CIBERNÉTICA Y COMPUTACIÓN II

GUÍA PARA PREPARAR EXAMEN EXTRAORDINARIO

AUTORES

ANGÉLICA VIANEY ZAVALETA HERNÁNDEZ

ANTONIO GRANILLO MARTÍNEZ

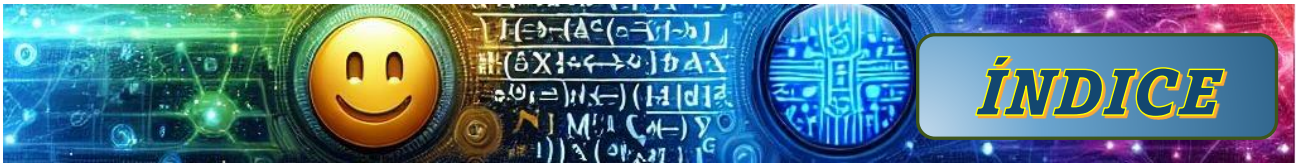
CÉSAR RAMÍREZ ORTEGA

MIGUEL ÁNGEL RODRÍGUEZ PADILLA

PEDRO ERNESTO NEQUIZ MEDINA

2025

COORDINACIÓN:
ANGÉLICA VIANEY ZAVALETA HERNÁNDEZ



La presente Guía ha sido diseñada específicamente para estudiantes de la asignatura Cibernética y Computación II, del 6° semestre de la E. N. Colegio de Ciencias y Humanidades, de acuerdo con los *Programas de Estudio – Área de Matemáticas – Cibernética y Computación II*, Ed. 2016 (UNAM-CCH. 2016); y con el propósito de facilitar la adquisición de los aprendizajes, habilidades y conocimientos respectivos; y así, quien desee ser sustentante de dicho examen extraordinario, pueda afrontar exitosamente la preparación correspondiente. Este documento, de acuerdo con los Programas de Estudio respectivos, abarca el total de las unidades: 1. Lenguaje de programación orientada a objetos con Java; 2. Estructuras de control de secuencia en Java; 3. Polimorfismo, constructores, colaboración y herencia de clases; y, 4. Interfaz gráfica de usuario. Todo lo anterior, con la finalidad de consolidar y fortalecer la adquisición de los aprendizajes, conocimientos y habilidades necesarios para superar con éxito el correspondiente examen extraordinario de Cibernética y Computación II (2016).

En este orden de ideas, y a fin de fortalecer la trayectoria académica de quien sustente el respectivo examen extraordinario de la asignatura Cibernética y Computación II, correspondiente al sexto semestre del CCH, el contenido de esta guía se ha estructurado de tal manera que, por cada Unidad se incluyen:

- **Los aprendizajes y temática** correspondientes.
- **Conceptos clave**, referidos en su contenido.
- Por cada aprendizaje, un apartado de **Actividades de aprendizaje teórico–prácticas**, y otro de **Autoevaluación del aprendizaje**.

  Nota: La ausencia del apartado Autoevaluación del aprendizaje en algunos Aprendizajes, se debe a que se trata de aprendizajes implícitos en otros.
- Las **Respuestas a las autoevaluaciones** propuestas en el apartado Autoevaluación del aprendizaje.
- Las referencias o fuentes consultadas.

De igual manera, a fin de enriquecer la preparación estudiantil, casi al final de este trabajo se incluyen los Enlaces de Código, mismos que vinculan el sitio web donde se pueden consultar directamente los códigos de programas, ya sean en su versión completa o segmentos de programa. Asimismo, al final esta obra se incluye un simulador de examen extraordinario, con el que cada estudiante sustentante podrá verificar su progreso en la preparación hacia la presentación del examen extraordinario respectivo.

Vale la pena resaltar que, con la intención de favorecer la didáctica para preparar la presentación del examen extraordinario correspondiente, la presente guía, en general, está permeada por una variedad de actividades didácticas tales como breve explicación del aprendizaje y de la temática, los códigos respectivos, sopas de letras, relación de columnas, cuestionarios, completar segmentos de códigos, programas completos, relación de columnas con imágenes, y clasificación de elementos con imágenes.

Los autores estamos convencidos de que lo anterior, permitirá reforzar y fortalecer la comprensión estudiantil, respecto de los conceptos teóricos a través de la práctica y la aplicación directa de la programación.

En esta Guía, el uso de recursos didácticos basados en imágenes y actividades interactivas busca que la adquisición de los aprendizajes, habilidades y conocimientos de esta asignatura sea más amena, dinámica y accesible, y permite que cada estudiante se adapte a estudiarla, independientemente del estilo de aprendizaje que posea. Inherentemente, al prepararse con esta guía, cada estudiante, no sólo aumentará sus posibilidades de aprobar el examen extraordinario, sino también fortalecerá, enriquecerá y consolidará una base sólida con respecto a la programación orientada a objetos con Java, lo que será esencial para su futuro académico y profesional.

El impacto de esta guía trasciende más allá del ámbito individual, ya que también incidirá en un beneficio para el Colegio de Ciencias y Humanidades, porque coadyuvará en que sus estudiantes que egresen posean un dominio sólido de los aprendizajes, habilidades y conocimientos relativos a la programación orientada a objetos con Java, referidos en la asignatura de Cibernética y Computación II, lo que se reflejará en la calidad educativa y el compromiso del CCH, invocados en la excelencia académica. Invitamos a la comunidad

estudiantil a que aproveche al máximo esta Guía, a fin de prepararse de manera integral y confiada para su examen extraordinario.

Apreciables estudiantes de sexto semestre, esta guía fue realizada para ustedes; y sabemos que los desafíos pueden parecer grandes, pero recuerden que cada quien posee el potencial necesario para superarlos. Tienen en sus manos un recurso diseñado para acompañarles en este último paso hacia su egreso; utilícenlo con confianza y dedicación; y recuerden que el esfuerzo que inviertan ahora se reflejará en sus logros futuros.

¡Ustedes son lo mejor de su Generación,
confíen en sus capacidades y avancen con determinación hacia su meta!

Atentamente,

Los Autores

Angélica Vianey Zavaleta Hernández
angelica.zavaleta@cch.unam.mx
Antonio Granillo Martínez
antonio.granillo@cch.unam.mx
César Ramírez Ortega
cesar.ramirez@cch.unam.mx
Miguel Ángel Rodríguez Padilla
miguel.rdz@cch.unam.mx
Pedro Ernesto Nequiz Medina
pedro.nequizmedina@cch.unam.mx

ÍNDICE

Instrucciones Generales para el Uso de esta Guía:

A fin de aprovechar al máximo el potencial de esta Guía, y afrontar exitosamente el examen extraordinario correspondiente; se sugiere seguir las instrucciones siguientes:

1. **Conoce la guía:**
   - Comienza por leer detenidamente la introducción. En ella encontrarás el propósito de esta guía y cómo te beneficiará en tu preparación.
   - Familiarízate con los temas que se desarrollaron en la guía: Lenguaje de programación orientada a objetos con Java, estructuras de control, polimorfismo, constructores, herencia e interfaz gráfica de usuario.
2. **Organiza tu Tiempo:**
   - Planifica tu estudio. Distribuye el tiempo de manera equitativa entre los diferentes temas.
   - Establece metas de estudio y celebra tus avances.
3. **Participa activamente:**
   - No te limites a leer. Participa activamente en cada actividad.
   - Las sopas de letras, las relaciones de columnas, los cuestionarios y los ejercicios de completar código están diseñados para reforzar tu aprendizaje de manera práctica y entretenida.
4. **Aprende visualmente:**
   - Presta especial atención a las actividades con imágenes. La relación de columnas y la clasificación de elementos con imágenes te ayudarán a visualizar y comprender mejor los conceptos.
5. **Practica la programación:**
   - Dediquen tiempo a completar los segmentos de código y los programas completos. La práctica es esencial para dominar la programación en Java.

- Los programas en Java presentados en esta guía están diseñados para que puedas practicarlos y comprender mejor los conceptos explicados. Estos programas pueden ser probados en cualquier IDE (Entorno de Desarrollo Integrado) de tu preferencia, como BlueJ, Eclipse, NetBeans, etc. Solo necesitas descargar los archivos disponibles en la sección de "Enlaces de Código" y cargarlos en el entorno de tu elección para ejecutarlos y experimentar con ellos.

6. **Autoevaluación continua:**
   - Utiliza las actividades de autoevaluación que se encuentran al final de cada aprendizaje. Identifica tus áreas de oportunidad y repasa los temas hasta que los comprendas.
   - Al finalizar cada tema, reflexiona sobre lo que has aprendido y cómo puedes aplicarlo.
7. **Recursos Adicionales:**
   - Si encuentras algún tema que te resulte difícil, no dudes en buscar recursos adicionales, como libros, tutoriales en línea o videos explicativos.
   - Recuerden que sus profesores están disponibles para resolver sus dudas en el Programa Institucional de Asesorías (PIA).
8. **Confía en ti mismo:**
   - Mantén una actitud positiva y confía en tus capacidades.
   - Recuerda que esta guía es una herramienta de apoyo, pero tu dedicación y esfuerzo son los que marcarán la diferencia.
9. **Beneficios adicionales:**
   - Al usar esta guía, no solo se preparan para el examen, sino que también fortalecen habilidades valiosas para su futuro académico y profesional.
   - Contribuye al prestigio de la ENCCH al demostrar un alto nivel de competencia en la asignatura de Cibernética y Computación II.

Recuerda que el esfuerzo de hoy se traduce en los logros del mañana.

¡Mucho éxito en tu preparación!

ÍNDICE

Propósito.

Al finalizar la unidad, cualquier estudiante conocerá las características del lenguaje de programación orientada a objetos con Java, y su entorno de desarrollo, mediante la definición de clases, atributos y métodos para la implementación de objetos en programas.

ÍNDICE

1.0 Conceptos Clave.

Clase.

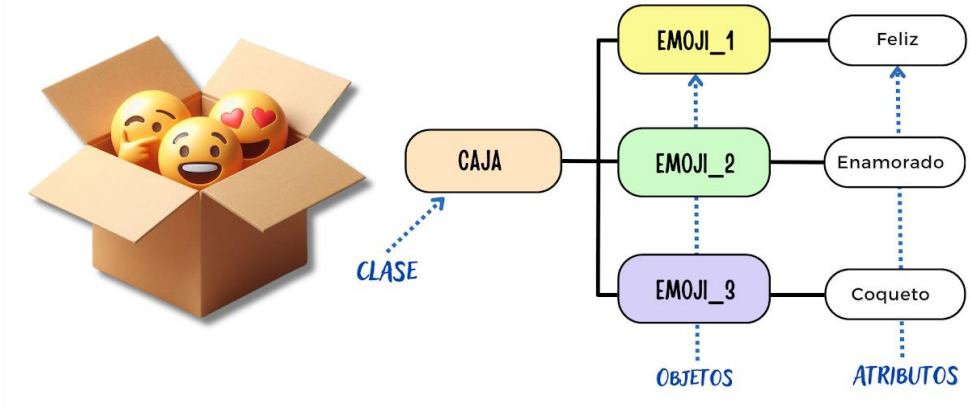
Es como el molde de una entidad o una plantilla y puede ser replicado tanto como sea necesario, nos servirá para crear objetos de forma ya predefinida. Cada clase tiene definida una serie de variables y funciones, o también conocidos como atributos y métodos, que operan en la clase.

Objeto.

Dentro de la programación, es una entidad donde se definen sus características y su comportamiento o acciones que pueda realizar. Los elementos de un objeto se dividen en dos categorías principales: propiedades y métodos

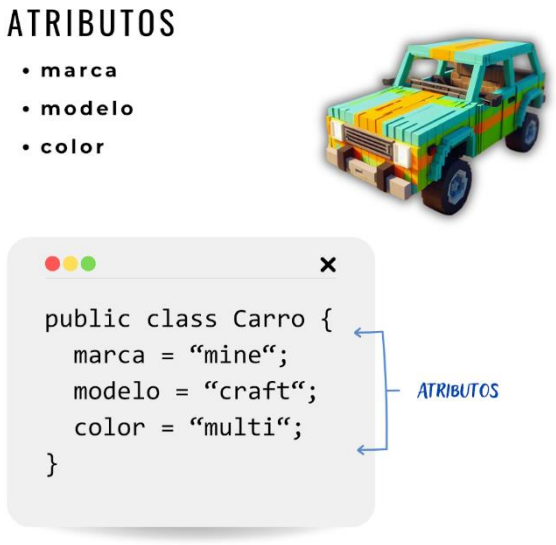
Identidad.

Es la propiedad (atributo) propia del objeto, que lo describe con sus atributos, que es una representación concreta y específica de una clase.



Atributos o Características.

Propiedades específicas que son guardadas en variables cuyos valores pueden ser diferentes con respecto a otros objetos.



Comportamiento (Método).

Los objetos pueden realizar acciones específicas descritas por instrucciones que conforman los métodos y cada método puede contener N cantidad de variables y acciones a realizar.

Abstracción.

Son las características de un objeto, además de las acciones que pueda realizar. Por ejemplo, si se tiene que el objeto a programar es un automóvil, entonces se aplica el concepto de abstracción, es decir se definen sus características y acciones.

Acciones: avanzar, retroceder, encendido de luces ….

Instancia.

Es la creación de un objeto a partir de una clase que actúa como una plantilla. Cada instancia tiene sus propios valores y propiedades, lo que le permite representar objetos únicos en un programa. La instanciación está relacionada con el encapsulamiento y el polimorfismo.

Operadores aritméticos.

Son los símbolos que permiten identificar los procesos que se realizarán. Estos, por lo general se usan para realizar las operaciones aritméticas conocidas que implican el

cálculo de valores, dentro de los básicos encontramos (suma +, resta -, multiplicación *, división /).resultantes se pueden combinar con otras expresiones casi indefinidamente. La sintaxis de las expresiones aritméticas.

Desarrollo ágil de software.

EI desarrollo ágil de software es un conjunto de metodologías que tratan de implementar software con más rapidez y mediante el uso de menos recursos.

Visite los sitios:

Agile Alliance (www.agilealliance.org) y

Agile Manifesto (www.agilemanifesto.org).

JVM.

Un compilador de Java no compila todas las clases en un programa de código de máquina. En su lugar, compila cada clase de forma independiente y, además, no en código de máquina, sino en un código intermedio especial (código de bytes). El código de bytes se compila en código de máquina cuando se inicia el programa.

Hay un programa especial para esto llamado máquina virtual ***Java*** (***JVM***). Primero se inicia y luego el programa se compone de ***bytecode***. Luego, la ***JVM*** compilará el código de bytes en código de máquina antes de que se ejecute el programa.

Programa en Java

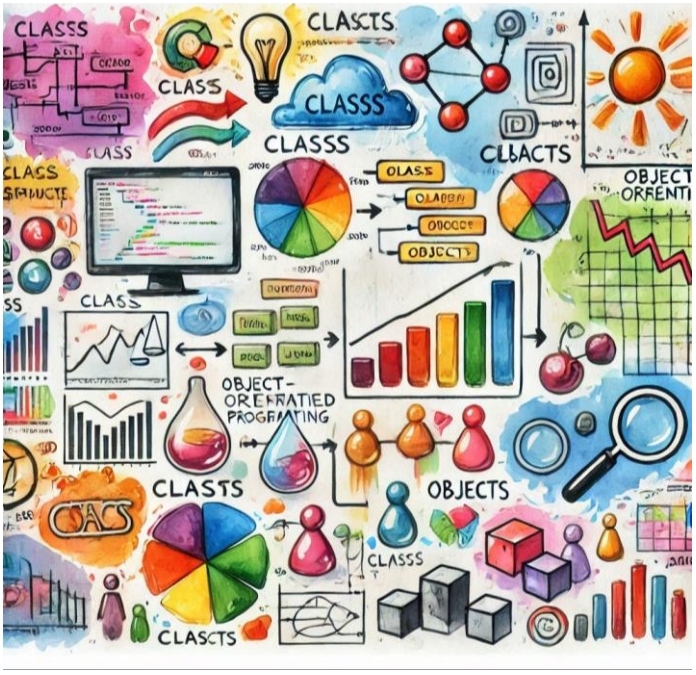
Un programa que consta de códigos especiales independientes (bytecode)

Un programa compuesto por código de máquina.

ÍNDICE

APRENDIZAJE 1.1

En relación con *Java*, como lenguaje orientado a objetos, conoce tanto los conceptos básicos, como la organización general de un programa.



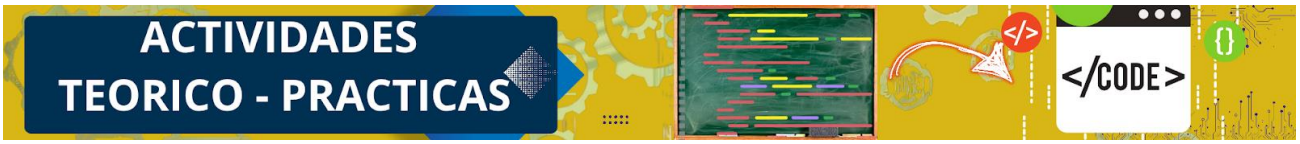
Temática:

Conceptos básicos de la programación orientada a objetos.

- Identidad.
- Atributos o características.
- Comportamiento.
- Abstracción.
- Encapsulamiento

Organización general de un programa en *Java*.

- Comentarios.
- Uso de bibliotecas.
- Identificadores.
- Palabras reservadas.
- Sentencias.
- Tipos de datos primitivos.
- Bloque de código.
- Operadores.
- Expresiones.



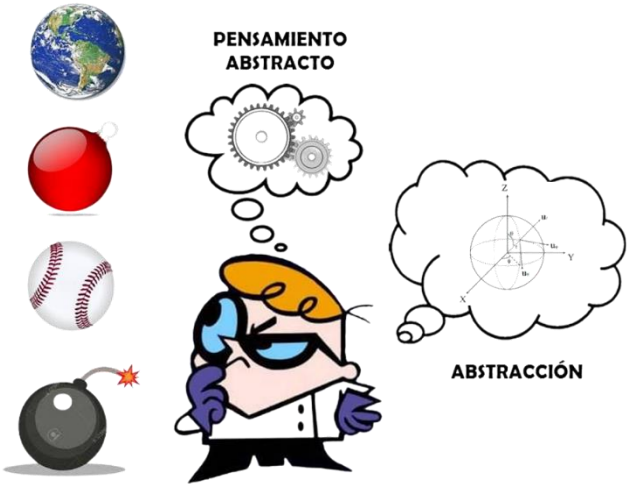
ÍNDICE

1.1.1. Actividades de aprendizaje teórico–prácticas.

Un objetivo clave de ***Java*** es poder escribir programas que se ejecuten en una gran variedad de sistemas computacionales y dispositivos controlados por computadora. A esto se le conoce algunas veces "*como escribir una vez, ejecutar en cualquier parte*"; y, en ***Java***, se le identifica como ***Clase***, como una plantilla o molde para generar más objetos similares.

Actividad:

Toma de ejemplo un elemento del mundo real y de forma descriptiva; enumera los elementos que componen a este, para poder generar lo que serían sus atributos de objeto y sus acciones (métodos) que podrían ser cada uno de ellos.



Ejemplo Perro:

Encapsulamiento: Las clases y sus objetos encapsulan (envuelven) sus atributos y métodos. Los objetos se pueden comunicar entre sí, pero todo atributo de una ***Clase*** tiene un grado de visibilidad de acuerdo con las necesidades del programa en *Java*. La encapsulación sirve para proteger los datos. El grado de visibilidad está determinado por las siguientes palabras reservadas:

- ***public:*** Accesible desde cualquier clase.
- ***private:*** *Accesible desde la clase que la definió.*
- ***protected****: Accesible desde la clase actual, sus descendientes como subclases o el paquete del que forma parte.*

En ***Java***, para asegurar que un programa esté encapsulado, se definen los atributos como privados (***private***), dado que deben estar siempre ocultos a los demás y se acceden con los métodos ***getter*** (lectura) y ***setter*** (escritura).

Existe la convención de nombrar a estos métodos utilizando los prefijos **get**, **set**, seguido del nombre del atributo. Véase el ejemplo siguiente.

Organización General de un programa en *Java*

```java
public class Persona {
    // Atributos
    private String nombre;

    // Constructor
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
    }

    // Métodos get y set para el atributo "nombre"
    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

Tipos de comentarios en *Java*

Insertamos comentarios para documentar los programas y mejorar su legibilidad. El compilador de Java ignora los comentarios, de manera que la computadora no hace nada cuando el programa se ejecuta. Por convención, comenzamos cada uno de los programas con un comentario, el cual indica el nombre del archivo.

De una sola línea:

Empieza con doble diagonal lo cual indica que es un comentario y no afectará al código

// Ejemplo de comentario

En cada nueva línea se debe comenzar nuevamente con la doble diagonal ***//*** siendo esta la forma en que realizará la lectura de nuestro código, línea por línea lo irá recorriendo el compilador.

// Programa para imprimir texto.

Multilínea

Java también cuenta con un tipo de comentario: los comentarios ***Javadoc***, que están delimitados por ***/***** y terminan con la secuencia ****/***. El compilador ignora todo el texto entre los delimitadores. Estos comentarios nos permiten incrustar la documentación de manera directa en nuestros programas, estos se pueden distribuir en varias líneas.

```
public class Carro {
 /**
 * Frase corta descriptiva
 * Descripción de la clase
 * @author Nombre Apellido
 * @version 0.1, 2024/05/01
 */
}
```

Para escribir un programa en ***Java*** lo primero que se debe hacer es crear una clase, para esto se puede emplear un editor de texto o un entorno de programación (***IDE***).

Buenas prácticas: Por convención, se requiere que todos los programas comiencen con un comentario que explique su propósito, el autor, la fecha y la hora de la última modificación de este.

Compilador:

Traduce el código que escribimos en lenguaje de programación a instrucciones que la computadora entienda. Es como traducir un texto de un idioma a otro.

Máquina Virtual de *Java*:

Es un entorno que actúa como un intermediario entre el código que escribimos y el hardware de la computadora. Es como una especie de computadora donde nuestro código se ejecuta, independientemente del sistema operativo que estemos utilizando. Permite que el mismo programa se ejecute en diferentes dispositivos sin necesidad de modificaciones.

Entorno de Desarrollo Integrado (*IDE*):

Es un editor de texto que contiene diferentes herramientas que ayudan a comprender e identificar mejor el código, la estructura, orden, uso correcto de palabras, etc. Es similar a un asistente que nos ayuda en todo el proceso de desarrollo del programa.

Uso de bibliotecas

Las bibliotecas en el lenguaje de programación ***Java*** le brindan al desarrollador una amplia posibilidad para crear aplicaciones, una biblioteca en Java es un conjunto de clases, en donde cada clase posee una serie de métodos y atributos los cuales facilitan la tarea del programador, las bibliotecas en ***Java*** nos permiten reutilizar código, utilizando los métodos y atributos de una clase, es decir evita la tarea de tener que desarrollar un programa el cual ya se encuentra disponible para ser utilizado. Ejemplo: ***import java.util.Scanner;***

Paquete	Descripción
java.lang	Contiene clases fundamentales de ***Java***, como ***String***, ***System***, ***Object***, etc.
java.util	Proporciona clases e interfaces de utilidad general, como listas, conjuntos, mapas, y clases de utilidad.
java.awt	Proporciona clases para crear interfaces gráficas de usuario (***GUI***).
javax.swing	Extensión de ***java.awt*** que proporciona componentes de interfaz gráfica más avanzados.
java.math	Contiene clases para operaciones matemáticas más complejas, como números grandes y precisión arbitraria.
java.util.Scanner	Ofrece la clase ***Scanner*** que permite leer datos de diferentes tipos de fuentes de entrada, como el teclado o archivos.

Identificadores

Los identificadores son los nombres que el programador asigna a variables, constantes, clases y métodos en un programa de cómputo. Para escribir un identificador en el lenguaje de programación ***Java*** se deben de cumplir las siguientes reglas:

- Los caracteres permitidos son los caracteres alfanuméricos (***AZ***), (***az***), (***0-9***).
- ***Java*** distingue entre mayúsculas y minúsculas.
- No pueden empezar con un dígito.
- No tienen límite en la longitud, pero es recomendable que el nombre del identificador tenga una longitud entre 4 y 15 caracteres.
- No puede ser una palabra reservada del lenguaje de programación ***Java***.
- No se aceptan espacios ni acentos, ni caracteres especiales, excepto el guion bajo "_".
- Se recomienda que el nombre del identificador haga referencia a su contenido.

Ejemplos de identificadores válidos:

edad	*nombre*	*sueldo*	*impuesto*	*calificа1*
sueldo_neto	*edad*	*nombre*	*_sueldo*	*SueldoNeto*

Las mayúsculas y minúsculas se consideran diferentes:

Palabras reservadas

Téngase en cuenta que, las palabras reservadas no pueden ser utilizadas como nombres de identificadores.

En ***Java***, las palabras reservadas son aquellas que tienen un significado especial y están destinadas para un propósito específico en el lenguaje.

Estas palabras están predefinidas por el lenguaje y no pueden ser utilizadas como nombres de variables, métodos, clases, etc.; y sirven para definir la estructura del código y controlar el flujo del programa.

while	***continue***	***for***	***new***	***switch***
true	***default***	***null***	***package***	***void***
boolean	***do***	***if***	***private***	***this***
break	***double***	***implements***	***protected***	***throw***
class	***else***	***import***	***public***	***while***
case	***float***	***instanceof***	***return***	***const***
catch	***extends***	***int***	***super***	***try***
char	***false***	***interface***	***static***	***void***

***class*:** define el comienzo de la declaración de una clase.

import: declara la importación de bibliotecas en ***Java***.

new: es el operador de creación de instancias.

Sentencias

Un programa en cualquier lenguaje de programación se compone de un conjunto de sentencias, la sentencia es una orden que se le da al programa para realizar una tarea específica con el propósito de resolver un problema, en el lenguaje de programación ***Java*** al final de una sentencia debemos de colocar un punto y coma (***;***).

En conjunto, las comillas dobles y los caracteres entre ellas son una cadena, lo que también se conoce como cadena de caracteres. El compilador no ignora los caracteres de espacio en blanco dentro de las cadenas. Éstas no pueden abarcar varias líneas de código.

Hay diferentes tipos de comandos. El lenguaje ***Java*** tiene un comando para cada ocasión. Cada comando define alguna acción específica. **Un punto y coma** (**`;`**) va al final de cada comando.

Ejemplo. Indica a la computadora que muestre un mensaje, al imprimir los caracteres contenidos entre los signos de comillas dobles (las comillas dobles no se muestran en la salida).

`println`: Imprime el texto que se encuentra entre comillas y realiza un salto de línea.

`print`: Imprime el texto que se encuentra entre comillas y no realiza un salto de línea.

println

