" verifica la condición antes de ejecutar el código.
- c) "do while" garantiza que el código se ejecute al menos una vez.

5. ¿Qué es un vector en Java?

- a) Una estructura de control.
- b) Un bucle "for".
- c) Estructura de datos que almacena elementos del mismo tipo de dato en una variable.

6. ¿Cómo se declara un vector de enteros en Java?

- a) “int numeros();”

- b) “int[] numeros;”

- c) “enteros numeros = new int[];”

7. ¿Qué es una matriz en Java?

- a) Un operador aritmético.

- b) Un arreglo unidimensional.

- c) Un arreglo bidimensional que almacena datos en filas y columnas.

8. ¿Cuál es la diferencia clave entre un vector y una matriz en Java?

- a) Un vector tiene filas y columnas.

- b) Una matriz es un arreglo unidimensional.

- c) Un vector es un arreglo unidimensional, mientras que una matriz es un arreglo bidimensional.

9. ¿Cuál es la estructura básica de una sentencia “switch case” en Java?

- a) Una expresión y un único caso.

- b) Una expresión y múltiples casos, pero sin bloques de código.

- c) Una expresión, múltiples casos y bloques de código para cada caso.

10. ¿Cuál es el resultado de la siguiente operación en Java?

```
int[] numeros = {2, 4, 6, 8, 10};
System.out.println(numeros[2]);
```

- a) 4

- b) 6

- c) 8

Referencias

Beúnes Cañete, J.E., Vargas Ricardo, A., Beúnes Cañete, J.E., & Vargas Ricardo, A. (2019). La introducción de la herramienta didáctica PSeInt en el proceso de enseñanza aprendizaje: una propuesta para Álgebra Lineal. *Transformación*, *15*(1).

Bohada Jaime, J.A., Delgado González, I.A., & Rodríguez Barrera, C. (2019). *Fundamentos de programación en lenguaje JAVA*. Editorial Fundación Universitaria Juan de Castellanos.

Borboa-Acosta, E., Gutiérrez-Cota, J.D., Montijo-Valenzuela, E.E., Soto-Patrón, D., Sámano-Hermosillo, E., & Torres-Amavizca, E.F. (2019). Evaluación de competencias en la finalización del curso: introducción a la programación básica multidisciplinaria con enfoque a solución de problemas, para alumnos de nuevo ingreso al nivel superior. *Revista Tecnológica-Educativa Docentes 2.0*, *7*(2). https://doi.org/10.37843/rted.v7i2.7

Cairo, O. (2006). *Fundamentos de programación. Piensa en C*. Pearson.

Caneo Salinas, O. (2018). *Cuaderno docente. Manual para la asignatura Computación I, Fundamentos*. https://www.upla.cl/bibliotecas/wp-content/uploads/Cuaderno-Docente_FP.pdf

Celi, P. (2023). *Fundamentos de programación basados en PSeInt–ITQ*. Doxa Ediciones.

Cruz-Barragán, A., Soberanes-Martín, A., & Lule-Peralta, A. (2019). PSeInt Technological tool to develop logical-mathematical intelligence in structured computer programming. *Journal of Technology and Innovation, 6*(19), 22-30. https://doi.org/10.35429/jti.2019.19.6.22.30

Del Prado, A., & Lamas, N. (2014). Alternativas para la enseñanza de pseudocódigo y diagrama de flujo. *Exactas.Unca.Edu.Ar*.

Duque, P.V. (2021). *Diseño estructurado de algoritmos aplicados en PSEINT*. Editorial Grupo Compás.

Educación Superior Abierta y a Distancia. (2010). *Fundamentos de programación Programa Desarrollado. https://www.academia.edu/7380397/Fundamentos_de_programación_Fundamentos_de_programación*

Fernández, L. (2014). Programación Orientada a Objetos en Java. *Universidad Politécnica De Madrid Escuela Universitaria De Informática Lenguajes, Proyectos Y Sistemas Informáticos*, *91*.

García Perdomo, E., Rodríguez Herrera, T. O., Borrero Suárez, S., Conde García, M. H., & Ibarra Gutiérrez, N. A. (2020). Articular la didáctica de la enseñanza de los fundamentos de programación para estudiantes de educación media de las instituciones de la ciudad de Neiva. *Encuentro Internacional De Educación En Ingeniería*. https://doi.org/10.26507/ponencia.855

García-Rodríguez, C.G., & Dugarte-Peña, G.L. (2022). Measuring the effect of using pseint as a tool to improve the introduction of students to programming: two comparative scenarios. *EDULEARN22 Proceedings*, *1*. https://doi.org/10.21125/edulearn.2022.1549

García Santillán, I. (2014). *Fundamentos de Programación Usando PSeInt*. Comisión de Publicación UPEC.

Hernández Bejarano, M., y Baquero Rey, L.E. (2020). *Fundamentos de programación web*. Editorial Universidad ECCI.