```
// Imprimirá el resultado

System.out.println("Bienvenido");
System.out.println(" a la programación en Java!");
```

Salida:
> Bienvenido
> a la programación en Java!

print

```
// Imprimirá el resultado

System.out.print("Bienvenido");
System.out.print(" a la programación en Java!");
```

Salida:
> Bienvenido a la programación en Java!

En la figura previa, el argumento ***`Bienvenido a la programación en Java`***, entre paréntesis y el punto y coma (**`;`**), se conoce como una instrucción. Por lo general, un método contiene una o más instrucciones que realizan su tarea. La mayoría de las instrucciones terminan con punto y coma (**`;`**).

Ejemplo 2:

CÓDIGO	RESULTADO

`System.out.println("Amigo");`
`System.out.println("IsThe");`
`System.out.println("Best");`
	Amigo
	IsThe
	Best

`System.out.print("Amigo");`
`System.out.println("IsThe");`
`System.out.print("Best");`
	AmigoIsThe
	Best

`System.out.print("Amigo");`
`System.out.print("IsThe");`
`System.out.print("Best");`
	AmigoIsTheBest

El *String*

Para asignar una cadena en ***Java***, debe escribir el *texto de la cadena* entre *comillas* (***" "***). Ejemplo:

CÓDIGO	EXPLICACIÓN
`String s = "Amigo";`	El identificador s contiene: "Amigo"
`String s = "123";`	El identificador s contiene: "123"
`String s = "Bond 007";`	El identificador s contiene: Bond 007

Así mismo te permite realizar operaciones, en el siguiente ejemplo se indica que muestre dos tipos de mensaje el primero será sin estas comillas dobles y en el segundo se agregara como una cadena de caracteres, observa como el resultado es diferente al colocarlas o retirar estas comillas dobles.

Ejemplo 3:

CÓDIGO	RESULTADO
`System.out.println( 1 + 1 );`	En la pantalla se muestra: 2
`System.out.println( "1" + "1" );`	En la pantalla se muestra: 11

En el primer resultado mostrará la operación matemática (sumatoria), obteniendo como resultado 2, mientras que en el segundo mensaje realizará una unión (concatenación) puesto que la computadora considera que "1" entre comillas es texto y no numérico. El programa completamente escrito, requerirá una declaración de una clase y un método main.

Tipos de datos primitivos o básicos

Java, aunque está orientado a objetos, posee elementos que no son objetos; estos son los llamados tipos primitivos, que representan a los números y los caracteres, y sólo se pueden realizar operaciones sobre ellos como la suma y resta, y otras que están definidas por el lenguaje directamente, mismos que se clasifican en:

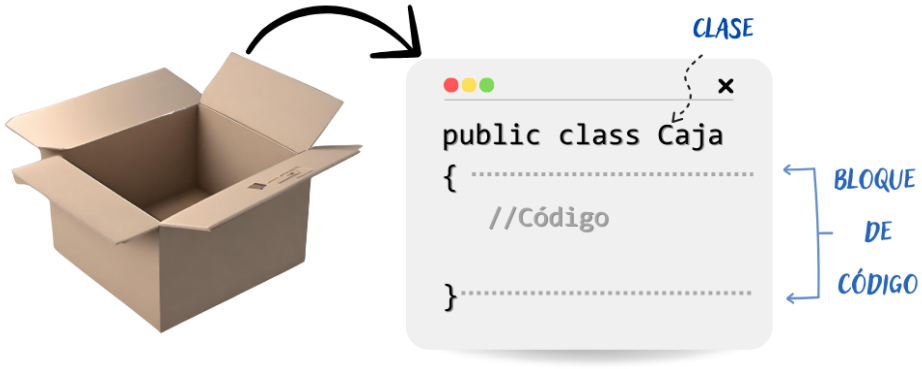
NOMBRE	TIPO	TAMAÑO	VALOR POR DEFECTO	RANGO
boolean	Lógico	1 bit	***False***	***True / false***
char	Carácter	16 bits	***Null***	***Unicode***
byte	Número entero	8 bits	***0***	***-128 a 127***
short	Número entero	16 bits	***0***	***-32,768 a 32,767***
int	Número enter	32 bits	***0***	***-2,147,483,648 a 2,147,483,649***
long	Número entero	64 bits	***0***	***-9,223,372,036,854,775,808 a 9,223,372,036,854,775,808***
float	Número real	32 bits	***0***	***34 x 10-38 a 34 x 1038***
double	Número real	64 bits	***0***	***1.8 x 10-308 a 1.8 x 10308***

***String*:**

Si bien ***String*** no es un tipo primitivo, ya que es una clase que se utiliza para manejar las cadenas de caracteres de forma similar a un dato primitivo, almacena los 255 caracteres de la tabla ***ASCII***.

Bloque de código

Es la sección de código delimitada por llaves.



Operadores aritméticos

Los operadores aritméticos se usan para calcular un valor de dos o más números, ya sea: sumar, restar, multiplicar, dividir, o cambiar el signo de un número de positivo a negativo o viceversa. Son elementos fundamentales que permiten la manipulación de datos y variables. Estas operaciones pueden abarcar desde simples cálculos matemáticos hasta evaluaciones de condiciones lógicas.

En ***Java*** se pueden operar los tipos numéricos primitivos, mediante los operadores matemáticos cuyas funciones están compiladas en la máquina virtual de Java:

OPERADOR	DESCRIPCIÓN	EJEMPLO Si: *a=5* *b= 3*	RESULTADO
+	Suma	*a + b*	*8*
–	Resta	*a – b*	*2*
***	Producto	*a * b*	*15*
/	División	*a / b*	*1.6*
%	Módulo	*a % b*	*2*

Tabla operadores aritméticos.

En la tabla siguiente, se utilizan símbolos o palabras reservadas para llevar a cabo acciones u operaciones de manera eficiente:

CÓDIGO	RESULTADO
int x = 1; ***int y = x*2;*** ***int z = 5*y*y + 2*y + 3;***	***x = 1*** ***y = 2*** ***z = 20 + 4 + 3***, o sea, ***z = 27***
int a = 5; ***int b = 1;*** ***int c = (a-b) * (a+b);***	***a = 5*** ***b = 1*** ***c = 4 * 6***, o sea, ***c = 24***
int a = 64; ***int b = a/8;*** ***int c = b/4;*** ***int d = c*3;***	***a = 64*** ***b = 8*** ***c = 2*** ***d = 6***

Operador de asignación: Es un operador binario que asigna el valor de la expresión que se encuentra a la derecha del operador, a la variable que se encuentra a la izquierda del operador.

OPERADOR	DESCRIPCIÓN	EJEMPLO Si: ***b= 9***	RESULTADO
=	Operador de asignación	***a = b***	***a = 9***

Operadores aritméticos incrementales: Estos operadores realizan una acción sobre un solo operando, el operando debe ser numérico, el resultado es del mismo tipo que el operando y siempre incrementa de uno en uno.

OPERADOR	DESCRIPCIÓN	EJEMPLO Si: ***b= 9***	RESULTADO
++	Incremento	***9++;***	***10***
variable ++	primero se utiliza el contenido de la variable y después se incrementa su valor	***dato=25;*** ***dato++;***	Se utiliza ***dato=25*** Después incrementa y, ***dato=26***
++variable	primero se incrementa el valor de la variable y luego se utiliza	***dato=25;*** ***++dato;***	Primero incrementa ***dato=26*** Después se utiliza ***dato=26***
--	Decremento	***10--***	***9***

Operadores relacionales: Comparan dos operandos de tipo primitivo (***byte***, ***int***, ***char***, ***float***, etc.), relacionándolos entre sí (de ahí su nombre). Devuelven un valor de tipo ***boolean***, este valor dependerá de si la comparación entre ambos operandos es cierta (***true***) o falsa (***false***).

OPERADOR	DESCRIPCIÓN	SINTAXIS	RESULTADO
>	Mayor que	***1 > 2***	***Falso***
>=	Mayor o igual que	***1 >= 2***	***Falso***
<	Menor que	***1 < 2***	***Verdadero***
<=	Menos o igual que	***1 <= 2***	***Verdadero***
==	Igual que	***1 == 1***	***Verdadero***
!=	Distinto de	***1 != 2***	***Verdadero***

Operadores lógicos o booleanos: Evalúan una expresión en la que están implicados uno o varios operandos. Devuelven un valor de tipo ***boolean***, este valor dependerá de si la comparación entre ambos operandos es cierta (***true***) o falsa (***false***).

OPERADOR	DESCRIPCIÓN	SINTAXIS	RESULTADO
&&	***And*** lógico ***(y)***	***1 && 2***	***uno*** y ***dos***
||	***Or*** lógico ***(o)***	***1 || 2***	***uno*** ó ***dos***
!	Negación lógica ***(no)***	***!2***	diferente de ***dos***

Tabla. Operadores Lógicos

Expresiones: Es una secuencia de operadores, variables, métodos, que devuelven un valor, se debe respetar el mismo tipo de dato con el que se envían y reciben en cada método; lo mismo ocurre cuando se declara el tipo de variable.

Ejemplos:

```
// Se almacenan los datos en la variable precio.
precio = producto * numero
iva = (precio * 16) / 200
residuo = 5 % 3
total = calcular()  //se invoca el método calcular
```

ÍNDICE

1.1.2 Autoevaluación del aprendizaje:

1. Sopa de letras.

Instrucciones: Encuentra la lista de palabras clave en la sopa de letras.

Á	Ü	Í	S	G	I	Ü	W	É	F	Ó	D	J	Q	L	Ü	Ñ	M	W	W	Ñ	Í	S	U	Y
Á	O	B	M	X	D	X	L	Ñ	F	T	P	Í	I	A	C	Y	R	Y	L	Ó	M	O	B	Ó
Z	T	Z	N	P	Ú	Q	K	L	V	U	O	D	K	X	E	C	W	Ü	Z	P	C	T	L	Í
K	N	P	G	Ó	Q	M	T	Q	C	J	E	C	D	E	O	É	P	H	Y	E	S	U	O	H
E	E	D	Ó	H	I	Ü	Q	X	É	N	V	Q	Í	É	Z	A	G	H	É	O	U	B	Q	O
L	I	É	Z	A	G	C	Z	É	T	J	B	Ü	N	É	L	U	U	G	V	F	V	I	U	Ñ
V	M	Ü	R	E	F	W	C	I	I	Í	G	E	É	A	X	Q	É	I	T	Q	N	R	E	Z
J	A	D	E	I	Ó	E	D	A	O	H	Q	É	B	W	Ü	Í	T	É	B	W	W	T	D	C
Á	T	C	T	Í	V	A	R	I	R	B	Ñ	R	Ó	R	E	I	U	H	E	Í	L	A	E	C
K	R	C	Ú	Z	D	E	E	L	Ó	T	A	M	S	Ú	M	G	C	É	L	N	O	B	C	R
J	O	J	R	G	F	H	O	É	G	S	S	Y	T	I	I	C	B	U	S	Y	C	L	Ó	S
Y	P	S	E	Q	J	Q	Y	E	R	R	G	B	R	Í	Ü	S	O	Ó	T	M	U	B	D	E
I	M	G	I	C	W	Á	D	E	É	T	X	P	A	I	L	Í	Ú	A	O	Á	Í	P	I	N
Q	O	Ü	P	L	V	O	S	Y	T	R	S	A	D	N	B	D	Ú	K	S	R	H	Í	G	O
Y	C	H	É	Ó	R	E	Á	Í	F	O	D	R	M	Í	V	T	J	A	Ü	S	Z	C	O	I
C	H	R	H	P	R	Ñ	B	U	T	Q	Ü	Ü	Y	P	U	J	U	Ú	Z	J	H	S	Ó	S
Í	Á	Q	P	V	Í	K	U	A	D	D	S	O	I	R	A	T	N	E	M	O	C	P	G	E
Y	R	Ó	A	R	Ó	U	D	A	Q	J	Z	Ñ	H	G	L	J	C	J	K	H	L	X	L	R
Z	G	D	G	E	E	E	E	B	K	O	B	W	Z	Y	R	C	C	B	K	Á	S	D	A	P
U	A	D	L	I	D	É	S	Ü	H	L	D	S	E	R	O	D	A	R	E	P	O	U	Ü	X
S	O	V	Á	S	M	É	Ó	X	N	X	W	I	É	Q	G	C	M	Ü	X	E	W	G	G	E
U	H	N	O	K	J	S	E	R	O	D	A	C	I	F	I	T	N	E	D	I	W	Ñ	K	I
V	R	P	L	Ó	V	D	M	M	Ñ	S	A	I	C	N	E	T	N	E	S	O	A	P	D	F
Á	I	V	W	S	U	S	O	D	E	B	I	B	L	I	O	T	E	C	A	S	Ñ	A	R	G
T	U	N	E	O	E	N	C	A	P	S	U	L	A	M	I	E	N	T	O	C	L	T	F	F

- COMENTARIOS
- IDENTIFICADORES
- PALABRAS RESERVADAS
- SENTENCIAS
- TIPOS DE DATOS PRIMITIVOS
- BLOQUE DE CÓDIGO
- OPERADORES
- EXPRESIONES
- USO DE BIBLIOTECAS
- IDENTIDAD
- ATRIBUTOS
- COMPORTAMIENTO
- ABSTRACCIÓN
- ENCAPSULAMIENTO

2. Relación de columnas.

Instrucciones: En la siguiente tabla, se presentan comentarios, uso de bibliotecas, identificadores, palabras reservadas, sentencias, tipos de datos primitivos, bloque de código, operadores, expresiones utilizadas en la programación en ***Java***. Relaciónalos con su descripción correcta y selecciona la respuesta que muestra la correspondencia correcta.

CONCEPTO	DEFINICIÓN
I. Comentarios II. Bibliotecas III. Identificadores IV. Palabras reservadas V. Sentencias VI. Tipos de datos primitivos VII. Bloque de código VIII. Operadores IX. Expresiones	a) Secuencia de símbolos que realiza una operación sobre uno o más valores. b) Instrucciones escritas en ***Java*** que realizan una acción específica. c) Palabras con significado especial en ***Java***, no pueden ser usadas como nombres de variables o métodos. d) Conjunto de clases definidas por Java para ser reutilizables y facilitar la tarea del programador, por ejemplo: ***java.util.Scanner***. e) Conjunto de instrucciones encerradas entre llaves. f) Información dentro del código que el compilador ignora, puede ser de una sola línea o múltiples líneas. g) Nombres que el programador asigna a las variables, métodos o clases en ***Java***. h) Valores básicos que maneja ***Java***, como ***int***, ***double***, ***boolean*** y ***char***. i) Tipo de dato primitivo que representa números enteros en un rango específico. j) Secuencia de operadores, variables, métodos, que devuelven un valor.

A) I:a - II:c - III:d - IV:e - V:g - VI:f - VII:d - VIII:b - IX:j
B) I:j - II:i - III:g - IV:c - V:b - VI:a - VII:e - VIII:d - IX:e
C) I:i - II:a - III:c - IV:b - V:j - VI:h - VII:e - VIII:b - IX:d
D) I:f - II:d - III:g - IV:c - V:b - VI:h - VII:e - VIII:a - IX:j

Programa que identifica el mayor de tres números.

Instrucciones: Completa los espacios en blanco colocando el número del código correspondiente y selecciona la respuesta que muestra la correspondencia correcta.

1.- ***int***	2.- clase	3.-***Scanner***	4.- ***// Mostrar el resultado***
5.- ***java.util.Scanner***	6.- ***{***	7.- ***}***	8.- ***nextInt***
9.- ***mayor = num3***	10.- ***System.out.print("Ingrese el tercer número: ");***		

a)
b)
c)
d)
e)
f)
g)
h)
i)

```
import __________;

public________MayorDeTresNumeros {
    public static void main(String[] args) _________
        Scanner teclado = new _________(System.in);

        // Solicitar los tres números al usuario
        System.out.print("Ingrese el primer número: ");
        _____num1 = teclado.nextInt();

        System.out.print("Ingrese el segundo número: ");
        int num2 = teclado.nextInt();


        _________
        int num3 = teclado.nextInt();

        // Determinar el número mayor
        int mayor;

        if (num1 >= num2 && num1 >= num3) {
            mayor = num1;
        } else if (num2 >= num1 && num2 >= num3) {
            mayor = num2;
        } else {
            ------------
        }


        ________
        System.out.println("El número mayor es: " + mayor);

        teclado.close();
    ----------
}
```

Opciones de respuesta:

A) a:5 - b:2 - c:6 - d:3 - e:1 - f:10 - g:9 - h:4 - i:7

B) a:1 - b:3 - c:5 - d:7 - e:2 - f:4 - g:8 - h:6 - i:5

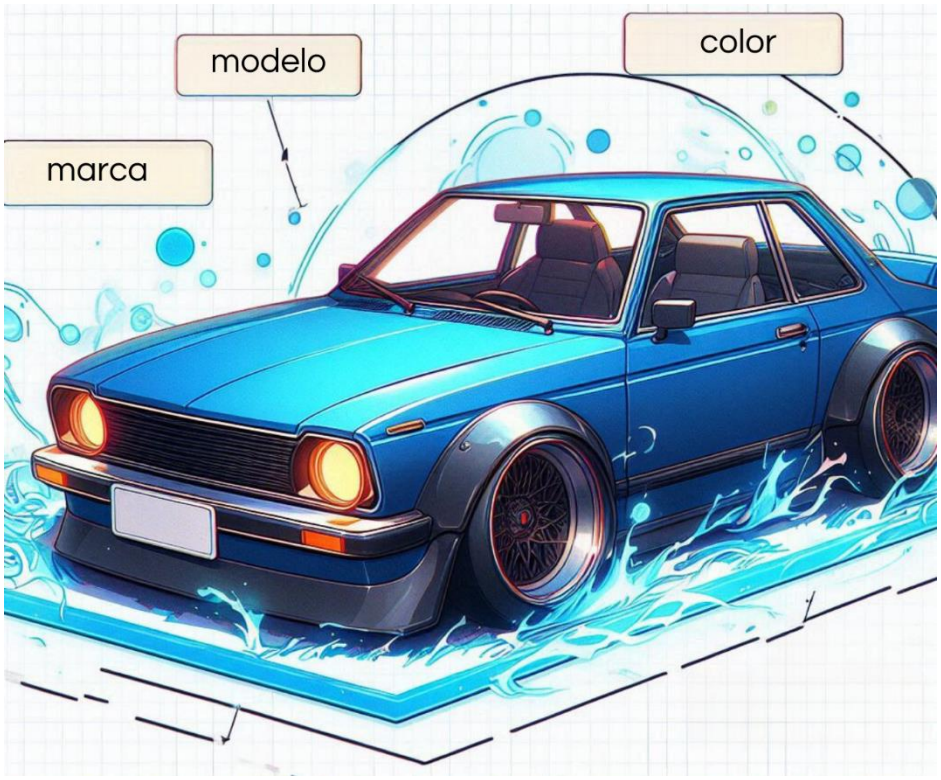
C) a:5 - b:2 - c:3 - d:6 - e:10 - f:1 - g:4 - h:9 - i:7

D) a:7- b:5 - c:3 - d:6 - e:9 - f:4 - g:10 - h:8 - i:1

ÍNDICE

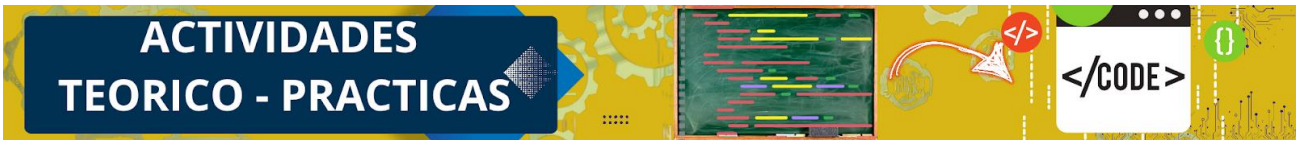
APRENDIZAJE 1.2

Describe, del lenguaje *Java*, los conceptos de Clase y atributo, y los implementa en programas.



Temática:

- Definición.
- Declaración.
- Atributos.
  - Definición.
  - Declaración.
  - Niveles de visibilidad.
- Implementación.



ÍNDICE

1.2.1. Actividades de aprendizaje teórico–prácticas.

En la programación orientada a objetos (***POO***), las clases y los objetos son los pilares fundamentales que nos permiten modelar elementos del mundo real dentro del ***software***. Estos conceptos son esenciales para definir la estructura y el comportamiento de los elementos que queremos representar. Vamos a explorar en detalle qué es una clase, cómo crear objetos a partir de ella y cómo estos interactúan dentro del código.

Ejemplo: Imagina un carro. Los atributos como la marca, el modelo, el color y la velocidad máxima describen sus características. Estos atributos se definen dentro de una clase, y el constructor de esa clase se encarga de inicializarlos cuando creamos un nuevo objeto (es decir, un nuevo carro). Este proceso es clave para entender cómo funciona ***POO***.

Definición de Clase

Es una plantilla que describe qué atributos (datos) y métodos (acciones) tendrá el objeto que crearemos. La clase no es un objeto en sí misma, pero define cómo será el objeto cuando lo construyamos.

Cuando defines una clase, especificas qué datos debe almacenar (por ejemplo, la marca o el color de un carro) y qué operaciones puede realizar sobre esos datos (como acelerar o frenar). En otras palabras, una clase es una plantilla que indica cómo construir un objeto y qué puede hacer ese objeto.

En ***POO***, es común usar diagramas de clase para visualizar de manera clara la estructura y relaciones de las clases, lo que facilita su implementación en el lenguaje de programación.

```
La estructura de una clase es:
class [nombre de la clase] {
  [atributos o variables de la clase]
  [métodos o funciones de la clase]
  [main]
}
```

Para declarar una clase en ***Java***, usamos la palabra clave ***class***, seguida del nombre de la clase. El nombre de la clase debe ser un sustantivo que represente el concepto u objeto que queremos modelar.

public class Carro { }

A continuación, tenemos un ejemplo básico de la declaración de una clase.

```
// Definimos la clase Carro
public class Carro {
    // Atributos de la clase
    String marca;
    String modelo;
    String color;
    // Métodos o comportamientos de la clase se pueden agregar aquí
    // (como: acelerar, frenar, etc.)
}
```

Los atributos son variables que pertenecen a una clase y describen las características de los objetos que se crearán a partir de esa clase. Los atributos son también conocidos como propiedades o campos, y pueden ser de diferentes tipos de datos, como ***int***, ***double***, ***String***, entre otros.

Para declarar atributos en una clase, simplemente declaramos variables dentro de la clase pero fuera de cualquier método. Los atributos pueden ser públicos, privados, protegidos, etc., dependiendo del nivel de acceso que queramos dar.

Ejemplo más completo de la clase ***Carro*** con atributos:

```java
// Ejemplo 2: Declaración de atributos en Java

// Definimos la clase Carro con sus atributos
public class Carro {
    // Atributos
    String marca;
    String modelo;
    String color;


    // Método para mostrar detalles del carro
    public void mostrarDetalles() {
        System.out.println("Marca: " + marca);
        System.out.println("Modelo: " + modelo);
        System.out.println("Color: " + color);
    }
}
```

Niveles de Visibilidad

public

Indica que la clase es accesible desde cualquier otro lugar en el código, tanto dentro como fuera del paquete en el que se encuentra definida.

Declarar una clase pública facilita la reutilización de esta en diferentes partes del programa. Además, se pueden crear bibliotecas de clases públicas que pueden ser utilizadas por otros desarrolladores. Sin embargo, al permitir el acceso a la clase desde cualquier parte del código, también se pierde el control sobre quién y cómo se utiliza la clase. Esto puede llevar a problemas de seguridad y mantenimiento del código.

private

Indica que la clase sólo es accesible desde dentro de la misma clase. Esta palabra clave se utiliza para crear clases internas o clases de ayuda que no deben ser visibles desde fuera de la clase que las contiene.

La ventaja de limitar el acceso a la clase solo desde dentro de la misma es que se garantiza una mayor seguridad y privacidad del código. Además, permite la creación de clases de

ayuda que solo se utilizan dentro de la misma clase principal. No obstante, esta propiedad limita la reutilización de la clase en otras partes del programa, ya que no se puede acceder a ella desde fuera de la clase. También puede llevar a la creación de código duplicado si se necesita una funcionalidad similar en otras partes del programa.

protected

Indica que la clase es accesible desde cualquier clase en el mismo paquete y desde cualquier subclase, independientemente del paquete en el que se encuentren.

Esta propiedad permite el acceso a la clase desde cualquier clase en el mismo paquete y desde cualquier subclase, lo que facilita la creación de jerarquías de clases y la reutilización de código en diferentes partes del programa. Aun así, al admitir el acceso a la clase desde diferentes partes del programa, también se pierde un poco el control sobre quién y cómo se utiliza la clase. Además, limita la accesibilidad de la clase a otros paquetes, lo que puede dificultar la reutilización del código en diferentes proyectos.

default

Si no se especifica ninguna palabra clave antes de ***class***, la clase tiene un nivel de acceso predeterminado, también conocido como ***default***. Tomando las propiedades de una clase ***private*** y su comportamiento.

NIVELES DE VISIBILIDAD

MODIFICADOR	CLASE	PACKAGE	SUBCLASE	TODOS
Public	Si	Si	Si	Si
Protected	Si	Si	Si	NO
Default	Si	Si	NO	NO
Private	Si	NO	NO	NO

ÍNDICE

1.2.2 Autoevaluación del aprendizaje:

Relación de columnas que se enfoca exclusivamente en los conceptos de Clase y Atributo en *Java*.

Instrucciones: Relaciona el concepto con su descripción correspondiente y selecciona la respuesta que muestra la correspondencia correcta.

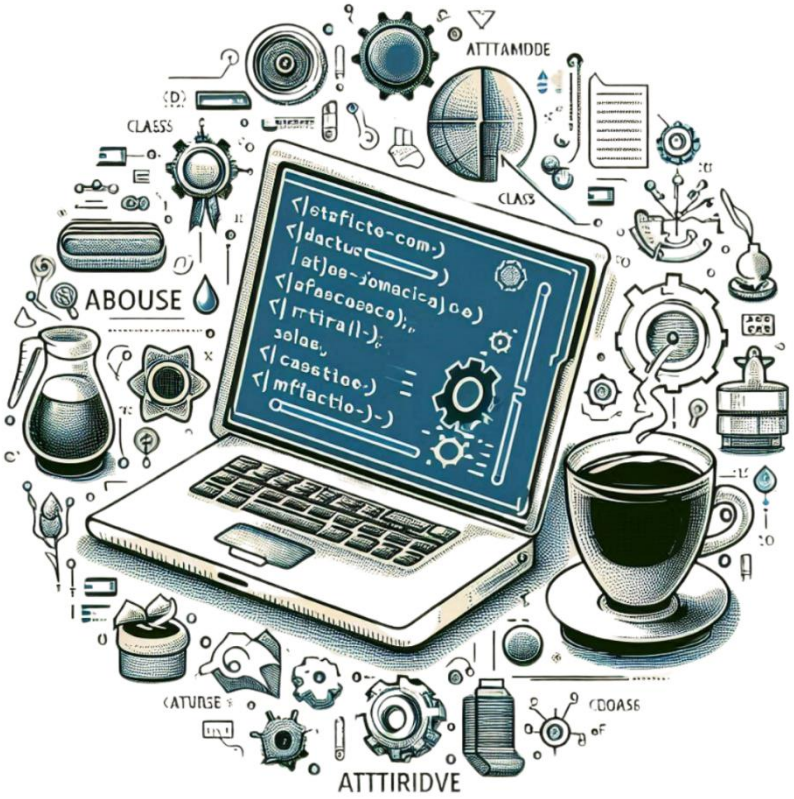
DESCRIPCIÓN	CONCEPTO
I. Plantilla o modelo que define la estructura y el comportamiento de los objetos.	a) Atributo
II. Característica o propiedad de un objeto.	
III. Representa los datos que almacenan el estado de un objeto.	
IV. Define las características que hacen que cada objeto sea único.	b) Clase
V. Es como un "plano" para crear objetos.	
VI. Variable que se declara dentro de una clase.	
VII. Los objetos se crean a partir de ella.	
VIII. También se conocen como variables de instancia o campos	

A) I:a - II:b - III:b - IV:a - V:b - VI:a - VII:b - VIII:a
B) I:b - II:a - III:a - IV:a - V:b - VI:a - VII:b - VIII:a
C) I:a - II:a - III:a - IV:b - V:a - VI:b - VII:b - VIII:a
D) I:b - II:a - III:b - IV:a - V:b - VI:a - VII:b - VIII:a

ÍNDICE

APRENDIZAJE 1.3

Describe los conceptos de métodos del lenguaje *Java*; conoce cómo instanciar objetos a partir de una Clase; e implementa programas utilizando métodos.



Temática:

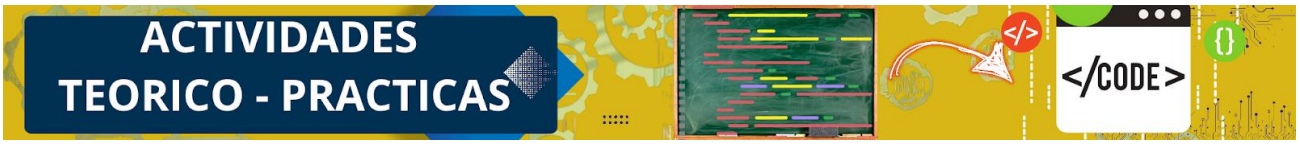
Métodos:

- Definición.
- Declaración.
- Parámetros.
- ***Getter***.
- ***Setter***.

Objetos:

- Definición.
- Declaración (instanciación).

Implementación.



ÍNDICE

1.3.1. Actividades de aprendizaje teórico–prácticas.

Método.

Es un conjunto de instrucciones cuyas características, son:

- Se define dentro de una clase.
- Tiene como objetivo realizar una tarea específica.
- Permiten dividir un programa en pequeñas funciones que pueden ser reutilizadas y ejecutadas cuando sea necesario, invocándolas por su nombre.

Cuando se llama a un método, el flujo de ejecución del programa se traslada a ese método, donde se ejecutan las instrucciones que contiene. Al llegar a la palabra clave ***return***, el método finaliza su ejecución y el control vuelve al punto del programa donde se realizó la llamada. Los métodos pueden devolver un valor (usando ***return***), o simplemente realizar una acción sin devolver nada (en ese caso, el tipo de retorno es ***void***).

Los métodos permiten crear código más modular y reutilizable, facilitando su mantenimiento y comprensión.

Las ventajas de emplear métodos son:

✓ Se construyen programas modulares.

✓ El código se reutiliza.

En el lenguaje de programación ***Java*** existen dos tipos principales de métodos:

- **Métodos de instancia**: Está siempre asociado a un objeto específico, lo que significa que, para poder utilizarlo, primero se debe crear una instancia (un objeto) de la clase. Estos métodos operan sobre los atributos del objeto y requieren que el objeto exista para ser invocados.

- **Métodos de clase** (también conocido como método estático): No depende de una instancia y puede ser invocado sin necesidad de crear un objeto. Para definir un método de clase, se utiliza la palabra clave ***static*** antes del tipo de retorno. Estos métodos suelen estar relacionados con la clase en general, más que con los datos específicos de un objeto.

Método *main()*.

En cualquier aplicación escrita en Java, debe existir el método ***main()***, cuyas características principales, son:

- Debe estar dentro de alguna de las clases.
- Es un método estático, lo que significa que no es necesario crear un objeto para ejecutarlo.
- Es el punto de inicio de la ejecución de un programa ***Java*** y también define el punto de salida de este.

Métodos (*getter* y *setter*).

En Java, un lenguaje orientado a objetos, las clases definen tanto los atributos (variables) como los métodos que operan sobre esos atributos. Por convención, los atributos de una clase suelen declararse como privados, en un proceso conocido como encapsulación, que tiene como objetivo proteger los datos de accesos y modificaciones no autorizadas. Para interactuar con estos atributos privados, se utilizan métodos especiales denominados getters y setters, los cuales se declaran como públicos y permiten controlar el acceso y modificación de los datos de manera segura.

Método *Getter*:

Un ***getter***, **también llamado método de acceso**, es un método que permite obtener el valor de una variable privada desde fuera de la clase. El nombre de un método ***getter*** comienza con la palabra ***get***, seguida del nombre de la variable cuyo valor se desea acceder, siguiendo la convención de utilizar el mismo nombre que la variable, pero con la primera letra en mayúscula. El ***getter*** debe tener un tipo de retorno que coincida con el tipo

de dato de la variable. Estos métodos aseguran un control preciso sobre el acceso a los datos, proporcionando el valor correcto sin exponer la variable directamente.

```
public String getColor() {
    return this.color;
}
```

Método *Setter*:

Un setter, **también conocido como método de modificación**, es un método que permite modificar el valor de una variable privada. Al igual que el ***getter***, el nombre de un ***setter*** comienza con la palabra set seguida del nombre de la variable. Este método recibe un parámetro que corresponde al nuevo valor que se desea asignar, y no tiene valor de retorno (debe ser ***void***). Los ***setters*** permiten controlar cómo y cuándo se modifica una variable privada, asegurando que se le asigne un valor válido y apropiado.

```
public void setColor(String nombre) {
    this.nombre = color;
}
```

La palabra clave ***this*** se utiliza para hacer referencia al objeto actual de la clase. En el caso de los métodos ***setter***, para referirse a la variable privada de la clase a la que se quiere asignar un valor.

Objetos.

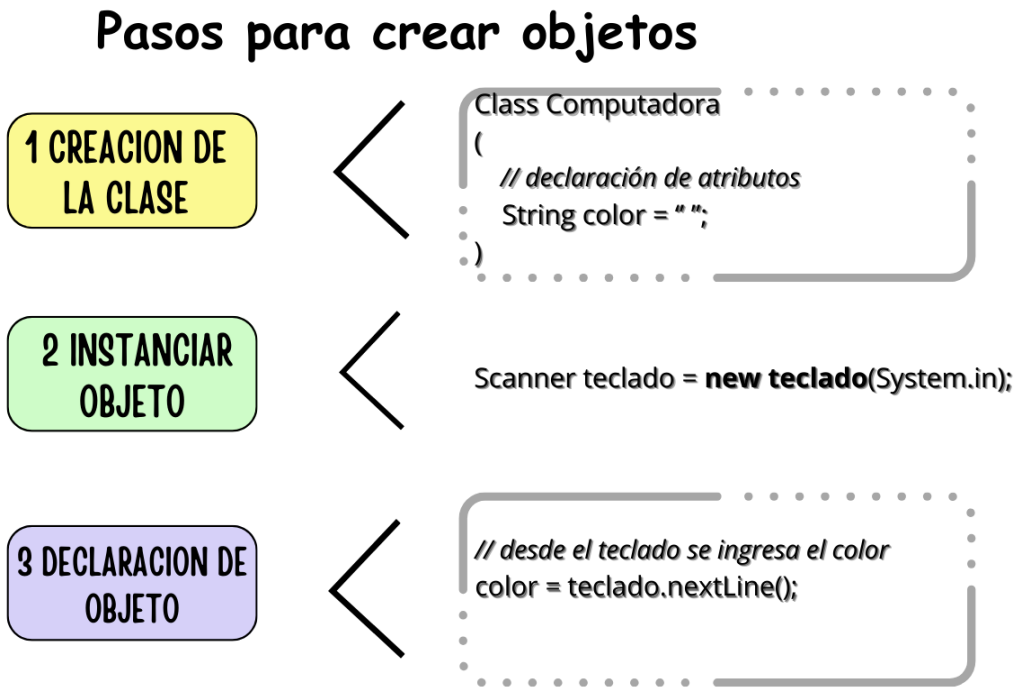
Es un componente autónomo creado a partir de una clase, es una estructura de datos existente en la memoria de la computadora que tiene atributos o datos almacenados por el objeto y operaciones disponibles (**métodos**).

Creación de un Objeto en Java.

Para crear un objeto en ***Java***, es fundamental:

- El nombre de la clase para la cual vamos a crear el objeto.
- El método constructor de dicha clase, que puede o no recibir parámetros.

El constructor tiene la función de inicializar los atributos de un objeto en el momento en que se crea, un proceso llamado instanciación.



La palabra clave ***new***, implica que se debe crear una nueva instancia de una clase específica.

ÍNDICE

1.3.2 Autoevaluación del aprendizaje:

5. Sopa de letras.

Instrucciones: encuentra las clases y métodos en la sopa de letras:

- ***Scanner***
- ***nextInt***
- ***nextDouble***
- ***nextLine***
- ***setvalor***
- ***getvalor***
- ***setter***
- ***getter***

N	Y	Á	Ú	É	S	G	S	Ü	X
E	X	Á	X	P	E	R	E	D	N
X	Ü	E	W	T	T	E	T	R	E
T	N	Ñ	T	F	T	N	V	O	X
D	L	E	J	Ú	E	N	A	L	T
O	R	A	X	T	R	A	L	A	L
U	J	Í	I	T	S	C	O	V	I
B	E	Ó	V	L	I	S	R	T	N
L	Ü	Í	W	Z	H	N	I	E	E
E	P	G	J	M	K	A	T	G	K

El programa crea un objeto Persona, le asigna el nombre "Ana" utilizando un método *setter*, y luego muestra el nombre en la consola utilizando un método *getter*. Este es un ejemplo básico de encapsulación en *Java*, donde los atributos privados se acceden y modifican a través de métodos públicos.

Instrucciones: Completa los espacios en blanco colocando el número del código correspondiente y selecciona la respuesta que muestra la correspondencia correcta.

1. ***persona1.setNombre("Ana");***	2. ***String getNombre()***	3. ***String nombre;***
4. ***persona1.getNombre();***	5. ***setNombre(String nombre)***	

a)
b)
c)
d)
e)

```
class Persona {
    private ________;

        public void ________ {
        this.nombre = nombre;
    }

        public ________ {
        return nombre;
    }
}

public class Main {
    public static void main(String[] args) {
        Persona persona1 = new Persona();

                ________

        System.out.println("El nombre es: " + ________
    }
}
```

Opciones de respuesta:

A) a:5 - b:3 - c:1 - d:4 - e:2

B) a:3 - b:5 - c:2 - d:1 - e:4

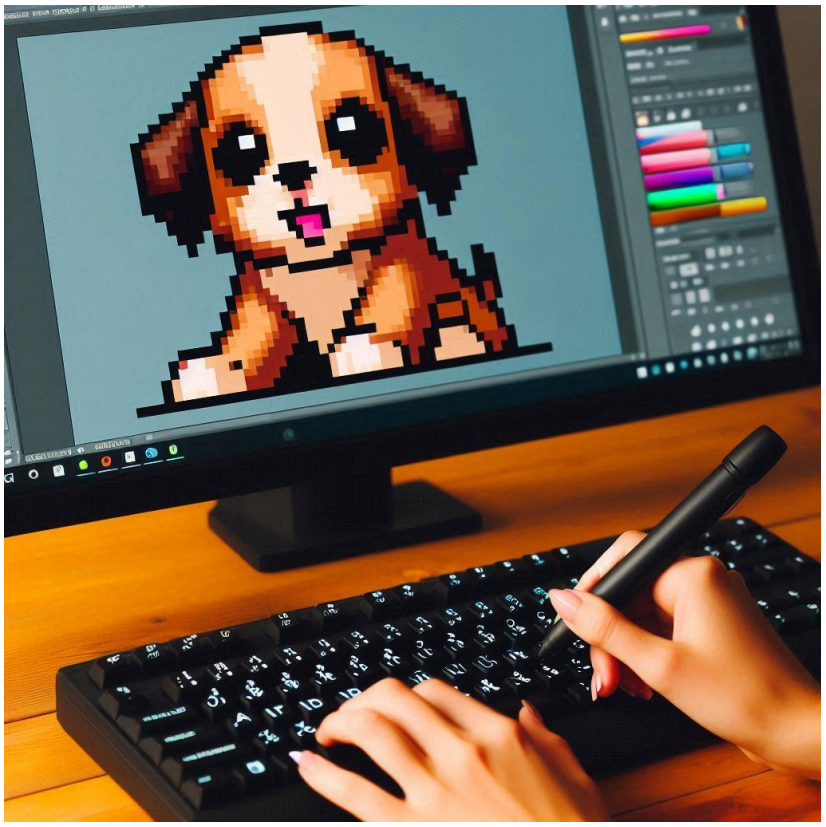
C) a:3 - b:4 - c:1 - d:2 - e:5

D) a:5 - b:3 - c:1 - d:2 - e:4

ÍNDICE

APRENDIZAJE 1.4

Empleará la Clase *Scanner* para la entrada de datos en la creación de un programa.



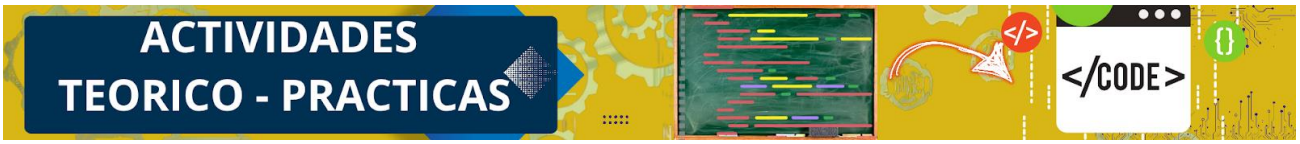
Temática:

La Clase *Scanner*:

- Importar la Clase ***java.util.Scanner***.
- Definición del objeto de la Clase ***Scanner*** (instanciación).
- Método ***system.in***
- Introducción de datos desde el teclado.

Errores sintácticos y lógicos.

Ejecución del programa.



ÍNDICE

1.4.1. Actividades de aprendizaje teórico–prácticas.

La Clase *Scanner*:

Permite leer valores de entrada de los siguientes tipos de datos (***int***, ***double***, ***float***, ***char*** y ***String***), y se encuentra localizada en el paquete ***java.util***; los valores de la entrada de datos pueden provenir de diversas fuentes, por ejemplo: teclado, archivo, etc. Para el caso de entrada por teclado, será necesario pasar como parámetro el objeto ***in*** en la clase ***System***.

Ejercicio: Crear una copia exacta de un objeto que ya se tiene…¿Cómo lo harías?

1° Debes crear un nuevo objeto de la misma clase.
2° Después debes recorrer todos los campos del objeto original; y
3° Copiar sus valores en el nuevo objeto.

Figura . Creación de clase.

Para definir (o instanciar) un objeto de la clase Scanner, se procede así:

```
Scanner nombre_del_objeto = new Scanner(System.in);
```

System.in representa los datos ingresados a través del teclado.

Podemos pasar información a esta clase a lo que se le conoce como argumento, para posteriormente poder utilizarla en nuestro código.

La clase ***Scanner*** proporciona una serie de métodos que permiten introducir diferentes tipos de datos, a continuación, se presentan los métodos más usuales de la clase ***Scanner***:

Método	Entrada tipo de dato
nextInt	Número tipo entero
nextFloat	Número tipo decimal
nextDouble	Número tipo decimal de doble precisión
nextcharAt(0)	Carácter
nextLine()	Cadena de caracteres

Ejemplo: Introduciendo datos desde el teclado:

El objetivo de los siguientes programas es mostrar la creación de clases y uso de los métodos ***getter*** y ***setter***, que son necesarios para la correcta operación del programa, creación del método ***main***, instanciar objetos e invocar métodos.

Pasos para tu primer Código

1° **Identificar el nombre y los atributos de la clase:**
El nombre de la clase debe ser descriptivo y reflejar el propósito de esta. Los atributos son las variables que conformarán el estado interno de los objetos creados a partir de la clase. Por ejemplo, si queremos crear una clase para representar un coche, podríamos definir los atributos «marca», «modelo», «año», «color», «precio», etc.

2° **Crear la clase:**

En ***Java***, las clases se definen con la palabra clave ***class***, seguida del nombre de la clase y las llaves para encerrar en un solo bloque.

```
// Importar la clase Scanner para la entrada de datos
import java.util.Scanner;

// Definición de la clase Carro
public class Carro {

}
```

3° Definir los atributos:

Dentro de la clase, se deben declarar los atributos como variables. Es recomendable definirlos como ***private*** para asegurar el encapsulamiento, y también se deben proporcionar los métodos ***getter*** (métodos de acceso) y ***setter*** (métodos de asignación) para acceder y modificar los valores de los atributos desde fuera de la clase.

```
// Atributos
public String marca;
public String modelo;
public int anio;
public String color;
```

4° Definir los métodos:

Los métodos son las acciones que puede realizar un objeto creado a partir de la clase. Es recomendable definirlos como ***public*** para permitir su acceso desde fuera de la clase

```
// Método para mostrar información del carro
public void mostrarInformacion() {
    System.out.println("Marca: " + getMarca());
    System.out.println("Modelo: " + getModelo());
    System.out.println("Año: " + getAnio());
    System.out.println("Color: " + getColor());
}
```

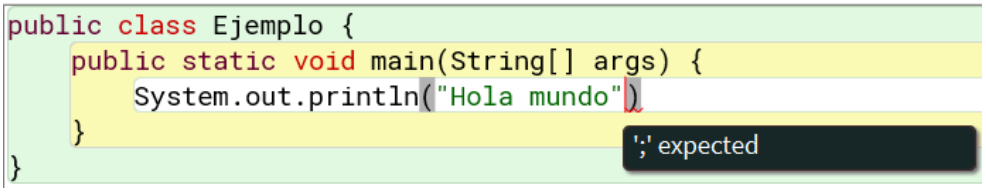
Puedes revisar el código anterior en el siguiente enlace:

https://onlinegdb.com/fg3qPQVS4

En resumen, las instancias son una parte fundamental de ***POO***, ya que permiten la creación de objetos concretos a partir de una clase y su posterior manipulación. Las instancias están relacionadas con el encapsulamiento y el polimorfismo, dos de los cuatro conceptos fundamentales de ***POO***.

TIPOS DE ERRORES

- **Sintácticos:** Ocurren cuando el programador escribe código que no va de acuerdo con las reglas de escritura del lenguaje de programación, por ejemplo:
  - Uso incorrecto de paréntesis, corchetes o llaves.
  - Olvidar cerrar comillas en cadenas de texto.
  - Declarar una variable con un nombre inválido o incorrecto.
  - No seguir la estructura adecuada para la definición de clases, métodos o bloques de código.



- **Lógicos:** Ocurren a causa de un mal diseño del programa. Puede ocurrir que una línea de código observe todas las reglas sintácticas del lenguaje, pero el código tenga una lógica equivocada, por ejemplo:
  - Condicionales incorrectamente formulados que producen resultados inesperados.
  - Algoritmos incorrectamente implementados que generan resultados incorrectos.
  - Acceso incorrecto a variables, arreglos o estructuras de datos que causa un comportamiento no deseado.
  - Falta de consideración de casos especiales en la lógica del programa.

En la figura siguiente, se muestran un resultado inesperado, debido a un error lógico:

```java
import java.util.Scanner;

public class CalculoRaizCuadrada {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Ingrese un número para calcular su raíz cuadrada: ");
        double numero = scanner.nextDouble();

        // Intentar calcular la raíz cuadrada
        double raizCuadrada = Math.sqrt(numero); // Esto generará un error si el número es negativo

        System.out.println("La raíz cuadrada de " + numero + " es: " + raizCuadrada);

        // Cerrar el escáner
        scanner.close();
    }
}
```

En la ejecución del programa, se visualizaría:

```
Ingrese un número para calcular su raíz cuadrada:
```

Si el Usuario ingresa: ***-4***

Entonces, se tendrá el siguiente:

Resultado

La raíz cuadrada de -4.0 es: NaN

ÍNDICE

1.4.2 Autoevaluación del aprendizaje:

Relación de columnas.

Instrucciones: En la siguiente tabla, se presentan la clase ***Scanner*** y los métodos ***setter*** y ***getter***, utilizados en la programación en Java. Relaciónalos con su descripción correcta y selecciona la respuesta que muestra la correspondencia correcta:

MÉTODO/CLASE	DESCRIPCIÓN
I. ***Scanner***	a) Método que permite obtener el valor de un atributo de una clase.
II. ***nextInt***	b) Lee un párrafo completo de texto ingresado por el usuario.
III. ***nextDouble***	c) Método setter que permite asignar un valor a un atributo de una clase.
IV. ***nextLine***	d) Lee un número entero ingresado por el usuario.
V. ***setValor***	e) Lee un número decimal ingresado por el usuario.
VI. ***getValor***	f) Clase que permite la lectura de datos desde diferentes fuentes, como el teclado o archivos.
	g) Lee una línea completa de texto ingresada por el usuario.

A) I:f, II. d, III. e, IV. g, V. c, VI. a

B) I:f - II:e -III:a - IV:b - V:c - VI:d

C) I:f - II:e -III:d - IV:c - V:b - VI:a

D) I:a - II:c -III:e - IV:g - V:b - VI:d

Programa que lee tu nombre y tu edad.

Instrucciones: Completa los espacios en blanco colocando el número de la palabra reservada correspondiente y selecciona la respuesta que muestra la correspondencia correcta:

1.- ***nextLine***	2.- ***nextInt***	3.- ***System.in***	4.- ***nextDouble***	5.- ***Scanner***

a)
```
import java.util._____;
```

b)
```
public class TecleaNombre
{
    public static void main (String[] args)
    {
        Scanner teclado = new Scanner(_____);

        String nombre;
        int edad;

        System.out.println("Teclea tu nombre: ");
        nombre = teclado._____();
        System.out.println("Ingresa tu edad: ");
        edad = teclado._____();