Hernández-Cruz, L.M., Mex-Álvarez, D.C., Flores-Guerrero, M.D., & Martínez-Gómez, S. D. J. (2021). Problem Based Learning as a strategy for teaching algorithms. *ECORFAN Journal-Democratic Republic of Congo, 7*(13), 1-9. https://doi.org/10.35429/ejdrc.2021.13.7.1.9

Huari Evangelista, F., & José Novara, P. (2014). Intérprete para probar un programa escrito en pseudocódigo. *Industrial Data*, *17*(1). https://doi.org/10.15381/idata.v17i1.12039

Instituto Tecnológico de Sonora. (2016). *Pseudocódigo y PSEINT*. https://www.itson.mx/oferta/isw/Documents/guia_pseint_2016.pdf

Laura-Ochoa, L., & Bedregal-Alpaca, N. (2022). Incorporation of Computational Thinking Practices to Enhance Learning in a Programming Course. *International Journal of Advanced Computer Science and Applications*, *13*(2). https://doi.org/10.14569/IJACSA.2022.0130224

Llerena, L., Rodriguez, N., Castro, J.W., & Acuña, S.T. (2019). Adapting usability techniques for application in open source Software: A multiple case study. *Information and Software Technology*, *107*, 48-64. https://doi.org/10.1016/j.infsof.2018.10.011

Martínez Ladrón de Guevara, J. (2020). *Fundamentos de programación en Java*. Editorial EME.

Peñarrieta, R. (2011). *Java y NetBeans 0*. https://www.academia.edu/9842118/Java_y_NetBeans_0

Perez, A. (n.d.). *Fundamentos de programación, Ejercicios prácticos*. https://www.academia.edu/32591810/Fundamentos_de_Programación

Pérez Narváez, H.O. Cazarez, J.L. (2017). *Introducción a la Programación*. PumaEditores.

Ronald, I., & Ayquipa, R. (n.d.). *TUTORIAL PSEINT Fundamentos de Programación*. http://pseint.sourceforge.net/

Sánchez, J. (2007). *Apuntes de Fundamentos de programación Java*. https://www.academia.edu/34340869/Fundamentos_de_programacion_C_Java

Sánchez, M., Bahamondez, E.V., & De Clunie, G.T. (2020). Use of PSeInt in teaching programming: A case study. *ACM International Conference Proceeding Series*, *Part F166737*. https://doi.org/10.1145/3401895.3402083

SENA. FAVA. (n.d.). *Fundamentos del lenguaje de programación JAVA*. https://www.academia.edu/90329823/Fundamentos_del_lenguaje_de_programación_JAVA

Toro Bonilla, J. (2022). Fundamentos de programación: JAVA. Editorial Universidad de Sevilla.

Uruchurtu Moreno, D.A., Solis Chavez, A., & Mendez Gurrola, I.I. (2020). Application and results of a practical-theoretical teaching process in the subject of programming fundamentals. *ICERI2020 Proceedings*, 7320-7327. https://doi.org/10.21125/iceri.2020.1572

Villalobos, J., & Casallas, R. (n.d.). *Fundamentos de programacion. Aprendizaje activo basado en casos*. Universidad de los Andes. https://www.academia.edu/33407771/Fundamentos_de_programacion

Anexos

Anexo 1. Solución Cuestionario Capítulo I

1. ¿Qué importancia tiene la programación en la resolución de problemas y tareas eficientes?

b) Importante

2. ¿Cómo se definen los algoritmos y qué característica fundamental poseen?

b) Son secuencias lógicas de pasos

3. ¿Qué es el pseudocódigo y cómo puede facilitar la comprensión y creación de algoritmos?

c) Una representación intermedia entre el lenguaje humano y de programación

4. ¿Qué es PSeInt y cómo se utiliza para apoyar a los principiantes en programación?

c) Un software que ayuda a desarrollar algoritmos en pseudocódigo

5. Describe la estructura general de un algoritmo en pseudocódigo.

b) Comienza con "Proceso" seguido del nombre del programa, luego sigue una secuencia de instrucciones y finaliza con "FinProceso"

6. ¿Por qué es esencial comprender la metodología de programación antes de aprender un lenguaje de programación?

c) Establece una estrategia para resolver problemas antes de abordar la sintaxis de un lenguaje

7. ¿Qué son los arreglos y cómo se dimensionan en PSeInt?

c) Son estructuras de datos homogéneas y se dimensionan con la instrucción "Dimensión"

8. Enumera y describe los tres tipos de datos básicos disponibles en PSeInt.

c) Numérico, Lógico y Carácter

9. ¿Cómo se establece la jerarquía de operadores matemáticos en PSeInt? ¿Qué es el "cortocircuito" en relación a los operadores & y |?

c) Sigue la jerarquía del álgebra y se puede alterar con paréntesis; el "cortocircuito" implica que en expresiones con & y |, si la primera parte determina el resultado, la segunda no se evalúa

10. ¿Qué son las funciones en PSeInt y cómo se utilizan en las expresiones?

b) Son operaciones predefinidas que procesan valores y devuelven resultados

11.¿Qué aspectos fundamentales se deben comprender para desarrollar programas y algoritmos eficaces?

c) La estructura de algoritmos, la metodología de programación, los tipos de datos, operadores y funciones.