        System.out.println("Hola " + nombre + ", tu edad es: " + edad);
    }
}
```

c)

d)

Opciones de respuesta:

A) a:2 - b:1 - c:3 - d:5

B) a:4 - b:1 - c:2 - d:3

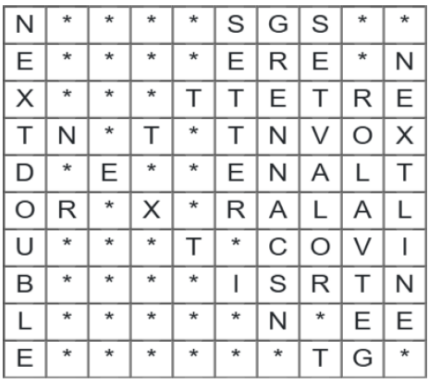
C) a:5 - b:3 - c:1 - d:2

D) a:4 - b:5 - c:2 - d:1

ÍNDICE

1.5 RESPUESTAS A LAS AUTOEVALUACIONES.

AUTOEVALUACIÓN	RESPUESTA CORRECTA
1.1.2	Sopa de letras: * S * * * O * * * * * * * * * * * I * * * * * * * * O B * * T * N * * * * * * * * * D * * * * * * * * * T L * * N * * Ó * * * * * * E * * * * * P * * * S U O * * E * * * I * * * * N * * * * * A * * * O * B Q * * I * * * * C * * T * * * * * L * * * V * * I U * * M * * * * * C I * * * * * A * * * I * * * R E * * A * * * * * D A * * * * B * * * T * * * * T D * * T * * * * A * * R * * R * * * I * * * * * A E * * R * * * D * * * * T A * * * M * * * * * * * C * * O * * * * * * * * S S * * I * * * * * * * * Ó S * P * * * * * * * R * * B R * * * * * * * * * D E * M * * * * * * E * * * P A * * * * * * * * * I N * O * * * * * S * * * S * * * * * * * * * * G O * C * * * * E * * * O * * * * * * * * * * * O I * * * * * R * * * T * * * * * * * * * * * * S * * * * V * * * A * * S O I R A T N E M O C * * E * * * A * * * D * * * * * * * * * * * * * * R * * D * * * E * * * * * * * * * * * * * * * P * A * * * D * * * * * * S E R O D A R E P O * * X S * * * S * * * * * * * * * * * * * * * * * E * * * O * * S E R O D A C I F I T N E D I * * * * * * P * * * * * * * S A I C N E T N E S * * * * * * I * * * U S O D E B I B L I O T E C A S * * * * T * * * * E N C A P S U L A M I E N T O * * * * *
	Relación de columnas: D) I:f - II:d - III:g - IV:c - V:b - VI:h - VII:e - VIII:a - IX:j
	Programa que identifica el mayor de tres números: A) a:5 - b:2 - c:6 - d:3 - e:1 - f:10 - g:9 - h:4 - i:7
1.2.2	Relacionar columnas DESCRIPCIÓN – CONCEPTO: C) I:b - II:a - III:a - IV:a - V:b - VI:a - VII:b - VIII:a

<table>
<tr><td rowspan="2">1.3.2</td><td>Sopa de letras – Clases y Métodos:
 
<table>
<tr><td>N</td><td>*</td><td>*</td><td>*</td><td>*</td><td>S</td><td>G</td><td>S</td><td>*</td><td>*</td></tr>
<tr><td>E</td><td>*</td><td>*</td><td>*</td><td>*</td><td>E</td><td>R</td><td>E</td><td>*</td><td>N</td></tr>
<tr><td>X</td><td>*</td><td>*</td><td>*</td><td>T</td><td>T</td><td>E</td><td>T</td><td>R</td><td>E</td></tr>
<tr><td>T</td><td>N</td><td>*</td><td>T</td><td>*</td><td>T</td><td>N</td><td>V</td><td>O</td><td>X</td></tr>
<tr><td>D</td><td>*</td><td>E</td><td>*</td><td>*</td><td>E</td><td>N</td><td>A</td><td>L</td><td>T</td></tr>
<tr><td>O</td><td>R</td><td>*</td><td>X</td><td>*</td><td>R</td><td>A</td><td>L</td><td>A</td><td>L</td></tr>
<tr><td>U</td><td>*</td><td>*</td><td>*</td><td>T</td><td>*</td><td>C</td><td>O</td><td>V</td><td>I</td></tr>
<tr><td>B</td><td>*</td><td>*</td><td>*</td><td>*</td><td>I</td><td>S</td><td>R</td><td>T</td><td>N</td></tr>
<tr><td>L</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>N</td><td>*</td><td>E</td><td>E</td></tr>
<tr><td>E</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>T</td><td>G</td><td>*</td></tr>
</table></td></tr>
<tr><td>Programa que crea un objeto Persona:
B) a:3 - b:5 - c:2 - d:1 - e:4</td></tr>
<tr><td>1.4.2</td><td>Relacionar columnas Scanner – Método/Clase:
A) I:f - II:d -III:e - IV:g - V:c - VI:a</td></tr>
<tr><td></td><td>Programa que lee nombre y edad:
C) a:5 - b:3 - c:1 - d:2</td></tr>
</table>

ÍNDICE

1.6 REFERENCIAS.

Java P. (2025). *Java Platform Standard Edition 8 Documentation*. España. Recuperado de https://docs.oracle.com/javase/8/docs

ÍNDICE

Propósito.

Al finalizar la unidad, cualquier estudiante utilizará las estructuras de control de secuencia para la resolución de problemas a través del lenguaje de programación orientado a objetos con ***Java***.

ÍNDICE

2.0 Conceptos Clave.

Ciclo o bucle.

Es una secuencia de instrucciones de código que se ejecuta repetidas veces, hasta que una condición se cumpla.

Sentencia simple.

Es un conjunto de sentencias encerradas entre llaves, que se utiliza para especificar un flujo secuencial.

Sentencia compuesta.

Es un conjunto de varias instrucciones o acciones que se agrupan y se ejecutan juntas.

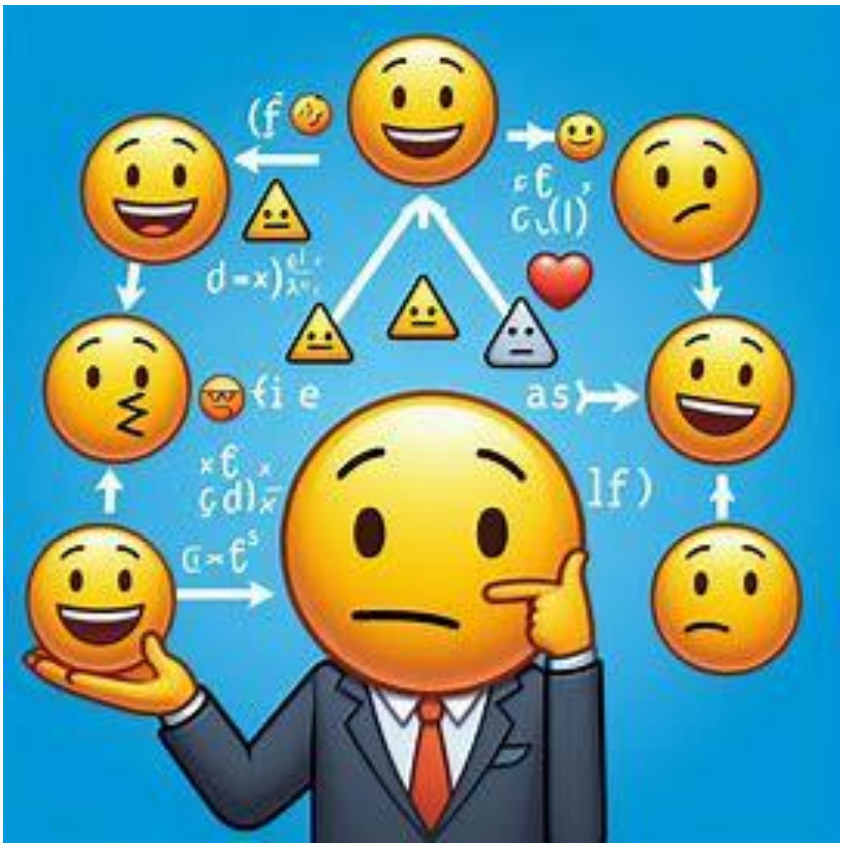
Estructuras de control.

Son bloques de código en programación que permiten tomar decisiones, repetir acciones o ejecutar instrucciones de forma controlada. Se organizan en tres tipos: secuencial, condicional y repetición.

ÍNDICE

APRENDIZAJE 2.1

Desarrolla programas que involucren las estructuras condicionales simples, compuestas y anidadas en los métodos de una Clase.

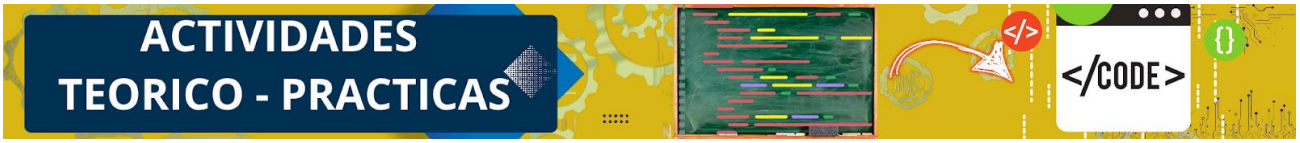


Temática:

Estructuras condicionales:

- Simples: ***if***.
- Compuestas: ***if-else***
  - Operadores:
    - Matemáticos.
    - Relacionales.
    - Lógicos.
  - Anidadas.

Implementación.



ÍNDICE

2.1.1. Actividades de aprendizaje teórico–prácticas.

Un programa escrito de modo secuencial ejecuta una sentencia después de otra, comienza con la primera y prosigue hasta la última, cada una se ejecuta una sola vez, el modo secuencial es adecuado para resolver problemas sencillos. Sin embargo, para solucionar problemas de tipo general, se necesita la capacidad de controlar cuáles son las sentencias para ejecutar en cada momento. Las estructuras de control determinan la secuencia de ejecución de las sentencias, se dividen en tres categorías en función de la secuencia de ejecución: ***secuencial***, ***condicional*** y ***repetición***. Este capítulo considera las sentencias ***if, if- else*** y ***switch***, estructuras condicionales que controlan si una sentencia o lista de sentencias se cumplen.

Estructura condicional Simple *if.*

En ***Java***, la estructura de control de selección principal es una sentencia ***if***; la cual tiene la **sintaxis** siguiente:

```
if (expresión) Acción
```

La condicional simple ***if*** funciona de la siguiente manera: se evalúa la expresión entre paréntesis, si la expresión es verdadera se ejecuta la acción, en caso contrario no se efectúa y sigue la ejecución en la siguiente sentencia; tal como se muestra en la ***Figura 1.***

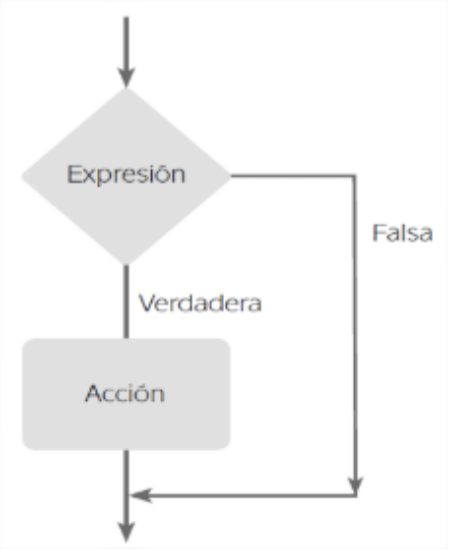


Figura 1.
Diagrama de flujo de una condicional simple ***if****.*

A partir de aquí se realizan una serie de ejemplos o actividades, en el lenguaje de programación ***Java***.

Ejemplo 1. Prueba de divisibilidad, este es un programa en el que se introducen dos números enteros y mediante una sentencia de selección se determina si son divisibles.

a. Insertar el siguiente código.

Clase principal:

```java
import java.util.Scanner;
//Creamos la clase Principal
public class Principal{
    public static void main(String args[]){
        //Declaramos las variables
        int num1, num2;
        //Creamos los 2 objetos teclado y objeto
        Scanner teclado =  new Scanner(System.in);
        Divisible objeto = new Divisible();
        //Mandamos el mensaje y guardamos el valor
        System.out.println("Ingresa primer entero");
        num1 = teclado.nextInt();
        objeto.setNum1(num1);
        System.out.println("Ingresa segundo entero");
        num2 = teclado.nextInt();
        objeto.setNum2(num2);
        //Mandamos a llamar el método Divisibles de la clase
        //Divisible
        objeto.Divisibles();
    }
}
```

Figura 2. *Código clase Principal.*

Clase Divisible:

```java
//Creamos la clase Divisible
public class Divisible{
    //Declaramos los atributos
    private int num1;
    private int num2;
    //Creamos el constructor
    public Divisible(){
        this.num1= 0;
        this.num2=0;

    }
    //Métodos set y get
    public void setNum1(int num1){
        this.num1=num1;
    }
    public int getNum1(){
        return num1;
    }
    public void setNum2(int num2){
        this.num2=num2;
    }
    public int getNum2(){
        return num2;
    }
    //Método if simple
    public void Divisibles(){
        //int n, d;
        if (num1 % num2 == 0){
            System.out.println( num1 +" es divisible por "+ num2);
        }
    }
}
```

Figura 3. *Código clase Divisible.*

b. Compilar y ejecutar el programa (Realizar pruebas).

```
Ingresa primer entero
10
Ingresa segundo entero
5
10 es divisible por 5
```

Figura 4. *Pruebas de ejecución del programa.*

Este programa lee dos números enteros y comprueba cuál es el valor del residuo de la división ***num1*** entre ***num2*** (***num1%num2***); si el residuo (***%***) es cero, ***num1*** es divisible entre ***num2***; en este caso **10** es divisible entre **5** y el residuo es **0**.

En el siguiente ejemplo se ingresan los valores de **7** y **5**, pero no se muestra, si **7** es divisible entre **5**. Esto se debe a que solamente estamos utilizando una ***condicional if simple***, el programa no regresa nada ya que solamente este tipo de condicionales simples muestran si cumple y de no ser así, hasta ahí termina el programa y por tal motivo no muestra nada.

```
Ingresa primer entero
7
Ingresa segundo entero
5
```

Figura 5. *Pruebas de ejecución del programa.*

Condicional compuesta *if – else*.

Un segundo formato de ***if*** es ***if-else***, cuya forma tiene la siguiente

sintaxis:

```
If (expresión)
      accion1
else
      accion2
```

En este formato ***accion1*** y ***accion2*** son, de forma individual, una única sentencia que termina con "punto y como" [***;***], o un grupo de sentencias entre llaves; la expresión se evalúa cuando se ejecuta la sentencia: si es verdadera, se efectúa ***accion1***; en caso contrario se ejecuta ***accion2***, la ***Figura 6*** muestra el diagrama de flujo, que indica la secuencia de ejecución del programa.

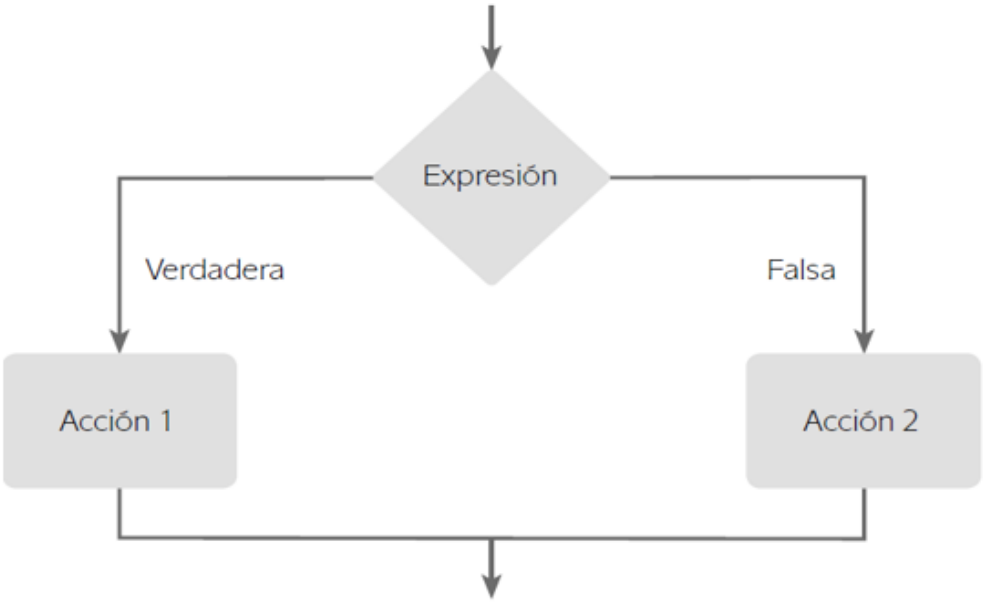


Figura 6. Diagrama de flujo de una condicional ***if-else***.

Ejemplo 2. Prueba de divisibilidad; en el programa ***Ejemplo 1***, se añadió la cláusula **`else`**; se leen dos números enteros y con el operador residuo (**%**) se comprueba si son divisibles o no.

Respuesta:

a. Escribir el siguiente código.

Clase principal:

```
import java.util.Scanner;
//Creamos la clase Principal
public class Principal{
    public static void main(String args[]){
        //Declaramos las variables
        int num1, num2;
        //Creamos los 2 objetos teclado y objeto
        Scanner teclado =  new Scanner(System.in);
        Divisible2 objeto = new Divisible2();
        //Mandamos el mensaje  y guardamos el valor
        System.out.print("Introduzca primer valor ");
        num1 = teclado.nextInt();
        objeto.setNum1(num1);
        System.out.print("Introduzca segundo valor ");
        num2 = teclado.nextInt();
        objeto.setNum2(num2);
        //Mandamos a llamar el método Divisibles de la clase
        //Divisible2
        objeto.Divisibles();
    }
}
```

Figura 7. *Código clase Principal.*

Clase Divisible2:

```
//Creamos la clase Divisible2
public class Divisible2{
    //Declaramos los atributos
    private int num1;
    private int num2;
    //Creamos el constructor
    public Divisible2(){
        this.num1= 0;
        this.num2=0;

    }
    //Métodos set y get
    public void setNum1(int num1){
        this.num1=num1;
    }
    public int getNum1(){
        return num1;
    }
    public void setNum2(int num2){
        this.num2=num2;
    }
    public int getNum2(){
        return num2;
    }
    //Método if simple
    public void Divisibles(){
        //Realizamos la comparación de num1 con num2;
        if (num1 % num2 == 0){
            System.out.println( num1 +" es divisible entre "+ num2);
        }else{
            System.out.println(num1 +" no es divisible entre "+num2);
        }
    }
}
```

Figura 8. *Código clase Divisible2.*

b. Compilar y ejecutar el programa (Realizar pruebas).

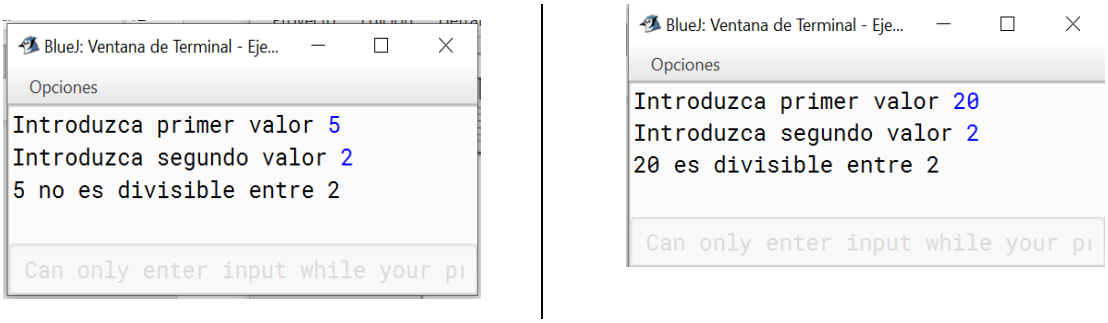


Figura 9. *Pruebas de ejecución del programa.*

Este programa realiza la comparación de dos números reales, si se cumple (***if***) que son divisibles, manda el mensaje ***es divisible entre***, sino se cumple (***else***), manda el mensaje ***no es divisible entre***.

Condicional *if-else* anidadas.

Hasta este momento, las sentencias ***if*** implementan decisiones que implican una o dos opciones, pero en esta sección, se mostrará que una sentencia ***if*** es anidada cuando alguna de las condiciones sea verdadera o falsa, también ***if***, se puede utilizar para tomar decisiones de otras opciones, tal como se muestra en la figura siguiente:

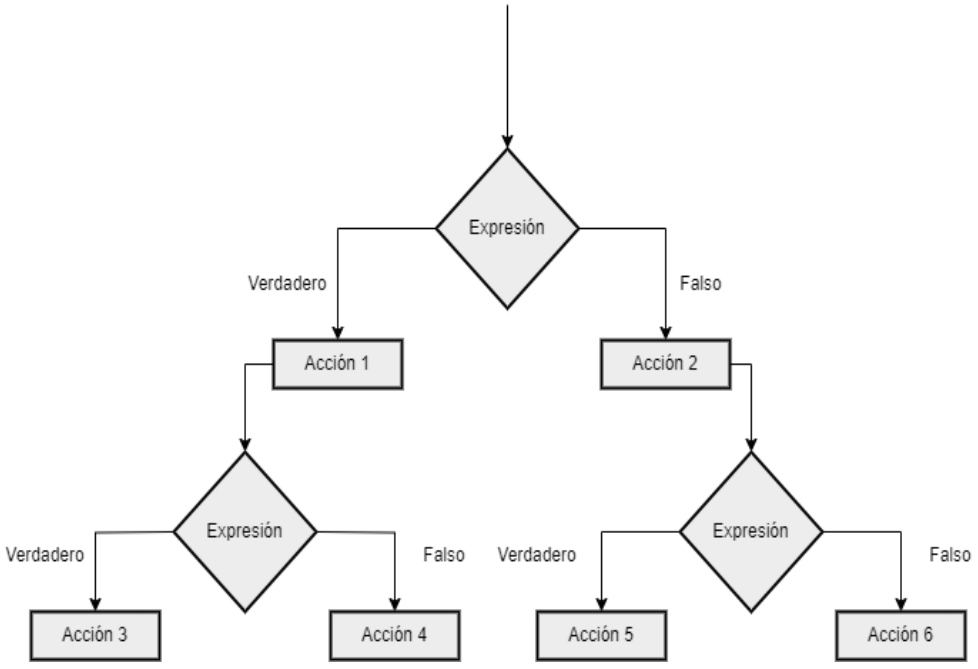


Figura 10. *Diagrama de flujo de una condicional anidada* ***if-else****.*

Sintaxis

```
if (expresión_lógica)
   Sentencia 1
else if (expresión_lógica2)
        Sentencia 2
     else if (expresión_lógica3)
             sentencia 3
          else
             sentencia 4;
```

Ejemplo 3. Comparación de un valor entero leído desde el teclado, muestra las condicionales anidadas ***if-else***.

a) Escribir el siguiente código.

Clase Principal:

```
import java.util.Scanner;
//Creamos la clase Principal
public class Principal{
    public static void main(String args[]){
        //Declaramos las variables
        int numero;
        //Creamos los 2 objetos teclado y objeto
        Scanner teclado =  new Scanner(System.in);
        PruebaCompuesta objeto = new PruebaCompuesta();
        //Mandamos el mensaje  y guardamos el valor
        System.out.println("Introduzca un número real");
        numero = teclado.nextInt();
        objeto.setNumero(numero);
        //Mandamos a llamar el método Compara de la clase
        //PruebaCompuesta
        objeto.Compara();
    }
}
```

Figura 11. *Código clase Principal.*

Clase PruebaCompuesta:

```java
//Creamos la clase PruebaCompuesta
public class PruebaCompuesta{
    //Declaramos los atributos
    private int numero;
    //Creamos el constructor
    public PruebaCompuesta(){
        this.numero= 0;
    }
    //Métodos set y get
    public void setNumero(int numero){
        this.numero=numero;
    }
    public int getNumero(){
        return numero;
    }
    //Método if simple
    public void Compara(){
        // Comparar número a cero
        if (numero > 0)
        {
            System.out.println(numero + " es mayor que cero");
            System.out.println
            ("Pruebe de nuevo introduciendo un número negativo");
        }
        else if (numero < 0)
        {
            System.out.println(numero + " es menor que cero");
            System.out.println
            ("Pruebe de nuevo introduciendo un número positivo");
        }
        else
        {
            System.out.println(numero + " es igual a cero");
            System.out.println
            ("¿Por qué no introduce un número negativo?");
        }
    }
}
```

Figura 12. *Código clase PruebaCompuesta.*

El ejemplo anterior realiza la comparación, si el número ingresado por el usuario ***es menor que cero (número < 0)***, ***mayor que cero (numero > 0)*** o ***igual a cero (numero = 0)***.

b) Compilar y ejecutar el programa (Realizar pruebas).

```
Introduzca un valor entero: 4
4 es mayor que cero
Pruebe de nuevo introduciendo un número negativo
Introduzca un valor entero: -2
-2 es menor que cero
Pruebe de nuevo introduciendo un número positivo
Introduzca un valor entero: 0
0 es igual a cero
¿Por qué no introduce un número negativo?
```

Figura 13. *Pruebas de ejecución del programa.*

ÍNDICE

2.1.2 Autoevaluación del aprendizaje:

Ejercicio 1. Programa que identifica el mayor de tres números.

Instrucciones: Completa los espacios en blanco colocando el número las estructuras condicionales simples, compuestas y anidada; selecciona la respuesta que muestra la correspondencia correcta.

1. ***else***	2. ***else if***	3. ***if***	4. ***}***	5. ***{***

```
import java.util.Scanner;

public class MayorDeTresNumeros {
    public static void main(String[] args) {
        Scanner teclado = new Scanner (System.in);

        // Solicitar los tres números al usuario
        System.out.print("Ingrese el primer número: ");
        int num1 = teclado.nextInt();

        System.out.print("Ingrese el segundo número: ");
        int num2 = teclado.nextInt();

        System.out.print("Ingrese el tercer número: ");
        int num3 = teclado.nextInt();

        // Determinar el número mayor
        int mayor;

a)      ________ (num1 >= num2 && num1 >= num3) {
            mayor = num1;
b)      } ________ if (num2 >= num1 && num2 >= num3) {
            mayor = num2;
c)      } ________ {
            mayor = num3;
        }

        // Mostrar el resultado
        System.out.println("El número mayor es: " + mayor);

        teclado.close();
    }
}
```

Opciones de respuesta:

A) a:3 - b:1 - c:3
B) a:5 - b:3 - c:1
C) a:3 - b:2 - c:1
D) a:5 - b:1 - c:3

Ejercicio 2. Programa que determina si eres mayor o menor de edad.

Instrucciones: Completa los espacios en blanco colocando el número las estructuras condicionales simples y/o compuestas correspondientes y selecciona la respuesta que muestra la correspondencia correcta.

1. ***else***	2. ***if***	3. ***}***	4. ***{***	5. ***else if***

```
import java.util.Scanner;  //llamado de la libreria Scanner
public class MenorMayor {
    public static void main (String [] args)
    {
        // declaración de variables
        int edad;

        // llamado al metodo Scanner para lectura desde teclado
        Scanner teclado = new Scanner (System.in);

        //solicitud e ingreso de datos
        System.out.print("Ingresa tu edad: ");
        edad = teclado.nextInt();

        //estructura condicional if simple
a)      _________ (edad >= 18) //operador relacional mayor que
b)      _________
        System.out.print("Eres mayor de edad");
c)      } _________ {
            System.out.print("Eres menor de edad");}
d)      _________
    }
```

Opciones de respuesta:

A) a:3 - b:1 - c:3 - d:4

B) a:4 - b:3 - c:1 - d:2

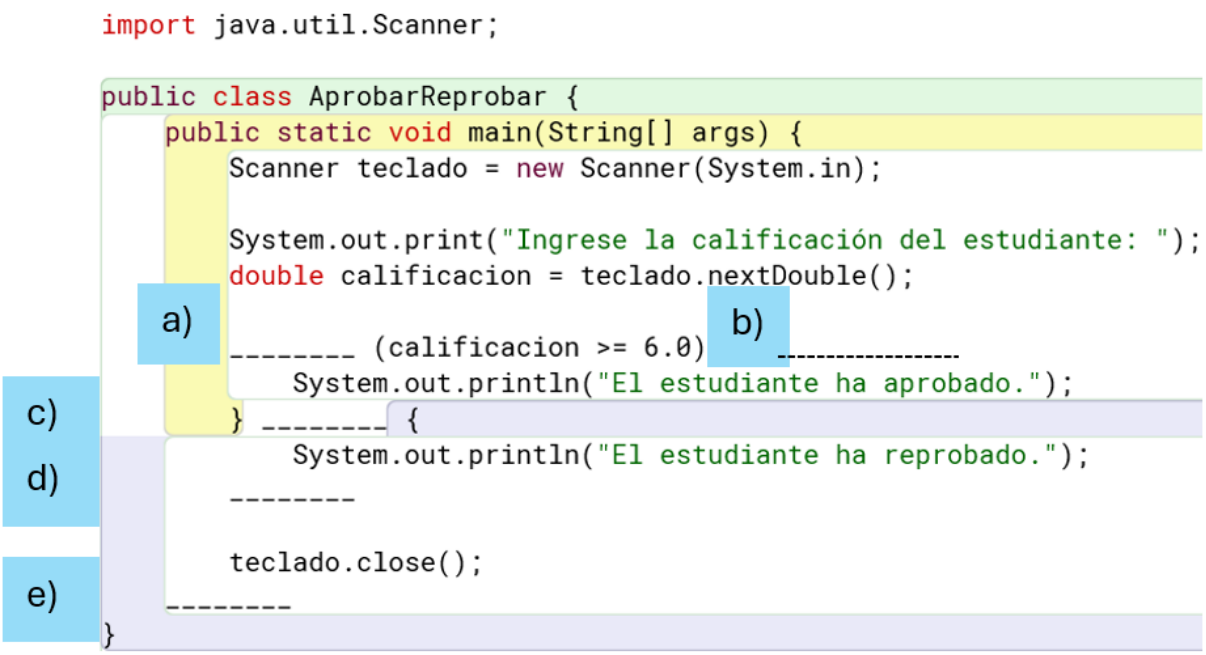
C) a:3 - b:2 - c:1 - d:4

D) a:2 - b:4 - c:1 - d:3

Ejercicio 3. Programa que determina si un estudiante aprueba o reprueba de acuerdo a una calificación ingresada por teclado.

Instrucciones: Completa los espacios en blanco colocando el número las estructuras condicionales simples y/o compuestas correspondientes; selecciona la respuesta que muestra la correspondencia correcta.

1. ***{***	2. ***if***	3. ***}***	4. ***else***	5. ***if else***	6. ***else if***



Opciones de respuesta:

A) a:3 - b:1 - c:3 - d:4 - e:2

B) a:2 - b:1 - c:4 - d:3 - e:3

C) a:3 - b:2 - c:1 - d:4 - e:3

D) a:2 - b:4 - c:1 - d:3 - e:3

ÍNDICE

APRENDIZAJE 2.2

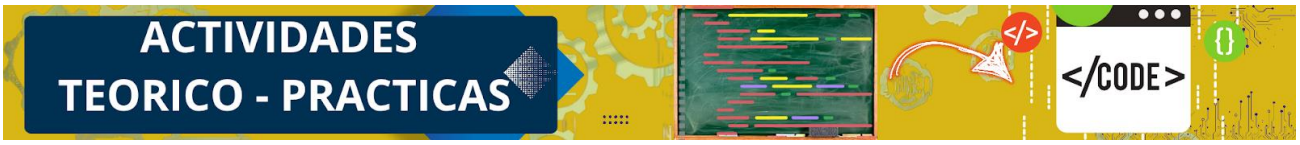
Desarrolla programas que involucren la estructura condicional múltiple en los métodos de una Clase.

Temática:

Estructura condicional múltiple:

- ***Switch***.

Implementación.



ÍNDICE

2.2.1. Actividades de aprendizaje teórico–prácticas.

En ***Java***, ***switch*** es una sentencia que se utiliza para elegir una de entre ***múltiples opciones***, es especialmente útil cuando la selección se basa en el valor de una variable simple o de una expresión simple denominada **expresión de control o selector**; el valor de dicha expresión puede ser solo de tipo ***int***, ***char o String***.

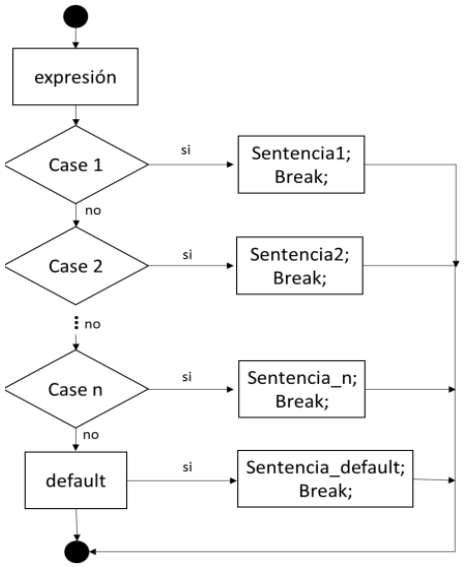


Figura 14. Diagrama de flujo de la sentencia ***switch***.

Sintaxis de la sentencia ***switch***.

```
switch (selector)
{
      case opcion_1: sentencias_1;
            break;
      case opcion_2: sentencias_2;
            break;
                  .
                  .
      case opcion_n: sentencias_n;
            break;
      default: sentencias_d
//opcional
}
```

La expresión de control o selector se evalúa y compara con cada una de las opciones del case, cada opción es única y diferente. Si el valor de la expresión de control es igual a una de las opciones del case, por ejemplo, ***opcion_1***, entonces la ejecución del bloque de sentencias comenzará *y* continuará hasta encontrar break o el final del switch. El tipo de dato de cada valor a evaluar y del selector debe ser el mismo.

La función de la palabra ***default*** consiste en ejecutar su código en caso de que el valor del selector no coincida con alguna de las opciones listadas en el ***case***.

Sentencia *switch*

Ejemplo 4. Considerando un rango entre 1 y 10 para asignar la nota de un curso, el programa ilustra la selección múltiple con la sentencia switch, tomar en cuenta la siguiente información:

El usuario debe de ingresar un número entre el 1 y el 10.

- Del 1 al 5, se deberá mandar el mensaje de “No acredito”.
- Del 6 y 7, se deberá mandar el mensaje de “Acredito”.
- Del 8, se deberá mandar el mensaje de “Notable”.
- Del 9 y 10, se deberá mandar el mensaje de “Excelente”.

Respuesta:

a) **Escribir el siguiente código.**

Clase Principal:

```
import java.util.Scanner;
//Creamos la clase Principal
public class Principal{
    public static void main(String args[]){
        //Declaramos las variables
        int nota;
        //Creamos los 2 objetos teclado y objeto
        Scanner teclado =  new Scanner(System.in);
        AsignaCalificacion objeto = new AsignaCalificacion();
        //Mandamos el mensaje  y guardamos el valor
        System.out.print("Introduzca la calificación (1 - 10), pulse enter: ");
        nota = teclado.nextInt();
        objeto.setNota(nota);
        //Mandamos a llamar el método Calificacion de la clase AsignaCalificacion
        objeto.Calificacion();
    }
}
```

Figura 15. *Código clase Principal.*

Clase AsignaCalificacion:

```java
//Creamos la clase AsignaCalificacion
public class AsignaCalificacion{
    //Declaramos los atributos
    private int nota;
    //Creamos el constructor
    public AsignaCalificacion(){
        this.nota= 0;
    }
    //Métodos set y get
    public void setNota(int nota){
        this.nota=nota;
    }

    public int getNota(){
        return nota;
    }
    //Método Calificación
    public void Calificacion(){
        //Realizamos la comparación entre las etiquetas;
        switch (nota)
        {
            case 9, 10: System.out.println("Excelente.");
                break;
            case 8 : System.out.println("Notable.");
                break;
            case 6, 7 : System.out.println("Acredito.");
                break;
            case 0,1,2,3,4,5 : System.out.println("No acredito.");
                break;
            default:
                System.out.println("no es posible esta nota.");
        }
        System.out.println("Final de programa.");
    }
}
```

Figura 16. *Código clase AsignaCalificacion.*

b) Compilar y ejecutar el programa (Realizar pruebas).

```
Introduzca la calificación (1 - 10), pulse enter: 4
No acredito.
Final de programa.
Introduzca la calificación (1 - 10), pulse enter: 6
Acredito.
Final de programa.
Introduzca la calificación (1 - 10), pulse enter: 8
Notable.
Final de programa.
Introduzca la calificación (1 - 10), pulse enter: 10
Excelente.
Final de programa.
```

Figura 17. *Pruebas de ejecución del programa.*

Cuando se ejecuta la sentencia ***switch***, se evalúa si el valor de la expresión es igual al valor de una opción; entonces se transfiere la secuencia de control a las sentencias asociadas con la opción correspondiente. Si ninguna opción coincide con el valor de nota, se ejecuta la sentencia ***default*** y las siguientes; por lo general, la última

sentencia que sigue a ***case*** es ***break***; esta última hace que el flujo de control del programa salte a la última sentencia de ***switch***. Si no existiera ***break*** también se ejecutarán las sentencias restantes de ***switch***.

ÍNDICE

2.2.2 Autoevaluación del aprendizaje:

Instrucciones: Completa los espacios en blanco colocando el número de la estructura de selección case que corresponde y selecciona la respuesta que muestra la correspondencia correcta.

1. ***else if***	2. ***case***	3. ***switch***	4. ***break***	5. ***if***	6. ***else***

a)
b)
c)
d)
e)

```java
import java.util.Scanner;
public class Calculadora {
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);
        int opcion;
        do {
            // Mostrar el menú
            System.out.println("\n--- CALCULADORA ---");
            System.out.println("1. Suma");
            System.out.println("2. Resta");
            System.out.println("3. Multiplicación");
            System.out.println("4. División");
            System.out.println("5. Salir");
            System.out.print("Elige una opción: ");
            opcion = teclado.nextInt();
            if (opcion >= 1 && opcion <= 4) {
                // Pedir los números solo si la opción es una operación válida
                System.out.print("Ingresa el primer número: ");
                double num1 = teclado.nextDouble();
                System.out.print("Ingresa el segundo número: ");
                double num2 = teclado.nextDouble();

                // Evaluar la opción con switch
                -------- (opcion) {
                    case 1:
                        System.out.println("Resultado: " + (num1 + num2));
                        break;
                    -------- 2:
                        System.out.println("Resultado: " + (num1 - num2));
                        break;
                    case 3:
                        System.out.println("Resultado: " + (num1 * num2));
                        --------;
                    -------- 4:
                        if (num2 != 0) {
                            System.out.println("Resultado: " + (num1 / num2));
                        } else {
                            System.out.println("Error: No se puede dividir entre 0.");
                        }
                        --------;
                }
            } else if (opcion == 5) {
                System.out.println("Saliendo del programa...");
            } else {
                System.out.println("Opción no válida. Intenta de nuevo.");
            }
        } while (opcion != 5); // Se repite hasta que el usuario elija salir
        teclado.close();
    }
}
```

Opciones de respuesta:

A) a:3 - b:2 - c:4 - d:2 - e:4

B) a:2 - b:1 - c:4 - d:3 - e:3

C) a:3 - b:2 - c:1 - d:4 - e:3

D) a:2 - b:4 - c:1 - d:3 - e:3

ÍNDICE

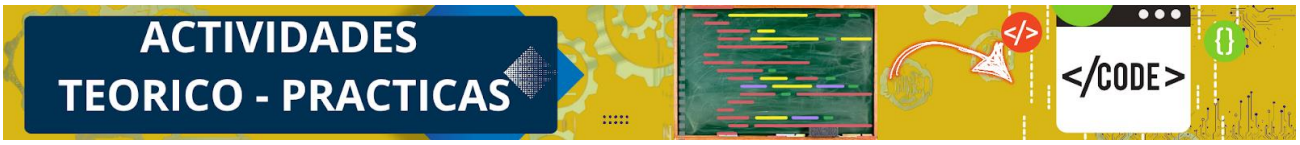
APRENDIZAJE 2.3

Desarrolla programas para resolver problemas que involucren la estructura repetitiva *for* en los métodos de una Clase.

Temática:

Estructura repetitiva: *for*.

Implementación



ÍNDICE

2.3.1. Actividades de aprendizaje teórico–prácticas.

Java proporciona tres tipos de estructuras repetitivas: ***for, while y do while***; el primero es la mejor forma de programar la ejecución de un bloque de sentencias un número fijo de veces.

Sintaxis:

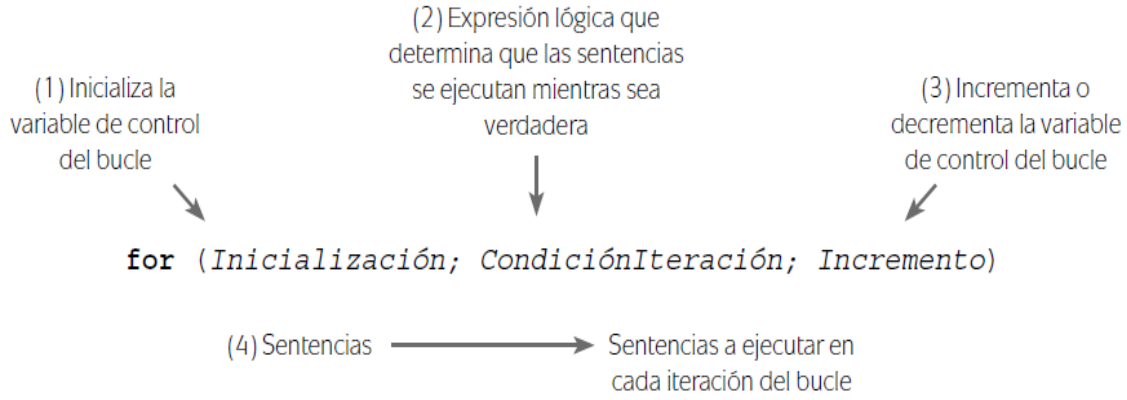


Diagrama:

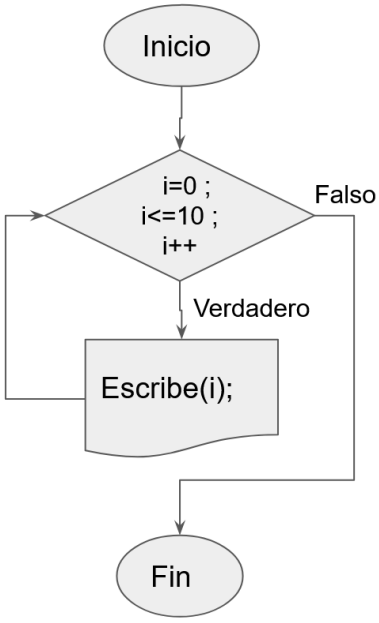


Figura 18. Diagrama de flujo de la estructura repetitiva ***for***.

Estructura repetitiva *for*.

Ejemplo 5. Utilizando la sentencia de repetición ***for***, crear un programa que pida al usuario ingresar un número entero, el cual después muestre la tabla a multiplicar de ese número ingresado.

Respuesta:

a) Escribir el siguiente código.

Clase principal:

```java
import java.util.Scanner;
//Creamos la clase Principal
public class Principal{
    public static void main(String args[]){
        //Declaramos la variables
        int numero;
        //Creamos los 2 objetos teclado y objeto
        Scanner teclado =  new Scanner(System.in);
        TablaMultiplicar objeto = new TablaMultiplicar();
        //Mandamos el mensaje  y guardamos el valor
        System.out.print("Ingresa un número: ");
        numero = teclado.nextInt();
        objeto.setNumero(numero);
        //Mandamos a llamar el método Tablas de la clase
        //TablaMultiplicar
        objeto.Tablas();
    }
}
```

Figura 19. *Código clase Principal.*

Clase TablaMultiplicar:

```java
//Creamos la clase TablaMultiplicar
public class TablaMultiplicar{
    //Declaramos los atributos
    private int numero;
    private int tabla;
    private int i;
    //Creamos el constructor
    public TablaMultiplicar(){
        this.numero= 0;
        this.tabla=0;
        this.i=0;
    }
    //Métodos set y get
    public void setNumero(int numero){
        this.numero=numero;
    }
    public int getNumero(){
        return numero;
    }
    //Método Tablas
    public void Tablas(){
        //Muestra tabla de multiplicar;
        for(i = 1; i<=10; i++){
            tabla = numero * i;
            System.out.println(numero + "*"+ i +"="+ tabla);
        }
    }
}
```

Figura 20. *Código clase TablaMultiplicar.*

b) Compilar y ejecutar el programa (Realizar pruebas).

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

```
Ingresa un número: 10
10*1=10
10*2=20
10*3=30
10*4=40
10*5=50
10*6=60
10*7=70
10*8=80
10*9=90
10*10=100
```

Figura 21. *Pruebas de ejecución del programa.*

ÍNDICE

2.3.2 Autoevaluación del aprendizaje:

Instrucciones: Selecciona la opción que contiene todas las respuestas correctas.

I.- ¿Cuál es la sintaxis de la sentencia repetitiva *for*?

a. ***for {inicialización; condición; incremento}***
b. ***for (incremento; condición; inicialización)***
c. ***for (inicialización; condición; incremento)***
d. ***for [incremento; condición; inicialización]***

II.- ¿Qué sucede si no se cumple inicialmente la condición de la sentencia repetitiva *for*?

a. Termina su ejecución.
b. Entra en un ciclo infinito.
c. No se ejecuta el ciclo.
d. Se detiene temporalmente.

III.- ¿Cuál es la diferencia principal entre un *for* y un *while* en *Java*?

a. ***for*** solo se usa con números, ***while*** con booleanos.
b. ***for*** es preferible cuando se conoce el número de iteraciones.
c. ***while*** siempre itera al menos una vez.
d. Ambos son intercambiables sin efectos diferentes.

IV.- ¿Cómo podrías optimizar un bucle *for* para reducir el número de iteraciones si conoces el rango de valores?

a. Incrementando siempre de uno en uno.
b. Utilizando una estructura if en cada iteración.
c. Ajustando el valor de incremento y el límite superior.
d. Colocando la condición fuera del bloque del bucle.

V.- ¿Cuáles son los elementos que conforman un bucle *for*?

a) Inicio, fin, condición, incremento.

b) Declaración, iteración, prueba, condición.

c) Inicio, declaración, final, comparación.

d) Inicialización, condición, actualización.

Opciones de respuesta:

A) I:a - II:c - III:b -IV:c - V:d

B) I:b - II:b - III:b -IV:c - V:a

C) I:c - II:a - III:b -IV:a - V:a

D) I:c - II:c - III:b -IV:c - V:d

ÍNDICE

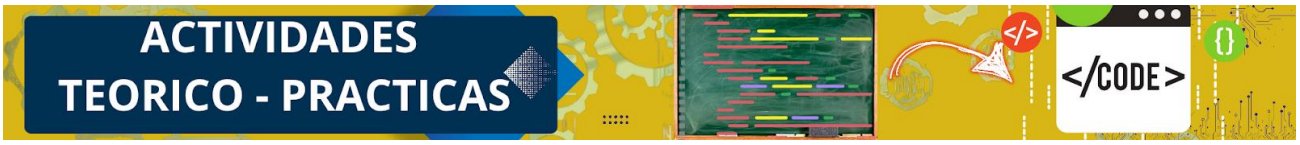
APRENDIZAJE 2.4

Desarrolla programas que involucren la estructura repetitiva *while* en los métodos de una Clase.

Temática:

Estructura repetitiva: *while*.

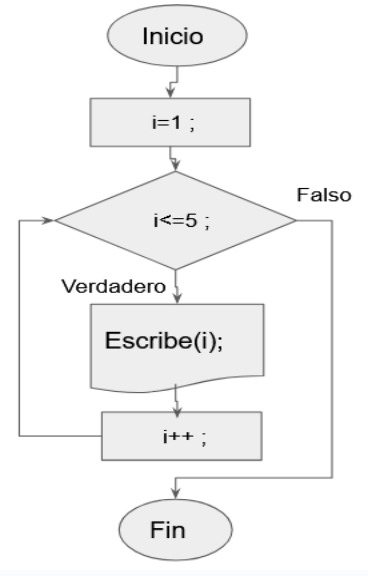
Implementación



ÍNDICE

2.4.1. Actividades de aprendizaje teórico–prácticas.

En la estructura repetitiva ***while***, la condición se evalúa antes de ejecutar el código dentro del bucle y el ciclo termina hasta que la condición sea falsa.

SINTAXIS:	DIAGRAMA:
while(condición_bucle){ ***sentencia;*** ***}***	 **Figura 22.** Diagrama de flujo de la estructura repetitiva ***while***.

Ejemplo 6. Utilizando la sentencia de repetición ***while***, crear un programa que pida al usuario ingresar un número entero, el cual después muestre la tabla a multiplicar de ese número ingresado.

Respuesta:

a) Escribir el siguiente código.

Clase principal:

```
import java.util.Scanner;
//Creamos la clase Principal
public class Principal{
    public static void main(String args[]){
        // //Declaramos la variables
        int numero;
        // //Creamos dos objetos teclado y objeto
        Scanner teclado =  new Scanner(System.in);
        TablaWhile objeto = new TablaWhile();
        //Mandamos el mensaje  y guardamos el valor
        System.out.print("Escriba la tabla a calcular: ");
        numero = teclado.nextInt();
        objeto.setNumero(numero);
        //Mandamos a llamar el método Tablas de la clase
        //TablaWhile
        objeto.Tablas();
    }
}
```

Figura 23. *Código clase Principal.*

Clase TablaWhile:

```
import java.util.Scanner;
//Creamos la clase TablaWhile
public class TablaWhile{
    //Declaramos los atributos
    private int numero;
    private int tabla;
    private int i;
    //Creamos el constructor
    public TablaWhile(){
        this.tabla= 0;
        this.i=1;
        this.numero=0;
    }
    //Métodos set y get
    public void setNumero(int numero){
        this.numero=numero;
    }
    public int getNumero(){
        return numero;
    }
    //Método Tablas
    public void Tablas(){
        // La fase de procesamiento usa la repetición controlada
        while (i <= 10) // itera 10 veces
        {
            tabla = numero * i;
            System.out.println(numero + "*"+ i +"="+ tabla);
            i = i + 1;
        }
    } // fin del método
}
```

Figura 24. *Código clase TablaWhile.*

b) Compilar y ejecutar el programa (Realizar pruebas).

```
Escriba la tabla a calcular: 3
3*1=3
3*2=6
3*3=9
3*4=12
3*5=15
3*6=18
3*7=21
3*8=24
3*9=27
3*10=30
```

```
Escriba la tabla a calcular: 8
8*1=8
8*2=16
8*3=24
8*4=32
8*5=40
8*6=48
8*7=56
8*8=64
8*9=72
8*10=80
```

Figura 25. *Pruebas de ejecución del programa.*

ÍNDICE

2.4.2 Autoevaluación del aprendizaje:

Ejercicios de relación de columnas, bucle ***while*** en ***Java***.

Instrucciones: Escribe en el paréntesis de la columna A, la letra que corresponda al concepto de la columna B y selecciona la respuesta que muestra la correspondencia correcta.

Columna A	**Columna B**
I. Inicialización de ciclo. II. Condición lógica de entrada. III. Concepto de bucle infinito. IV. Sentencia que realiza el incremento dentro del bucle. V. En qué momento se da la salida del bucle.	**a.** ***while (i <= 10)*** **b.** Cuando la condición lógica sea falsa. **c.** El bucle no termina, no hay incremento o decremento de la variable de control. **d.** ***i = 0;*** **e.** ***i==*** **f.** ***i++***

Opciones de respuesta:

A) I:a - II:c - III:b -IV:e - V:d

B) I:d - II:a - III:c - IV:f - V:b

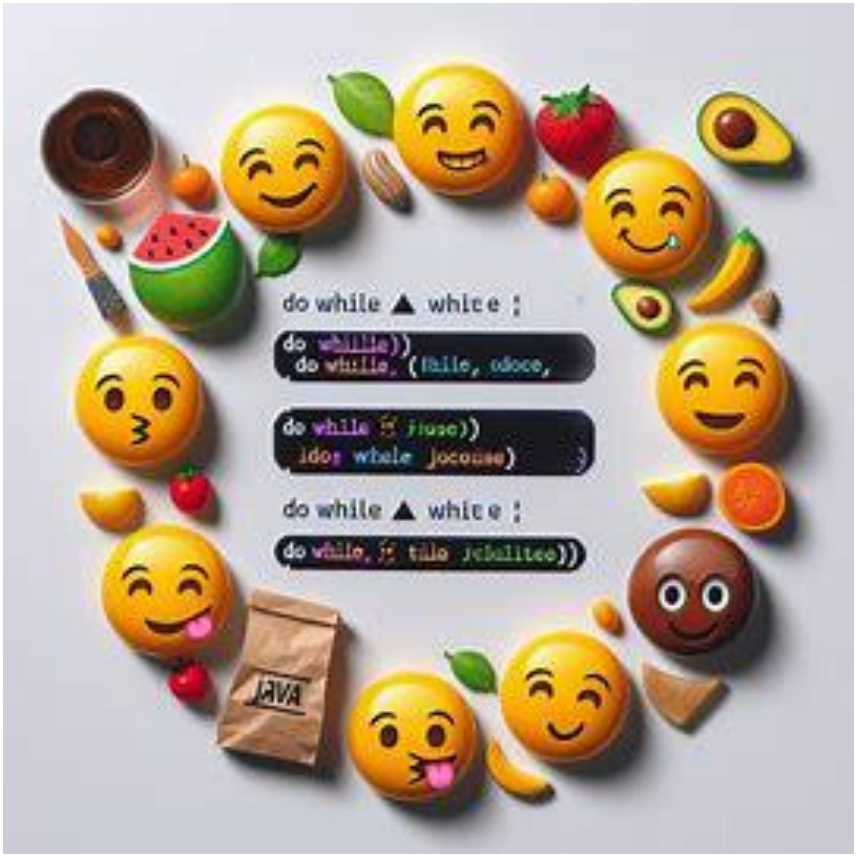
C) I:c - II:a - III:b -IV:f - V:e

D) I:a - II:c - III:e -IV:b - V:d

ÍNDICE

APRENDIZAJE 2.5

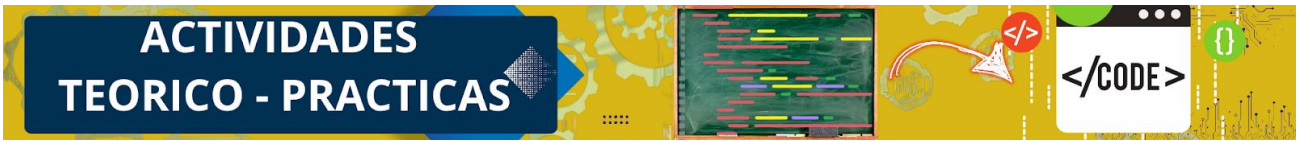
Desarrolla programas que involucren la estructura repetitiva *do-while* en los métodos de una Clase.



Temática:

Estructura repetitiva: *do-while*.

Implementación



ÍNDICE

2.5.1. Actividades de aprendizaje teórico–prácticas.

Una estructura repetitiva ***do-while*** es similar a ***while*** excepto que el cuerpo del bucle siempre se ejecuta al menos una vez.

```
while (expresión_lógica){
     sentencia_1;
     sentencia_2;
     . . .
     sentencia_n;
   }
```

```
do{
    sentencia_1;
    sentencia_2;
    . . .
    sentencia_n;
 } while (expresión_lógica);
```

La construcción ***do*** comienza ejecutando las *sentencias*; en seguida se evalúa la *expresión*; si ésta es verdadera, entonces se repite la ejecución de las sentencias; continuando hasta que la expresión sea falsa.

SINTAXIS:	DIAGRAMA:

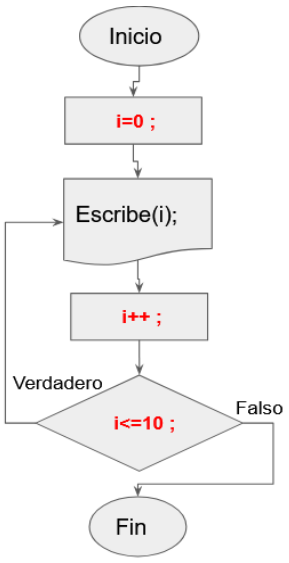
`do {`
`  sentencia;`
`} while(condición);`
	

Figura 26. Diagrama de flujo de la estructura repetitiva ***do-while***.

Ejemplo 7. Utilizando la sentencia de repetición **`do-while`**, crear un programa que pida al usuario ingresar un número entero, el cual después muestre la tabla a multiplicar de ese número ingresado.

Respuesta:

a) Escribir el siguiente código.

Clase Principal:

```
import java.util.Scanner;
//Creamos la clase Principal
public class Principal{
    public static void main(String args[]){
        // //Declaramos la variables
        int numero;
        // //Creamos dos objetos teclado y objeto
        Scanner teclado =  new Scanner(System.in);
        TablaDoWhile objeto = new TablaDoWhile();
        //Mandamos el mensaje  y guardamos el valor
        System.out.print("Escriba la tabla a calcular: ");
        numero = teclado.nextInt();
        objeto.setNumero(numero);
        //Mandamos a llamar el método Tablas de la clase
        //TablaDoWhile
        objeto.Tablas();
    }
}
```

Figura 27. *Código clase Principal.*

Clase Saludo:

```
import java.util.Scanner;
//Creamos la clase TablaDoWhile
public class TablaDoWhile{
    //Declaramos los atributos
    private int numero;
    private int tabla;
    private int i;
    //Creamos el constructor
    public TablaDoWhile(){
        this.tabla= 0;
        this.i=1;
        this.numero=0;
    }
    //Métodos set y get
    public void setNumero(int numero){
        this.numero=numero;
    }
    public int getNumero(){
        return numero;
    }
    //Método Tablas
    public void Tablas(){
        // La fase de procesamiento usa la repetición controlada
        do{
            tabla = numero * i;
            System.out.println(numero + "*"+ i +"="+ tabla);
            i = i + 1;
        }
        while (i <= 10); // itera 10 veces
    }
} // fin del método
```

Figura 28. *Código clase TablaDoWhile.*

b) Compilar y ejecutar el programa (Realizar pruebas).