Anexo 2. Solución Cuestionario Capítulo 2

1. ¿Qué determina la secuencia de instrucciones ejecutadas en la instrucción "Si-Entonces-SiNo"?

b) Una variable lógica

10. **¿Qué se evalúa en la instrucción «Mientras» antes de ejecutar el cuerpo del ciclo?**

b) Una condición lógica

11. **La instrucción "Segun" se utiliza para realizar:**

b) Toma de decisiones

12. **¿Qué ocurre en la instrucción «Para» después de ejecutar el cuerpo del ciclo?**

a) Se incrementa la variable y se verifica la condición

13. **En la instrucción "Si-Entonces-SiNo", ¿qué sucede si la condición es falsa y no se incluye la cláusula "SiNo"?**

a) Se ejecuta la secuencia de instrucciones siguiente

14. **¿Qué papel juega la cláusula «Con Paso <paso>» en la instrucción «Para»?**

b) Indica cuánto se incrementa la variable en cada repetición

15. **¿Cuál es el propósito de la instrucción «Segun» en términos de control de flujo?**

c) Realizar diferentes acciones basadas en el valor de una variable

16. **En la instrucción "Mientras", ¿qué sucede si la condición es falsa desde el principio?**

b) No se ejecuta ninguna instrucción

17. **¿Qué se debe hacer para evitar un ciclo infinito en la instrucción «Para»?**

    c) Asegurarse de que las instrucciones dentro del ciclo modifiquen la variable para cambiar la condición

18. **¿Cuál es el propósito principal de las estructuras de control en programación?**

    c) Controlar el flujo de ejecución y tomar decisiones en función de condiciones

Anexo 3 Autoevaluación Capítulo 3

1. ¿Cuál es el operador utilizado para asignar un valor a una variable en Java?

b) =

2. ¿Cuál es el operador utilizado para realizar una suma en Java?

a) +

3. **¿Cuál de los siguientes tipos de datos en Java permite almacenar valores enteros más largos?**

d) Long

4. **¿Qué hace la sentencia if en Java?**

b) Evalúa una expresión lógica y ejecuta un bloque de código si es verdadera

5. **¿Cuál es la representación de caracteres individuales en Java?**

c) Character

6. **¿Cuál es el tipo de dato que representa valores booleanos en Java?**

b) Boolean

7. **¿Cuál de las siguientes opciones es un tipo primitivo en Java?**

c) Float

8. **¿Cuál de los siguientes operadores aritméticos se utiliza para la división en Java?**

c) /

9. **¿Qué es el operador de asignación += en Java?**

b) Un operador de asignación de suma

10. **¿Qué tipo de operadores permiten encadenar varias condiciones y ejecutar un bloque de código según la condición que se cumpla?**

a) Operadores lógicos

Anexo 4. Solución Cuestionario Capítulo 4

1. ¿Cuál es el propósito principal del comando "switch case" en Java?

Respuesta: c) Tomar decisiones basadas en el valor de una expresión.

2. ¿Qué afirmación sobre el operador "%" en Java es correcta?

Respuesta: b) Calcula el residuo de la división.

3. ¿En Java, cuál es el propósito de la estructura de control "for"?

Respuesta: c) Crear bucles controlados por una variable de control.

4. ¿Cuál es la principal diferencia entre la estructura "do while" y la estructura "while" en Java?

Respuesta: c) `do while` garantiza que el código se ejecute al menos una vez.

5. ¿Qué es un vector en Java?

Respuesta: c) Una estructura de datos que almacena elementos del mismo tipo en una variable.

6. ¿Cómo se declara un vector de enteros en Java?

Respuesta: b) `int[] numeros;`

7. ¿Qué es una matriz en Java?

Respuesta: c) Un arreglo bidimensional que almacena datos en filas y columnas.

8. ¿Cuál es la diferencia clave entre un vector y una matriz en Java?

Respuesta: c) Un vector es un arreglo unidimensional, mientras que una matriz es un arreglo bidimensional.

9. ¿Cuál es la estructura básica de una sentencia "switch case" en Java?

Respuesta: c) Una expresión, múltiples casos y bloques de código para cada caso.

10. ¿Cuál es el resultado de la siguiente operación en Java?

Respuesta: b) 6

Religación
Press
Ideas desde el Sur Global

Religación Press
Ideas desde el Sur Global

Instituto Nelson Torres

ISBN: 978-9942-642-36-3