```
Escriba la tabla a calcular: 5
5*1=5
5*2=10
5*3=15
5*4=20
5*5=25
5*6=30
5*7=35
5*8=40
5*9=45
5*10=50
```

Figura 29. *Pruebas de ejecución del programa.*

ÍNDICE

2.5.2 Autoevaluación del aprendizaje:

Instrucciones: Completa los espacios en blanco colocando el número de la estructura condicional ***do-while*** correspondiente y selecciona la respuesta que muestra la correspondencia correcta.

1. ***{***	2. ***}***	3. ***while***	4. ***do***

a)

b)

```java
import java.util.Scanner;
public class PromedioPositivos {
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);
        int sum = 0;
        int count = 0;
        int num;
        --------  { // Completa aquí
            System.out.print("Ingresa un número positivo (o un número negativo para terminar): ");
            num = teclado.nextInt();

            if (num >= 0) {
                sum += num;
                count++;
            }
        } --------  (num >= 0); // Completa aquí
        if (count > 0) {
            double promedio = (double) sum / count;
            System.out.println("El promedio de los números positivos es: " + promedio);
        } else {
            System.out.println("No ingresaste ningún número positivo.");
        }
    }
}
```

Opciones de respuesta:

A) a:1 - b:2

B) a:2 - b:1

C) a:3 - b:2

D) a:4 - b:3

ÍNDICE

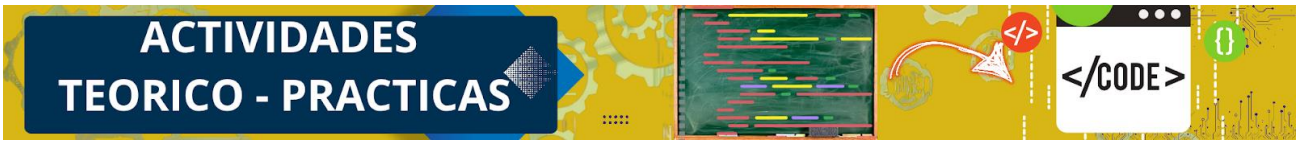
APRENDIZAJE 2.6

Resuelve problemas que involucren el uso de los arreglos unidimensionales en los métodos de una Clase.

Temática:

Arreglos unidimensionales:

- Concepto.
- Declaración.
- Tipos.
- Uso.



ÍNDICE

2.6.1. Actividades de aprendizaje teórico–prácticas.

Arreglo:

Es una secuencia de datos, del mismo tipo, llamados elementos, y pueden ser datos simples de Java, o de una clase previamente declarada como tal.

Se declara de modo similar a otros tipos de datos, excepto que para especificar que se trata de un arreglo, se hace uso de corchetes, de dos formas:

1° A continuación del tipo de datos.
2° Después del nombre del arreglo.

La sintaxis de declaración de variables arreglo en Java es:	***tipo_dato [] nombre_arreglo;*** ***tipo_dato nombre_arreglo[];***

Tipos: Los elementos de un arreglo pueden ser de tipo primitivos.

Uso: Normalmente el arreglo se utiliza para almacenar tipos ***char***, ***String***, ***int*** o ***float***. Un arreglo puede contener, por ejemplo, la edad de los alumnos de una clase, las temperaturas de cada día de un mes. Cada ítem del arreglo se denomina elemento.

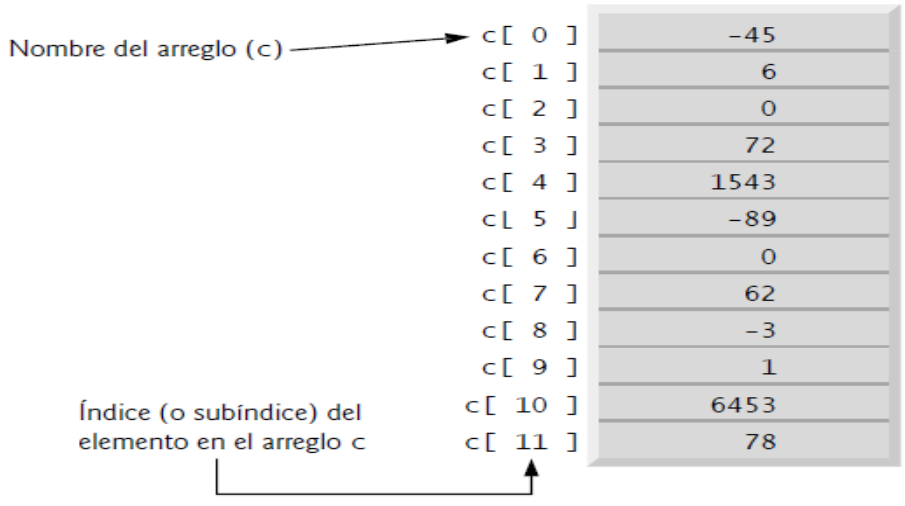


Figura 30. Ejemplo visual de un arreglo.

En la **Figura 30** se muestra una representación lógica de un arreglo de enteros, llamado c. Este arreglo contiene 12 *elementos*. Un programa puede hacer referencia a cualquiera de estos elementos mediante una **expresión de acceso a un arreglo** que contiene el *nombre* del arreglo, seguido por el *índice* del elemento específico encerrado entre **corchetes** (***[ ]***). El primer elemento en cualquier arreglo tiene el **índice cero**, y algunas veces se le denomina **elemento cero**. Por lo tanto, los elementos del arreglo c son c[0], c[1], c[2], y así en lo sucesivo. El mayor índice en el arreglo c es 11: 1 menos que 12, el número de elementos en el arreglo. Los nombres de los arreglos siguen las mismas convenciones que los demás nombres de variables.

ÍNDICE

2.6.2 Autoevaluación del aprendizaje:

Relación de columnas: Arreglos en ***Java***

Instrucciones: Relaciona los conceptos en la Columna A con sus descripciones o ejemplos en la Columna B; y selecciona la respuesta que muestra la correspondencia correcta.

Columna A	Columna B
I. Es la forma correcta de declarar un arreglo unidimensional II. Es lo forma en que muestra un elemento de un arreglo unidimensional III. Forma de declaración un arreglo bidimensional de 3x3 IV. Inicialización válida de un arreglo unidimensional de 3 enteros. V. ¿Qué es un arreglo bidimensional? VI. La forma de indicar el acceso a un elemento específico de un arreglo bidimensional también llamada una matriz 2D	a. ***int[][] matriz = new int[3][3];*** b. Espacio de almacenamiento con filas y columnas c. ***int[] arreglo = new int[5];*** d. ***System.out.println(arreglo[])*** e. ***int[] arreglo = new int[]{1, 2, 3}*** f. ***System.out.println(arreglo[2]);*** b) ***matriz[1][2]***

Opciones de respuesta:

A) I:c -II:f -III:a - IV:e - V:b - VI: g

B) I:a -II:f -III:b - IV:c - V:g - VI: d

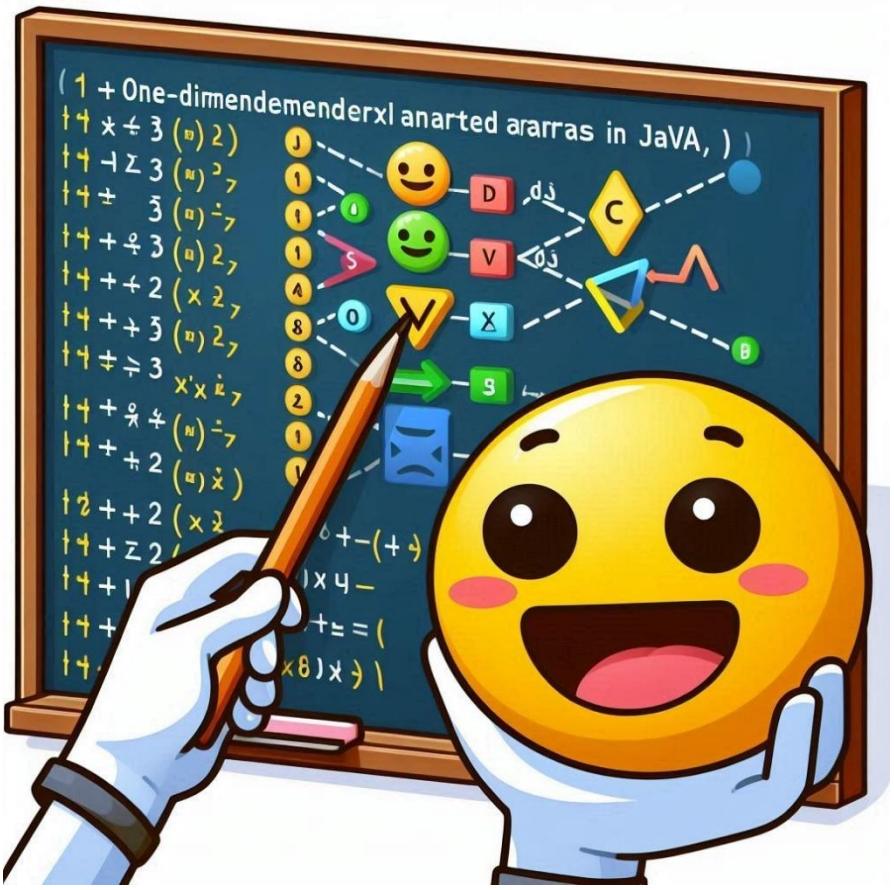
C) I:b -II:d -III:e - IV:f - V:c - VI: a

D) I:f -II:a -III:d - IV:g - V:b - VI: c

ÍNDICE

APRENDIZAJE 2.7

Desarrolla programas que involucren el uso de los arreglos unidimensionales en los métodos de una Clase.

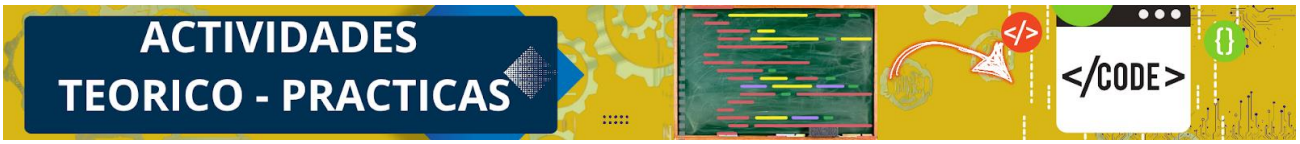


Temática:

Aplicación de arreglos unidimensionales.

- Acceso.
- Recorrido.

Implementación.



ÍNDICE

2.7.1. Actividades de aprendizaje teórico–prácticas.

Acceso y recorrido de un arreglo unidimensional

Uso de un inicializador de arreglos

Puedes crear un arreglo e inicializar sus elementos con un inicializador de arreglo, que es una lista de expresiones separadas por comas, también conocida como lista inicializadora, y que está encerrada entre llaves. En este caso, la longitud del arreglo se determina con base en el número de elementos en la lista inicializadora. Por ejemplo, la declaración

```
int[] n = { 10, 20, 30, 40, 50 };
```

crea un arreglo de *cinco* elementos con los valores de índices 0 a 4. El elemento n[0] se inicializa con 10, n[1] se inicializa con 20, y así en lo sucesivo. Cuando el compilador encuentra la declaración de un arreglo que incluye una lista inicializadora, cuenta el número de inicializadores en la lista para determinar el tamaño del arreglo y después establece la operación new apropiada “tras bambalinas”.

Enseguida se muestra un ejemplo al respecto.

Implementación de programas que involucren arreglos unidimensionales.

Ejemplo 8. Inicializa un arreglo de 10 enteros con 10 valores y muestra el arreglo en formato de tabla.

Respuesta:

a) Escribir el siguiente código.

```
public class InicArreglo
{
    public static void main(String[] args)
    {
        //La lista inicializadora especifica el valor para cada elemento
        int[] arreglo = {32, 27, 64, 18, 95, 14, 90, 70, 60, 37}; // crea el objeto arreglo
        System.out.println("Indice  " + "Valor"); // encabezados de columnas
        // imprime el valor de cada elemento del arreglo
        for (int contador = 0; contador < arreglo.length; contador++)
            System.out.println( "  "+ contador + "       "+ arreglo[contador]);
    }
}
```

Figura 31. Código de la clase InicArreglo*.*

b) Compilar y ejecutar el programa (Realizar pruebas).

```
Indice  Valor
  0       32
  1       27
  2       64
  3       18
  4       95
  5       14
  6       90
  7       70
  8       60
  9       37
```

Figura 32. *Pruebas de ejecución del programa.*

ÍNDICE

2.7.2 Autoevaluación del aprendizaje:

Cuestionario: Arreglos bidimensionales.

Instrucciones: Selecciona la opción que contiene todas las respuestas correctas.

Declaración correcta de una matriz de enteros.

I. ¿Cuál de las siguientes opciones declara correctamente una matriz de enteros de 3 filas x 2 columnas en Java?

a) `int matriz = new int[3,2];`
b) `int[][] matriz = new int[3][2];`
c) `int matriz[][] = new int[3,2];`
d) `int matriz[3][2] = new int[][];`

Inicialización correcta de una matriz

II. ¿Cuál de las siguientes opciones muestra una inicialización válida de una matriz de 2 filas x 3 columnas en *Java*?

a) `int matriz = { {1, 2, 3}, {4, 5, 6} };`
b) `int[][] matriz = { {1, 2, 3}, {4, 5, 6} };`
c) `int[][] matriz = new int[2][3] { {1, 2, 3}, {4, 5, 6} };`
d) `int[][] matriz = new int[2,3] { {1, 2, 3}, {4, 5, 6} };`

Acceso correcto a un elemento de una matriz

III: Si tenemos la siguiente matriz:

```
int[][] matriz = { {10, 20, 30}, {40, 50, 60} };
```

¿Cuál de las siguientes opciones accede correctamente al valor 50 de la matriz?

a) `System.out.println(matriz[2][1]);`
b) `System.out.println(matriz[2,2]);`
c) `System.out.println(matriz[1][1]);`
d) `System.out.println(matriz(1)(1));`

Opciones de respuesta:
A) I:a -II:c - III:b
B) I:b -II:b - III:c
C) I:b -II:b - III:b
D) I:c -II:b - III:c

ÍNDICE

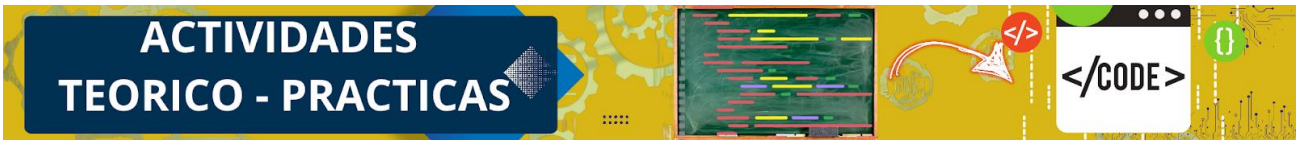
APRENDIZAJE 2.8

Realiza programas que involucren usar arreglos bidimensionales.

Temática:

Arreglos bidimensionales.

- Concepto.
- Declaración.
- Uso.



ÍNDICE

2.8.1. Actividades de aprendizaje teórico–prácticas.

Hasta este punto todos los arreglos fueron unidimensionales o de una sola dimensión, caracterizados por tener un solo subíndice; estos arreglos también se conocen como listas. Los arreglos multidimensionales tienen más de una dimensión y, en consecuencia, más de un índice, los más utilizados son los de dos dimensiones, conocidos como tablas o matrices; sin embargo, es posible crear arreglos de tantas dimensiones como las aplicaciones requieren: tres, cuatro o más.

Un arreglo de dos dimensiones equivale a una tabla con múltiples filas y múltiples columnas, (**ver Figura 33**).

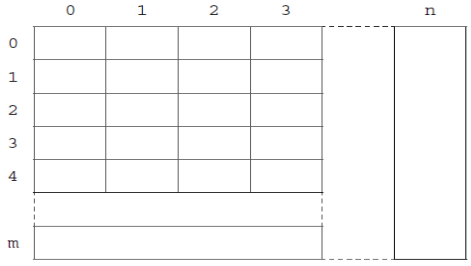


Figura 33. Estructura de un arreglo de dos dimensiones.

Si las filas se etiquetan de ***0*** a ***m*** y las columnas de ***0*** a ***n***, el número de elementos que tendrá el arreglo será el resultado del producto ***(m+1)*(n+1)***; un elemento es localizado mediante las coordenadas representadas por su número de fila y de columna ***(a, b)***.

La sintaxis para la declaración de un arreglo de dos dimensiones es:

```
<tipo de datoElemento> <nombre arreglo > [] [];
```

o bien

```
<tipo de datoElemento> [] []<nombre arreglo >;
```

Ejemplos de declaración de matrices:

```
char pantalla[][];
int puestos[][];
double [][]matriz;
```

Estas declaraciones no reservan memoria para los elementos de la matriz, en realidad son referencias; para reservar memoria y especificar el número de filas y de columnas.

El operador ***new*** se utiliza para crear las respectivas matrices:

```
pantalla = new char[80][24]; // 80 filas y 24 columnas
puestos = new int[10][5]; // 10 filas por 5 columnas
final int N = 4;
matriz = new double[N][N]; // matriz cuadrada de N*N elementos
```

El operador ***new*** se puede aplicar a la vez que se hace la declaración; la sintaxis para definir una matriz es:

```
<tipo de datoElemento> <nombre arreglo >[][]=
new<tipo de datoElemento> [<NúmeroDeFilas<]
[<NúmeroDeColumnas>];
```

Un arreglo bidimensional en realidad es un *arreglo de arreglos*; es decir, un arreglo unidimensional donde cada elemento no es un valor entero, de coma flotante o carácter, sino otro arreglo.

Los elementos se almacenan en memoria de modo que el subíndice más próximo al nombre del arreglo es la fila y el otro subíndice, la columna; la **Tabla 1** representa a todos los elementos y sus posiciones relativas en memoria del

arreglo int [][]tabla = new int[4][2].

Elemento	Posición relativa de memoria
Tabla [0][0]	0
Tabla [0][1]	4
Tabla [1][0]	8
Tabla [1][1]	12
Tabla [2][0]	16
Tabla [2][1]	20
Tabla [3][0]	24
Tabla [3][1]	28

Tabla 1. Ejemplo de arreglo bidimensional en la memoria.

Inicialización de arreglos multidimensionales

La manera recomendada de inicialización es encerrar entre llaves la lista de constantes de cada fila separadas por comas, como en los ejemplos siguientes:

a) Definir una matriz de 2 filas por 3 columnas en cada fila (ver **Tabla 2**.):

```
int tabla1[][] = { {51, 52, 53},{54, 55, 56} };
```

		0	1	2	Columnas
Filas	**0**	51	52	53	
	1	54	55	56	

Tabla 2. Matriz de 2 filas por 3 columnas.

b) Definir una matriz de 3 filas por 4 columnas en cada fila (ver **Tabla 3):**

```
int tabla2[][] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };
```

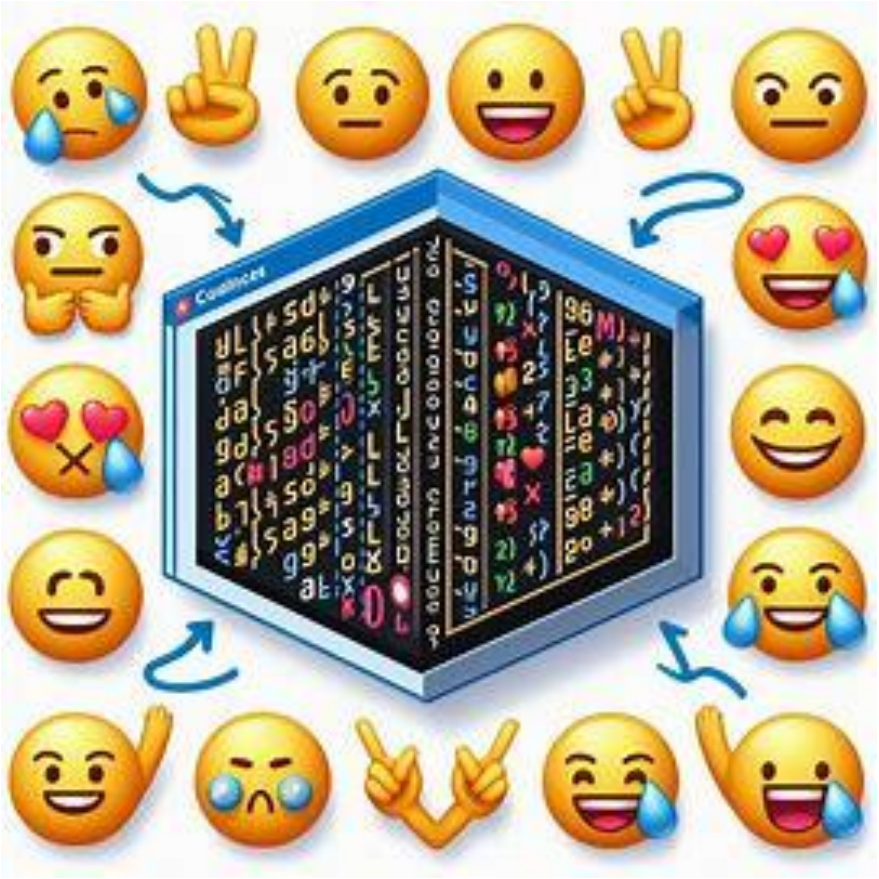
		0	1	2	3	Columnas
Filas	**0**	1	2	3	4	
	1	5	6	7	8	
	2	9	10	11	12	

Tabla 3. Matriz de 3 filas por 4 columnas.

ÍNDICE

APRENDIZAJE 2.9

Desarrolla programas que involucren el uso de los arreglos bidimensionales en los métodos de una Clase.

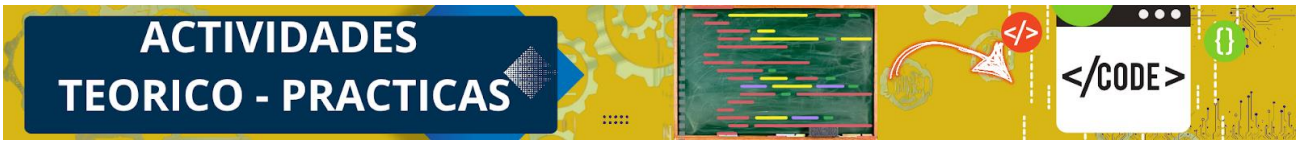


Temática:

Aplicación de los arreglos bidimensionales.

- Con estructuras repetitivas.
- Acceso.
- Recorrido.

Implementación.



2.9.1. Actividades de aprendizaje teórico–prácticas.

Ejemplo 9. Crear un programa para dar entrada a los elementos que contendrá el arreglo y posteriormente visualizarlos en un arreglo bidimensional.

El método ***`Leer()`*** da entrada a los elementos de la matriz que se pasa como argumento, y el método visualizar() muestra la tabla en la pantalla.

Respuesta:

a) Escribir el siguiente código.

```
import java.util.Scanner;
public class ArregloBi{
     //Creamos el método leer
    static void leer(int a[][]){
        int i,j;
        Scanner entrada = new Scanner(System.in);
        System.out.println("Entrada de datos de la matriz");
        for (i = 0; i < a.length; i++)
        {
            System.out.println("Fila: " + i);
            for (j = 0; j < a[i].length; j++)
                a[i][j]= entrada.nextInt();
        }
    }
     //Creamos el método visulizar
    static void visualizar (int a[][]){
        int i,j;
        System.out.println("\n\t Matriz leída\n");
        for (i = 0; i < a.length; i++)
        {
            for (j = 0; j < a[i].length; j++)
                System.out.print(a[i][j] + " ");
            System.out.println(" ");
        }
    }
    //Creamos el método main
    public static void main(String [] args){
        int v[][]= new int[3][5];
        leer(v);
        visualizar(v);
    }
}
```

Figura 34. Código de la clase ArregloBi.

b) Compilar y ejecutar el programa (Realizar pruebas).

```
Opciones
Entrada de datos de la matriz
Fila: 0
1
2
3
4
5
Fila: 1
9
8
7
6
5
Fila: 2
10
11
12
13
14

         Matriz leída

1 2 3 4 5
9 8 7 6 5
10 11 12 13 14
```

Figura 35. *Pruebas de ejecución del programa.*

ÍNDICE

APRENDIZAJE 2.10

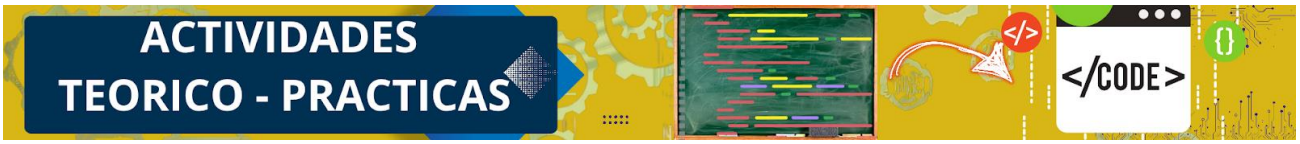
Desarrolla un proyecto que utilice las sentencias vistas hasta el momento, incluyendo los arreglos.

Temática:

Diseño de una aplicación.

Planteamiento del problema.

- Diseño.
- Desarrollo.



ÍNDICE

2.10.1. Actividades de aprendizaje teórico–prácticas.

Proyecto Final

Se solicita crear un proyecto llamado Calificaciones, este programa deberá de solicitar al usuario que seleccione una opción de un menú como se describe a continuación:

Menú de opciones:

1) **Ingresar calificaciones:** en esta opción se deberá solicitar al usuario que ingrese sus calificaciones, con un máximo de cinco registros.
2) **Mostrar calificaciones:** para esta segunda opción del menú, deberá de mostrar las calificaciones ingresadas por el usuario.
3) **Calcular promedio:** aquí se deberá de calcular el promedio de las calificaciones registradas por el usuario y si el usuario no ha ingresado ninguna calificación mandar el mensaje de sin registró de calificaciones.
4) **Mostrar mayor y menor calificación:** en esta opción se deberá de mostrar la mayor calificación registrada por el usuario o la menor calificación.
5) **Salir:** finalmente en esta opción se deberá de terminar el programa.

La solución respectiva se muestra enseguida…

Respuesta: Para la realización de este proyecto se recomienda utilizar las estructuras vistas a lo largo de esta unidad, condicionales (***if, if-else, switch***), de repetición (***for, while, do-while***) y el uso de los arreglos.

a) Escribir el siguiente código.

```
import java.util.Scanner;
public class GestionCalif {
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);
        int opcion;
        double[] calificaciones = new double[5];
        int contador = 0;
        //Mostrar el menú del programa
        do {
            System.out.println("\nMenú de opciones:");
            System.out.println("1. Ingresar calificaciones");
            System.out.println("2. Mostrar calificaciones");
            System.out.println("3. Calcular promedio");
            System.out.println("4. Mostrar mayor y menor calificación");
            System.out.println("5. Salir");
            System.out.print("Seleccione una opción: ");
            opcion = teclado.nextInt();

            switch (opcion)
            {
                case 1://Opción 1, ingresar calificación
                    if (contador < calificaciones.length) {
                        System.out.print("Ingrese la calificación: ");
                        calificaciones[contador] = teclado.nextDouble();
                        contador++;
                    } else {
                        System.out.println("No se pueden ingresar más calificaciones.");
                    }
                    break;
                case 2://Opción 2, mostrar las calificaciones
                    System.out.println("Calificaciones registradas:");
                    for (int i = 0; i < contador; i++) {
                        System.out.println("Calificación " + (i + 1) + ": " + calificaciones[i]);
                    }
                    break;
                case 3:// Opción 3, calcular el promedio de las calificaciones
                    if (contador == 0) {
                        System.out.println("No hay calificaciones registradas.");
                    } else {
                        double suma = 0;
                        for (int i = 0; i < contador; i++) {
                            suma += calificaciones[i];
                        }
                        double promedio = suma / contador;
                        System.out.println("El promedio de las calificaciones es: " + promedio);
                    }
                    break;
                case 4:// Opción 4, mostra la mayor y menor calificación
                    if (contador == 0) {
                        System.out.println("No hay calificaciones registradas.");
                    } else {
                        double mayor = calificaciones[0];
                        double menor = calificaciones[0];
```

```
                    for (int i = 1; i < contador; i++) {
                        if (calificaciones[i] > mayor) {
                            mayor = calificaciones[i];
                        }
                        if (calificaciones[i] < menor) {
                            menor = calificaciones[i];
                        }
                    }
                    System.out.println("Mayor calificación: " + mayor);
                    System.out.println("Menor calificación: " + menor);
                }
                break;
            case 5:// Opción 5, terminar el programa
                System.out.println("Saliendo del programa...");
                break;
            default:
                System.out.println("Opción no válida, intente de nuevo.");
        }

    } while (opcion != 5);
    teclado.close();
  }
}
```

Figura 35. Código de la clase GestionCalif*.*

b) Compilar y ejecutar el programa (Realizar pruebas).

```
Menú de opciones:
1. Ingresar calificaciones
2. Mostrar calificaciones
3. Calcular promedio
4. Mostrar mayor y menor calificación
5. Salir
Seleccione una opción:
```

```
Menú de opciones:
1. Ingresar calificaciones
2. Mostrar calificaciones
3. Calcular promedio
4. Mostrar mayor y menor calificación
5. Salir
Seleccione una opción: 1
Ingrese la calificación: 6
```

```
Menú de opciones:
1. Ingresar calificaciones
2. Mostrar calificaciones
3. Calcular promedio
4. Mostrar mayor y menor calificación
5. Salir
Seleccione una opción: 2
Calificaciones registradas:
Calificación 1: 6.0
Calificación 2: 7.0
Calificación 3: 8.0
Calificación 4: 9.0
Calificación 5: 6.0
```

```
Menú de opciones:
1. Ingresar calificaciones
2. Mostrar calificaciones
3. Calcular promedio
4. Mostrar mayor y menor calificación
5. Salir
Seleccione una opción: 3
El promedio de las calificaciones es: 7.2
```

```
Menú de opciones:
1. Ingresar calificaciones
2. Mostrar calificaciones
3. Calcular promedio
4. Mostrar mayor y menor calificación
5. Salir
Seleccione una opción: 4
Mayor calificación: 9.0
Menor calificación: 6.0
```

```
Menú de opciones:
1. Ingresar calificaciones
2. Mostrar calificaciones
3. Calcular promedio
4. Mostrar mayor y menor calificación
5. Salir
Seleccione una opción: 5
Saliendo del programa...
```

Figura 36. *Pruebas de ejecución del programa.*

ÍNDICE

2.11 RESPUESTAS A LAS AUTOEVALUACIONES.

AUTOEVALUACIÓN	RESPUESTA CORRECTA
2.1.2	Programa que identifica el mayor de 3 números: C) a:3 - b:2 - c:1
	Programa que identifica si eres mayor de edad: D) a:2 - b:4 - c:1 - d:3
	Programa que determina si un estudiante aprueba o reprueba: B) a:2 - b:1 - c:4 - d:3 - e:3
2.2.2	Programa de la estructura de selección: A) a:3 - b:2 - c:4 - d:2 - e:4
2.3.2	Cuestionario: Selecciona opción: D) I:c - II:c - III:b -IV:c - V:d
2.4.2	Relaciona columnas: Bucle ***while***. B) I:d - II:a - III:c - IV:f - V:b
2.5.2	Completar programa: Estructura condicional ***do-while***. D) a:4 - b:3
2.6.2	Relaciona columnas: Arreglos en ***Java***. A) I:c -II:f -III:a - IV:e - V:b - VI: g
2.7.2	Cuestionario: Arreglos bidimensionales. B) I:b -II:b - III:c

ÍNDICE

2.12 REFERENCIAS.

De Ernesto. (s/f). *[@LaGeekipediaDeErnesto]. Curso de programación JAVA desde cero*. Youtube. Recuperado el 13 de marzo de 2025, de http://www.youtube.com/playlist?list=PLyvsggKtwbLX9LrDnl1-K6QtYo7m0yXWB

Joyanes, L. & Zahonero, I. (2011). PROGRAMACIÓN EN JAVA 6. algoritmos, programación orientada a objetos e interfaz gráfica de usuarios.1ª Ed. Mac Graw Hill. CDMX. México.

Román, C. (s/f). *Programación con JAVA*. Unam. México. Recuperado el 30 de mayo 2024, de http://profesores.fi-b.unam.mx/carlos/java/

ÍNDICE

Propósito.

Al finalizar la unidad, cualquier estudiante implementará programas en Java mediante el uso de polimorfismo, constructores, colaboración y herencia de Clases para aprovechar las bondades de la programación orientada a objetos.

ÍNDICE

3.0 Conceptos Clave.

Constructor.

"El constructor de una clase es un método "especial" a través del cual podemos crear los objetos de la clase. Toda clase tiene al menos un constructor. Podemos programarlo explícitamente o bien aceptar el constructor por defecto que ***Java*** definirá por nosotros en caso de que no lo hayamos programado" (Sznajdleder, 2000, p. 44). El constructor tiene como finalidad la inicialización de los atributos de la clase y posiblemente ejecutar algunos de los métodos de la clase.

Colaboración entre clases.

Las clases en ***Java*** pueden trabajar juntas para lograr un objetivo común, normalmente en un problema resuelto con la metodología de programación orientada a objetos no interviene una sola clase, sino que hay muchas clases que interactúan y se comunican, a esto se le conoce como colaboración entre clases.

Herencia.

Es una forma de reutilización de código en la que se crea una nueva clase que recibe los atributos de una ya existente.

Implementación.

Se define como poner en funcionamiento, aplicar métodos y medidas para llevar a cabo algo (RAE, 2023).

Polimorfismo.

La palabra "polimorfismo" está formada por raíces griegas y significa "cualidad de tener muchas formas". Sus componentes léxicos son: polys (muchos) y morfo (formas). En la programación orientada a objetos se refiere a la habilidad de un objeto de realizar una acción de diferentes maneras utilizando métodos iguales que se implementan de formas diferentes en varias clases.

Reutilizar.

Volver a utilizar algo, bien con la función que desempeñaba anteriormente o con otros fines (RAE, 2023). En programación se le puede llamar optimización del código.

Subclase.

Es la clase hija que se deriva de una clase padre o superclase.

Superclase.

En los lenguajes de programación orientada a objetos como ***Java***, la superclase es la clase padre de donde se partirá para la creación de las clases hijas.

ÍNDICE

APRENDIZAJE 3.1

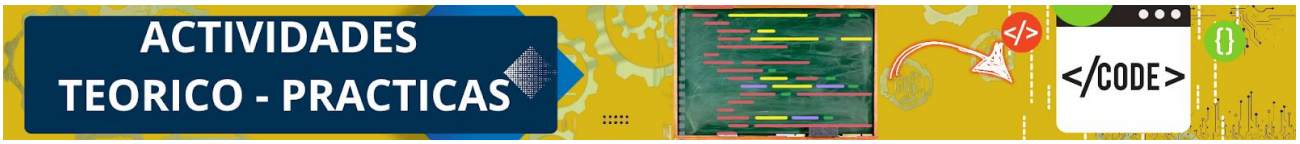
Conoce el concepto de polimorfismo y constructor; y desarrolla programas que los involucren.

Temática:

Concepto de polimorfismo.

Concepto de constructor.

Implementación de constructores con polimorfismo.



ÍNDICE

3.1.1. Actividades de aprendizaje teórico–prácticas.

El polimorfismo es uno de los conceptos fundamentales de la programación orientada a objetos. Se refiere a la capacidad de un objeto de tomar muchas formas diferentes. En ***Java***, el polimorfismo se puede implementar de varias maneras, una de las cuales es mediante el uso de constructores con polimorfismo.

Un constructor es un método especial que se utiliza para inicializar los objetos de una clase. En ***Java***, una clase puede tener varios constructores, cada uno con diferentes parámetros. Cuando se utiliza el polimorfismo con constructores, se pueden crear objetos de diferentes clases utilizando el mismo constructor, pues el polimorfismo permite a una clase la capacidad para tener diferentes constructores con diferentes parámetros.

Para implementar constructores con polimorfismo en ***Java***, se deben seguir los siguientes pasos: utilizar una clase padre llamada “Mascota”, y dos clases hijas llamadas “Perro” y “Gato” respectivamente, como lo muestra la figura siguiente:

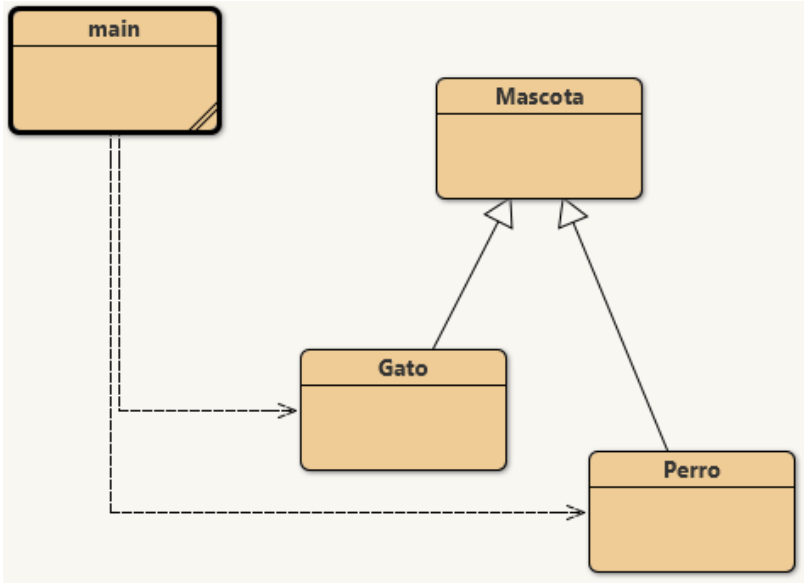


Figura 1. Diagrama de clases

El código correspondiente, se muestra a continuación:

Paso 1. Crear una clase padre con un constructor que tome los parámetros necesarios para inicializar los objetos de la clase.

```
public class Mascota
{
   private String nombre;

   public Mascota (String nombre) {
        this.nombre = nombre;
}

public String getNombre() {

        return nombre;
}
}
```

Figura 2. Clase Mascota

Paso 2. Crear una o más clases hijas que heredan de la clase padre y que tengan sus propios constructores.

Paso 3. En las clases hijas, llamar al constructor de la clase padre utilizando la palabra reservada “super”.

```
public class Perro extends Mascota {
    private String raza;

    public Perro(String nombre, String raza) {
        super(nombre);
        this.raza = raza;
    }

    public String getRaza() {
        return raza;
    }
}
```

Figura 3. Clase hija Perro

```java
public class Gato extends Mascota {
    private String esCazador;

    public Gato(String nombre, String esCaz
        super(nombre);
        this.esCazador = esCazador;
    }

    public String esCazador() {
    return esCazador;
    }
}
```

Figura 4. Clase hija Gato

Paso 4. Crear objetos de las clases hijas utilizando el constructor correspondiente. En el método ***main()***, se crean dos objetos de las clases hijas Perro y Gato. Luego, se llama a los métodos ***getNombre()*** y ***getRaza()*** en cada objeto, lo que demuestra el polimorfismo.

```java
public class main
{
    public static void main(String[] args) {
        Perro mascota1 = new Perro("Fido", "Labrador");
        Gato mascota2 = new Gato("Garfield", "si");
        System.out.println(mascota1.getNombre() + " es un perro de raza " +  mascota1.getRaza());
        System.out.println(mascota2.getNombre() + " es un gato que " + mascota2.esCazador() + " es cazador");

    }
}
```

Figura 5. ***Main*** del programa

Paso 5. Ejecutar el programa y visualizar los resultados.

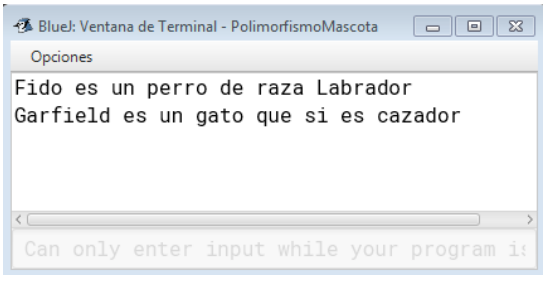


Figura 6. Ejecución del programa

El **polimorfismo,** al ser una característica de la programación orientada a objetos, que permite que un mismo método se comporte de diferentes maneras según el objeto que lo

utilice, además de realizarse en métodos constructores, se puede implementar en cualquier otro método, en Java, esto se logra con **herencia** y **sobrescritura de métodos**.

Para esto se recomienda el uso de ***@Override*** que es una **anotación** en Java que se utiliza para indicar que un método en una subclase (clase hija) está **sobrescribiendo (*overriding*)** un método de su superclase (clase padre).

Esta anotación ayuda a evitar errores y hace que el código sea más claro y fácil de mantener, se considera una buena práctica de programación, **evita errores de escritura**: si el método no coincide con el de la superclase (por ejemplo, por un error en el nombre o los parámetros), el compilador generará un error; **mejora la legibilidad**: indica claramente que un método está sobrescribiendo otro; **asegura un comportamiento correcto**: evita que accidentalmente creemos un nuevo método en lugar de sobrescribir uno existente.

Reglas importantes sobre ***@Override***

1. El método debe tener el mismo nombre que en la superclase.
2. Debe tener los mismos parámetros (mismo número y tipo de argumentos).
3. El modificador de acceso no puede ser más restrictivo (por ejemplo, si en la superclase es ***public***, en la subclase no puede ser ***private***).
4. Solo se usa en herencia, no en métodos nuevos de una clase.

A continuación, se muestra el mismo ejemplo anterior, pero ahora agregando el método:

```
public void hacerSonido() {
        System.out.println("Esta mascota hace un sonido...");
    }
```

Con esto, se creará en la clase padre Mascota y se sobreescribirá en las clases Perro y Gato, comportándose de diferentes maneras según el objeto que lo utilice.

El código completo respecto al Polimorfismo con ***@Overrride***, se muestra en la figura siguiente:

```
public class Mascota
{
   private String nombre;

   public Mascota (String nombre) {
        this.nombre = nombre;

}

public String getNombre() {

        return nombre;

}
public void hacerSonido() {
        System.out.println("Esta mascota hace un sonido...");
    }
}
```

```
public class Perro extends Mascota {
    private String raza;

    public Perro(String nombre, String raza) {
         super(nombre);
        this.raza = raza;
    }

    public String getRaza() {
        return raza;
    }

    @Override
    public void hacerSonido() {
        System.out.println("El perro hace 'Guau'...");
    }
}
```

```java
public class Gato extends Mascota {
    private String esCazador;

    public Gato(String nombre, String esCazador) {
    super(nombre);
          this.esCazador = esCazador;
    }

    public String esCazador() {
    return esCazador;
    }
@Override
public void hacerSonido() {
        System.out.println("El gato hace 'miau'...");
    }
}
```

```java
public class main
{
   public static void main(String[] args) {
     Perro mascota1 = new Perro("Fido", "Labrador");
     Gato mascota2 = new Gato("Garfield", "si");
     System.out.println(mascota1.getNombre() + " es un perro de raza " +  mascota1.getRaza());
     System.out.println("¿Que sonido hace el Perro?");
     mascota1.hacerSonido();
     System.out.println(mascota2.getNombre() + " es un gato que " + mascota2.esCazador() + " es cazador");
     System.out.println("¿Que sonido hace el Gato?");
     mascota2.hacerSonido();
   }
}
```

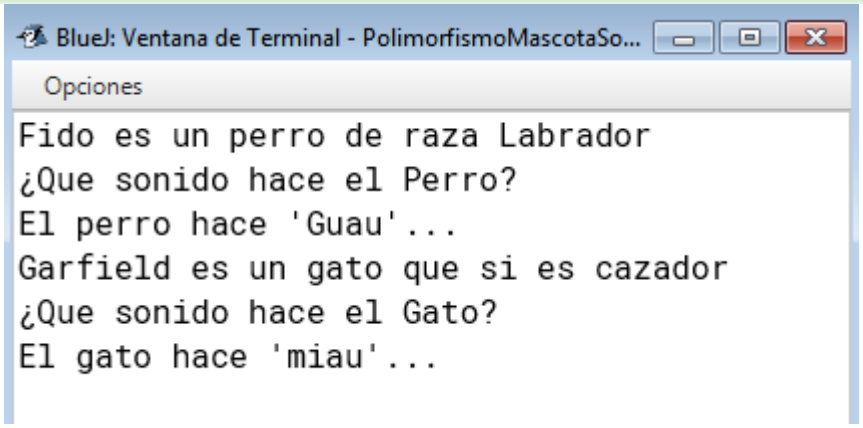


Figura 7. Programa Polimorfismo con ***@Overrride***.

ÍNDICE

3.1.2 Autoevaluación del aprendizaje:

Instrucciones: Lee con atención las preguntas y elige el inciso que responda a la respuesta correcta.

1. ¿Qué es el polimorfismo en Java?

   A) La capacidad de una clase para tener diferentes constructores en un mismo objeto.
   B) La capacidad de una clase para tener diferentes usos de un mismo método.
   C) La capacidad de una clase para tener diferentes variables en un mismo método.
   D) La capacidad de una clase para tener diferentes atributos en un mismo objeto.

2. En ***Java***, un constructor es un método que se debe llamar cuando se…

   A) destruye un objeto.
   B) crea un objeto.
   C) se llama a otro método.
   D) se llama a un método estático.

3. ¿Qué es el polimorfismo de constructores en ***Java*** ?

   A) La capacidad de una clase para tener diferentes constructores con el mismo nombre.
   B) La capacidad de una clase para tener diferentes constructores con diferentes modificadores de acceso.
   C) La capacidad de una clase para tener diferentes constructores con diferentes tipos de retorno.
   D) La capacidad de una clase para tener diferentes constructores con diferentes parámetros.

4. Se utiliza para indicar que un método en una subclase (clase hija) está sobrescribiendo un método de su superclase (clase padre).

   A) ***extends***
   B) ***super***
   C) ***@Override***
   D) ***println();***

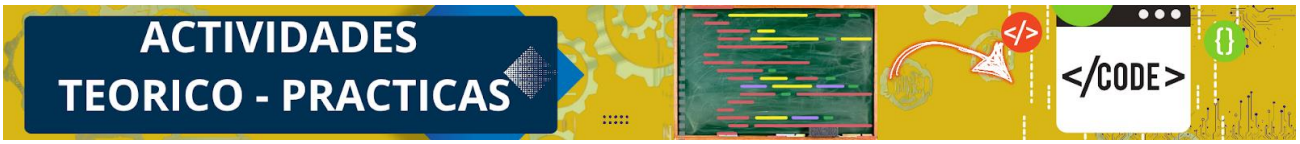
ÍNDICE

APRENDIZAJE 3.2

Comprende la colaboración de Clases para la resolución de problemas.

Temática:

Interacción y comunicación entre Clases.



ÍNDICE

3.2.1. Actividades de aprendizaje teórico–prácticas.

La colaboración de clases en ***Java*** es una técnica fundamental en la programación orientada a objetos (***POO***) que permite la interacción entre diferentes clases para resolver problemas de manera estructurada, modular y reutilizando el código. Esta idea se puede entender como la forma en que diferentes elementos de un programa trabajan juntos para lograr un objetivo común.

Elementos que se deben considerar en la Colaboración de clases: atributos y métodos.

Las clases pueden colaborar de diferentes maneras:

- Asociación: Una clase utiliza los servicios proporcionados por otra clase. Por ejemplo, un dueño de una mascota usa los servicios de una veterinaria.
- Composición: Una clase contiene instancias de otras clases como parte de su estructura. Por ejemplo, un automóvil tiene un motor y ruedas como componentes.
- Agregación: Una clase contiene una colección de otras clases. Por ejemplo, una universidad tiene uno o varios estudiantes.

Cuando desarrollamos un programa en ***Java***, no escribimos todo en una sola clase, sino que dividimos el código en varias clases con responsabilidades específicas. Estas clases interactúan entre sí mediante la creación de objetos, la llamada a métodos y el intercambio de datos.

En el siguiente problema se desea implementar el código para mostrar la relación Dueño-Mascota, donde la finalidad del código es mostrar el nombre de un Dueño asociado con el nombre de una Mascota, haciendo uso de la comunicación entre clases, en especial la Agregación, donde una clase contiene una colección de otras clases. En este caso un dueño tiene una mascota.

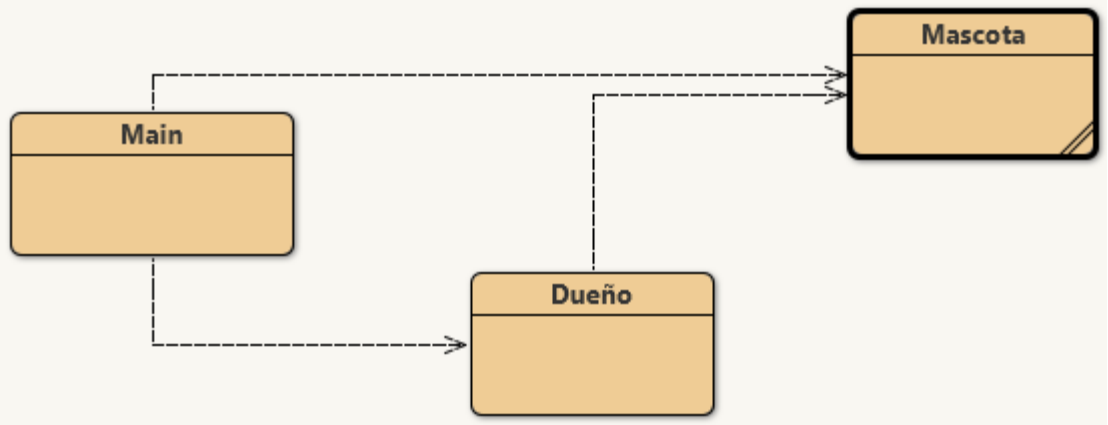


Figura 8. Diagrama de Clases

Paso 1. Diseño de Clases Relacionadas a Mascotas: crear dos clases: Mascota y Dueño.

Paso 2. Definir atributos relevantes para cada clase (por ejemplo, nombre de la mascota, nombre del dueño, etc.)

```java
public class Mascota {
    private String nombre;

    public Mascota(String nombre) {
        this.nombre = nombre;
    }

    public String getNombre() {
        return nombre;
    }
}
```

Figura 9. Clase Mascota

```java
public class Dueño {
    private String nombre;
    private Mascota mascota;

    public Dueño(String nombre, Mascota mascota) {
        this.nombre = nombre;
        this.mascota = mascota;
    }

    public void presentarMascota() {
        System.out.println("Soy " + nombre + " y mi mascota es " + mascota.getNombre());
    }
}
```

Figura 10. Clase Dueño

Paso 3. Colaboración entre Clases: las clases Mascota y Dueño colaboran de manera de **Agregación** donde un dueño puede tener una mascota, una mascota pertenece a un único dueño.

La colaboración de **Agregación** se da cuando utilizamos la clase Mascota como un tipo de dato que puede utilizarse como parte de los atributos de la clase principal que en este caso es Dueño.

```java
// Clase principal con el método main
public class Main {
    public static void main(String[] args) {
        // Creación del objeto perro
        Mascota perro = new Mascota("Fido");
        Dueño persona = new Dueño("Juan", perro);
        persona.presentarMascota();
        }
}
```

Figura 11. Main del programa

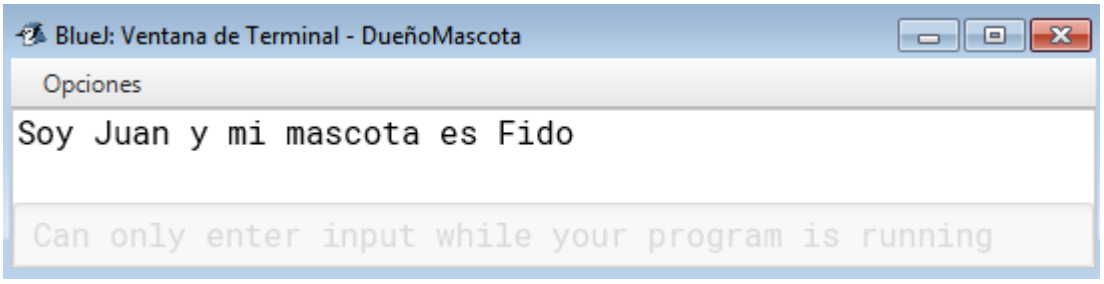


Figura 12. Ejecución del programa

Para ejemplificar el tipo de **Colaboración por Asociación** donde una clase utiliza los servicios proporcionados por otra clase, en este caso la clase Dueño usa los servicios de la clase Veterinaria mediante el método **"atenderMascota()"**. Resolveremos un problema que permita la **gestión de una veterinaria**, podríamos tener estas clases:

1. Clase Mascota: representa a una mascota con nombre, especie y estado de salud.
2. Clase Dueño: representa a una persona que tiene una mascota y puede llevarla a la veterinaria.
3. Clase Veterinaria: simula una clínica veterinaria donde se pueden atender mascotas.

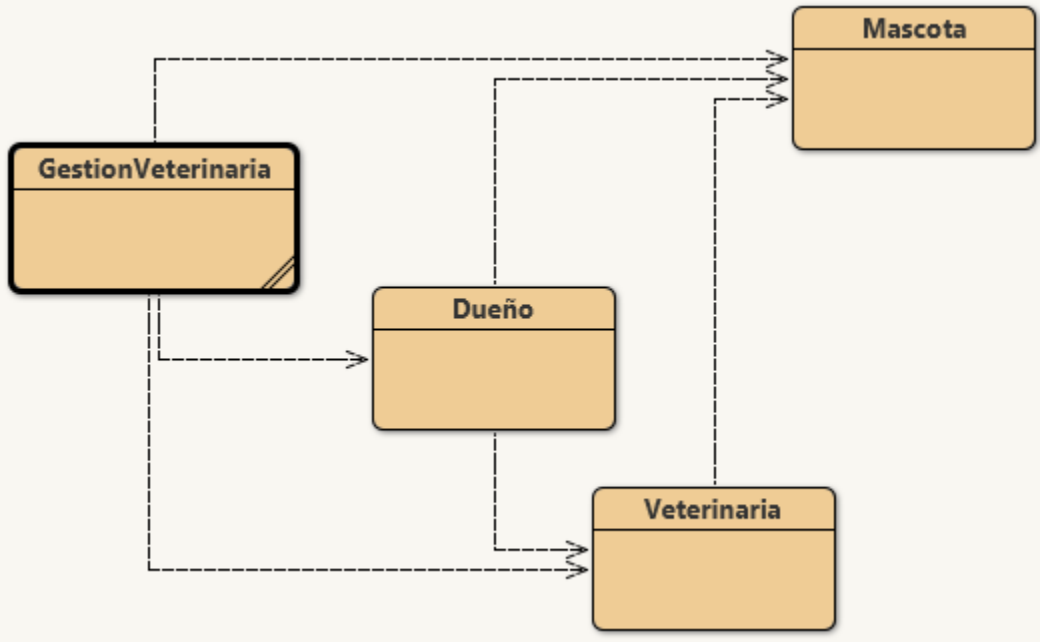


Figura 13. Diagrama de clases

Paso 1. Diseño de Clases Relacionadas a Mascotas: crear dos clases: Mascota, Dueño, Veterinaria.

Paso 2. Definir atributos y métodos para cada clase.

```java
// Clase que representa una mascota
class Mascota {
    private String nombre;
    private String especie;
    private boolean enferma;

    // Constructor de la mascota
    public Mascota(String nombre, String especie) {
        this.nombre = nombre;
        this.especie = especie;
        this.enferma = false; // Por defecto, la mascota está sana
    }

    // Obtener el nombre de la mascota
    public String getNombre() {
        return nombre;
    }

    // Saber si la mascota está enferma
    public boolean estaEnferma() {
        return enferma;
    }

    // Método para enfermar a la mascota (simulación)
    public void enfermar() {
        enferma = true;
        System.out.println(":( " + nombre + " se ha enfermado.");
    }

    // Método para curar a la mascota
    public void curar() {
        enferma = false;
        System.out.println(":) " + nombre + " ha sido curado.");
    }
}
```

Figura 14. Clase Mascota

En el método **llevarAVeterinario()**, se observa la **Colaboración entre clases por Asociación** al pasar como parámetro la variable veterinaria de tipo clase Veterinaria y posteriormente realizar el llamado al método **atenderMascota(mascota)**.

```
// Clase que representa un dueño de mascota
class Dueño {
    private String nombre;
    private Mascota mascota; // Cada dueño tiene una mascota

    // Constructor del dueño con su mascota
    public Dueño(String nombre, Mascota mascota) {
        this.nombre = nombre;
        this.mascota = mascota;
    }

    // Método para llevar la mascota a la veterinaria si está enferma
    public void llevarAVeterinaria(Veterinaria veterinaria) {
        System.out.println("\n ->" + nombre + " lleva a " + mascota.getNombre() + " a la veterinaria.");
        veterinaria.atenderMascota(mascota);
    }

    // Método para enfermar a la mascota (simulación)
    public void mascotaSeEnferma() {
        mascota.enfermar();
    }
}
```

Figura 15. Clase Dueño

De igual manera, en la clase Veterinaria se observa la **Colaboración entre clases por Asociación** con la clase Mascota**,** al pasar como parámetro la variable mascota de tipo clase Mascota al método **atenderMascota(mascota),** observa que ya en el método se puede utilizar todos los métodos de la clase Mascota, pues esta ha sido pasada como parámetro.

```
// Clase que representa la veterinaria
class Veterinaria {
    private String nombreVeterinaria;

    // Constructor de la veterinaria
    public Veterinaria(String nombreVeterinaria) {
        this.nombreVeterinaria = nombreVeterinaria;
    }

    // Método para atender a la mascota
    public void atenderMascota(Mascota mascota) {
        if (mascota.estaEnferma()) {
            System.out.println("  Atendiendo a " + mascota.getNombre() + " en " + nombreVeterinaria + "...");
            mascota.curar();
        } else {
            System.out.println(":) " + mascota.getNombre() + " está sano y no necesita atención.");
        }
    }
}
```

Figura 16. Clase Veterinaria

```java
// Clase principal con el método main
public class GestionVeterinaria {
    public static void main(String[] args) {
        // Creación de una veterinaria
        Veterinaria vet = new Veterinaria("Veterinaria Patitas Felices");

        // Creación de mascotas
        Mascota mascota1 = new Mascota("Luna", "Perro");
        Mascota mascota2 = new Mascota("Michi", "Gato");

        // Creación de dueños con sus mascotas
        Dueño dueño1 = new Dueño("Carlos", mascota1);
        Dueño dueño2 = new Dueño("Ana", mascota2);

        // Simulación de enfermedad y visita a la veterinaria
        // Unicode \uD83D\uDC3E para el emoji de huellitas 🐾
        System.out.println("\uD83D\uDC3E Bienvenido a la Veterinaria Patitas Felices \n");
        dueño1.mascotaSeEnferma(); // Luna se enferma
        dueño1.llevarAVeterinaria(vet); // Carlos lleva a Luna a la veterinaria
        dueño2.llevarAVeterinaria(vet); // Ana lleva a Michi, que está sano
    }
}
```

Figura 17. Clase Main-Gestión de Veterinaria

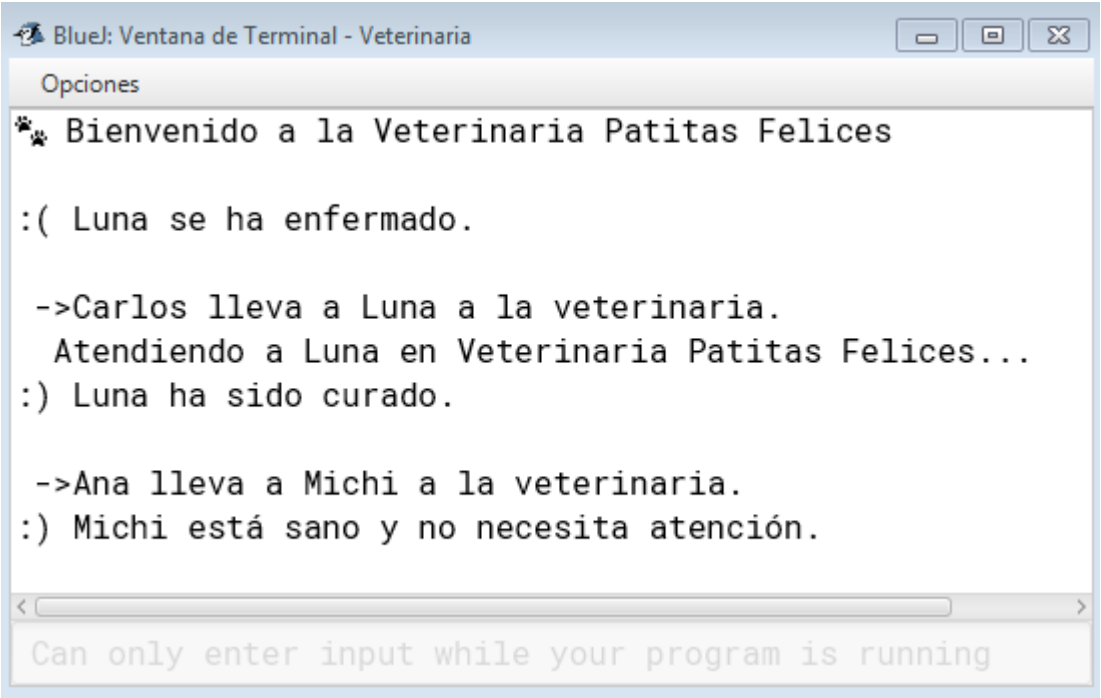


Figura 18. Ejecución del programa

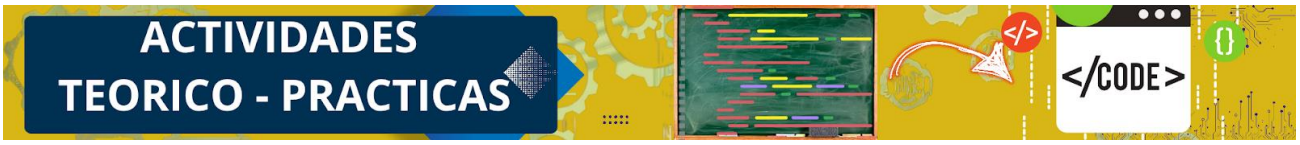
ÍNDICE

APRENDIZAJE 3.3

Desarrolla programas que involucren la colaboración de Clases.

Temática:

Interacción y comunicación entre Clases.



ÍNDICE

3.3.1. Actividades de aprendizaje teórico–prácticas.

La colaboración de clase en el desarrollo de programas puede ser de gran utilidad principalmente para:

- **Organización:** Separar la lógica en diferentes clases hace que el código sea más claro y fácil de entender.
- **Reutilización:** Se pueden crear más objetos sin reescribir código.
- **Mantenimiento:** Si necesitamos modificar la lógica, solo editamos un mínimo de código

Por ejemplo, en un **sistema de gestión de biblioteca**, podríamos tener estas clases:

1. **Libro** (representa un libro con título, autor, etc.).
2. **Usuario** (representa a una persona que toma libros prestados).
3. **Biblioteca** (gestiona los libros y los préstamos).

Cada clase tiene sus propias propiedades y métodos, pero trabajan juntas para resolver el problema de gestionar una biblioteca.

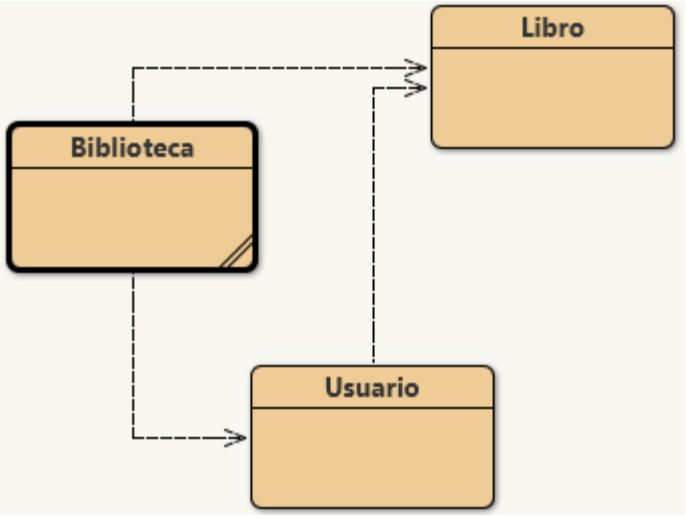


Figura 19. Diagrama de clases

Explicación paso a paso

1. Clase Libro
    - Representa un libro con título y estado de disponibilidad.
    - Métodos: ***prestar()***, ***devolver()***, ***isDisponible()***, ***getTitulo()***.
2. Clase Usuario
    - Representa a una persona que toma libros prestados.
    - Métodos: ***pedirLibro()***, ***devolverLibro()***.
3. Clase Biblioteca (con ***main***)
    - Crea libros y usuarios.
    - Simula préstamos y devoluciones con mensajes en la consola.

```java
// Clase que representa un libro en la biblioteca
class Libro {
    private String titulo;   // Título del libro
    private boolean disponible; // Estado del libro: true si está disponible, false si está prestado

    // Constructor para inicializar el libro con un título
    public Libro(String titulo) {
        this.titulo = titulo;
        this.disponible = true; // Por defecto, un libro está disponible al crearse
    }

    // Método para obtener el título del libro
    public String getTitulo() {
        return titulo;
    }

    // Método para verificar si el libro está disponible
    public boolean isDisponible() {
        return disponible;
    }

    // Método para prestar un libro si está disponible
    public void prestar() {
        if (disponible) {  // Si el libro está disponible, se presta
            disponible = false;
            System.out.println("📖 Libro prestado con éxito: " + titulo);
        } else {  // Si no está disponible, se informa al usuario
            System.out.println("⚠️  El libro '" + titulo + "' se encuentra en prestamo.");
        }
    }

    // Método para devolver un libro y hacerlo disponible nuevamente
    public void devolver() {
        disponible = true;
        System.out.println("✅ Libro devuelto: " + titulo);
    }
}
```

Figura 20. Clase Libro

```java
// Clase que representa un usuario de la biblioteca
class Usuario {
    private String nombre; // Nombre del usuario

    // Constructor para inicializar el usuario con un nombre
    public Usuario(String nombre) {
        this.nombre = nombre;
    }

    // Método para que el usuario pida prestado un libro
    public void pedirLibro(Libro libro) {
        System.out.println("\n📢 " + nombre + " solicita en prestamo el libro: " + libro.getTitulo());
        libro.prestar();
    }

    // Método para que el usuario devuelva un libro
    public void devolverLibro(Libro libro) {
        System.out.println("\n📢 " + nombre + " devuelve el libro: " + libro.getTitulo());
        libro.devolver();
    }
}
```

Figura 21. Clase Usuario

```java
// Clase principal que simula el sistema de la biblioteca
public class Biblioteca {
    public static void main(String[] args) {
        // Creación de objetos de libros
        Libro libro1 = new Libro("El Principito");
        Libro libro2 = new Libro("Cien años de soledad");

        // Creación de objetos de usuarios
        Usuario usuario1 = new Usuario("Carlos");
        Usuario usuario2 = new Usuario("María");

        // Prueba del sistema
        System.out.println("\n--- 🏛 PRÉSTAMOS DE LIBROS ---");
        usuario1.pedirLibro(libro1);  // Carlos pide "El Principito"
        usuario2.pedirLibro(libro2);  // María pide "Cien años de soledad"
        usuario1.pedirLibro(libro1);  // Carlos intenta pedirlo de nuevo un libro ya prestado

        System.out.println("\n---   DEVOLUCIÓN DE LIBROS ---");
        usuario1.devolverLibro(libro1); // Carlos devuelve "El Principito"
        usuario2.devolverLibro(libro2); // María devuelve "Cien años de soledad"

        System.out.println("\n--- ♻  NUEVO PRÉSTAMO ---");
        usuario1.pedirLibro(libro1);  // Ahora puede volver a pedirlo pues ya fue devuelto
    }
}
```

Figura 22. Clase Biblioteca-***Main***

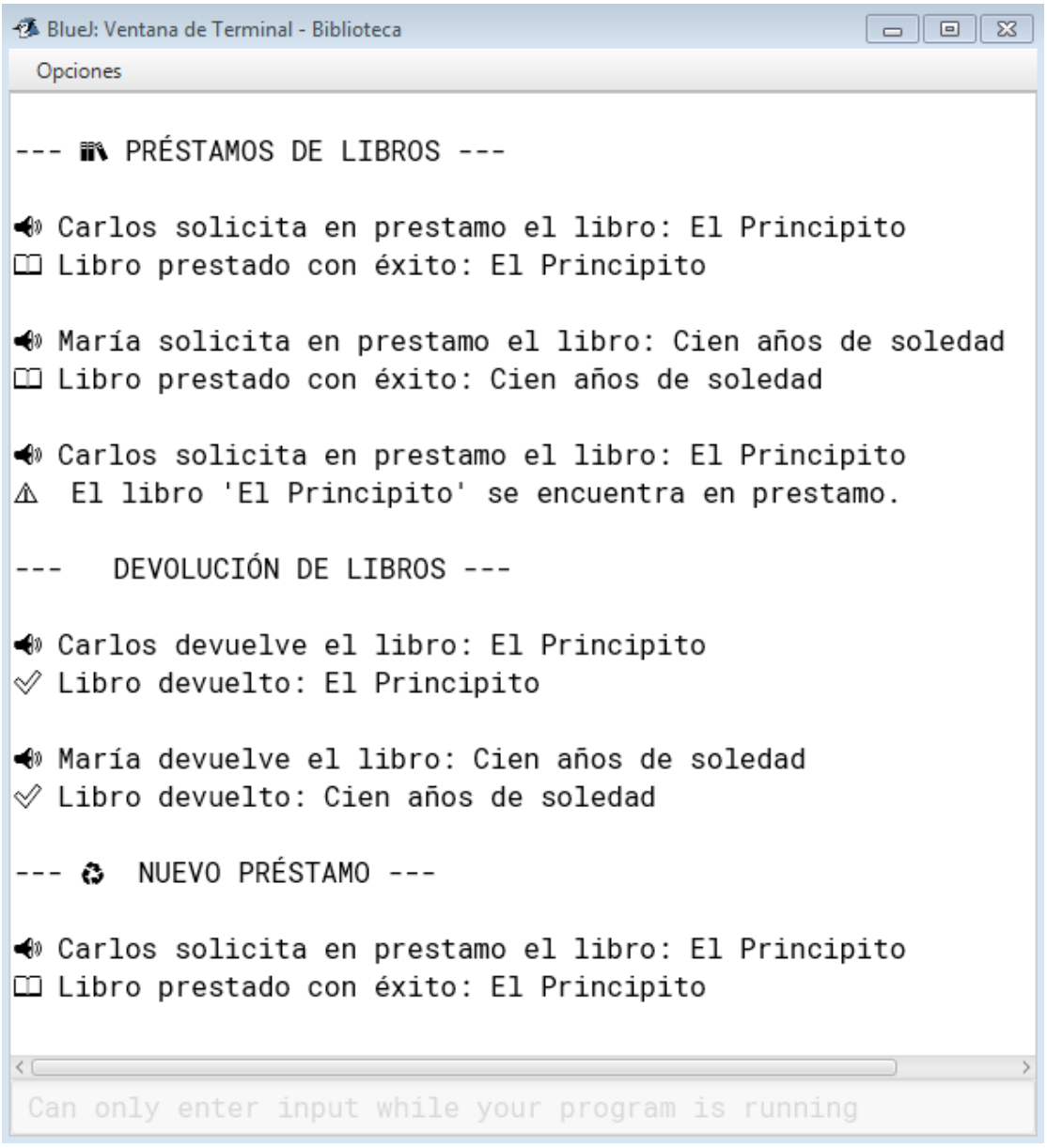


Figura 23. Ejecución del programa

ÍNDICE

3.3.2 Autoevaluación del aprendizaje:

Instrucciones: Relaciona cada segmento de Código con su Descripción correspondiente.

Código	Descripción

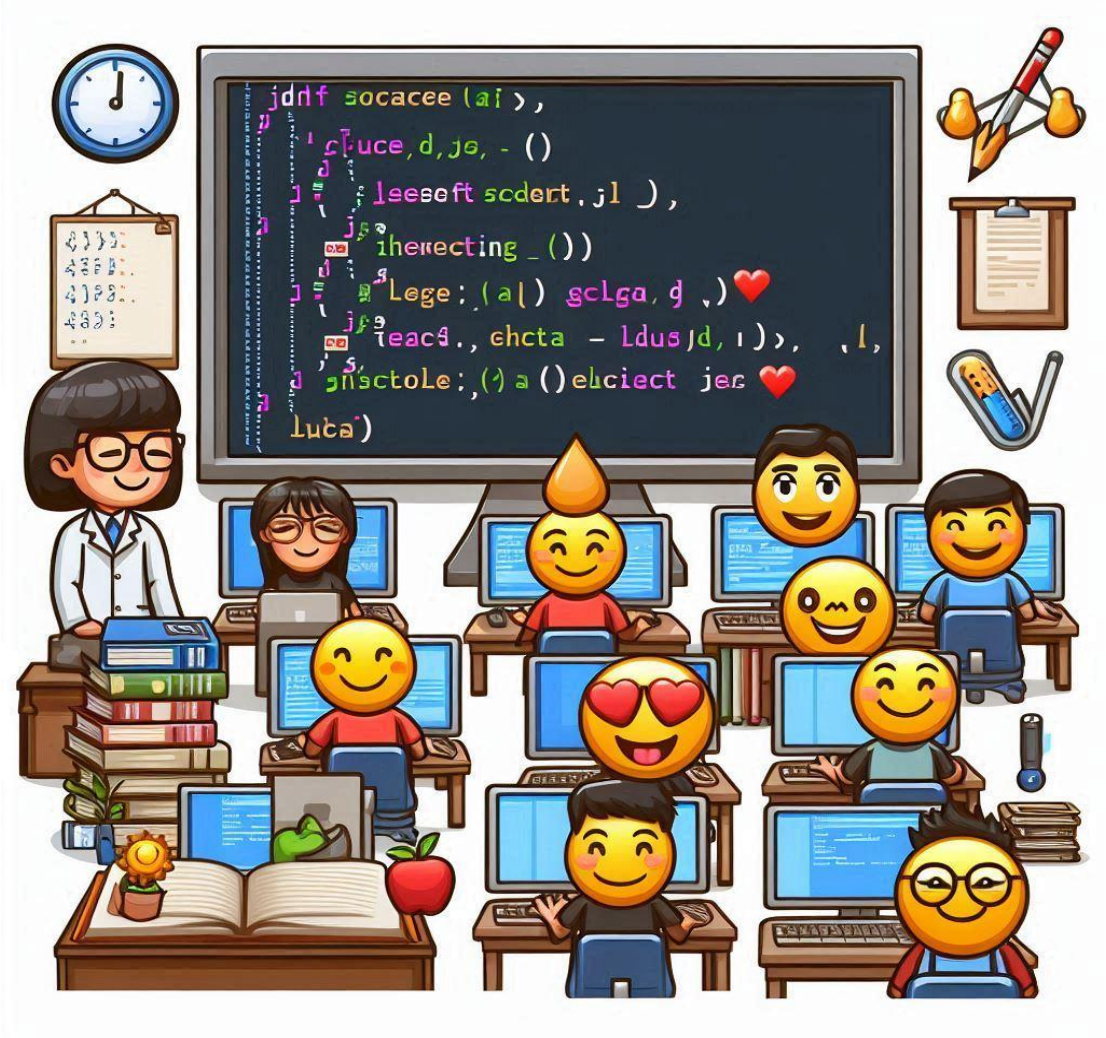
I.
`class Cliente { String nombre; }`
II.
`public void facturar(Tienda t, double venta) { }`
III.
`class Tienda { Cliente cliente; Producto producto; }`
IV.
`public void comprar(Producto p, Cliente c) { }`
V.
`class Producto { double precio; int stock; }`
	a) Método que permite asociar un Producto con un Cliente en una compra.
	b) Clase que almacena los productos disponibles en la tienda.
	c) Clase que gestiona la relación entre Cliente y Producto.
	d) Clase que almacena información del Cliente.
	e) Método que permite asociar un Producto con su stock.
	f) Método que permite asociar las transacciones relacionadas con las ventas de la tienda.

A) I:c, II: a, III: b, IV: e, V: f
B) I:d, II: f, III: c, IV: a, V: b
C) I:c, II: a, III: d, IV: e, V: b
D) I:d, II: f, III: c, IV: a, V: e

ÍNDICE

APRENDIZAJE 3.4

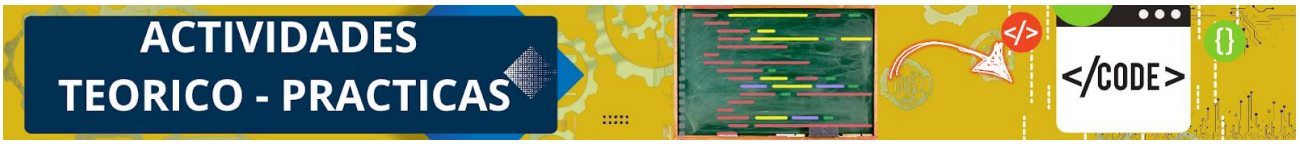
Comprende el concepto de herencia en la resolución de un problema.



Temática:

Herencia:

- Superclase.
- Subclase.
- Ventajas.



ÍNDICE

3.4.1. Actividades de aprendizaje teórico–prácticas.

Las clases modelan el hecho de que el mundo real contiene objetos con propiedades (atributos) y comportamiento (métodos).

> La herencia modela el hecho de que estos objetos tienden a organizarse en jerarquías. Esta jerarquía desde el punto de vista del modelado se denomina relación de generalización o es-un. En programación orientada a objetos, la relación de generalización se denomina herencia. Cada clase derivada hereda las características de la clase base y además cada clase derivada añade sus propias características (Joyanes, 2014).

En resumen, podemos decir que la herencia es una forma de **reutilización de código**, en la que se crea una nueva clase, que permite evitar la duplicidad de código, permitiendo que aquello que es común a varias clases se escriba sólo una vez logrando así reutilizar el código.

La siguiente figura muestra un ejemplo de herencia donde se puede apreciar la jerarquía de la clase padre o superclase que en este caso es **"Persona"** y de esta dependen dos clases hijas o subclases que son **"Estudiante"** y **"Profesor".**

Figura 24. Jerarquía de herencia.

La forma de representar la herencia es identificar la clase padre y la(s) clase(s) hija(s), en la clase padre deben de estar todos los atributos generales y en la(s) clase(s) hija(s) los atributos propios que son únicos de cada una, en la siguiente figura se muestra el ejemplo

de la clase Persona cuyos atributos son: nombre, apellido paterno, apellido materno y edad, estos atributos se heredarán a las clases hijas Estudiante y Profesor, logrando la reutilización de atributos por lo que las clase hijas solo necesitaran sus atributos que las hace únicas, en el caso de la clase Estudiante sus atributos únicos serán: número de cuenta y promedio, y de la clase Profesor sus atributos únicos serán: número de empleado y sueldo, logrando así la correlación necesaria en la herencia, donde podamos decir que un **estudiante <u>es-una</u> persona** y un **profesor <u>es-una</u> persona.**

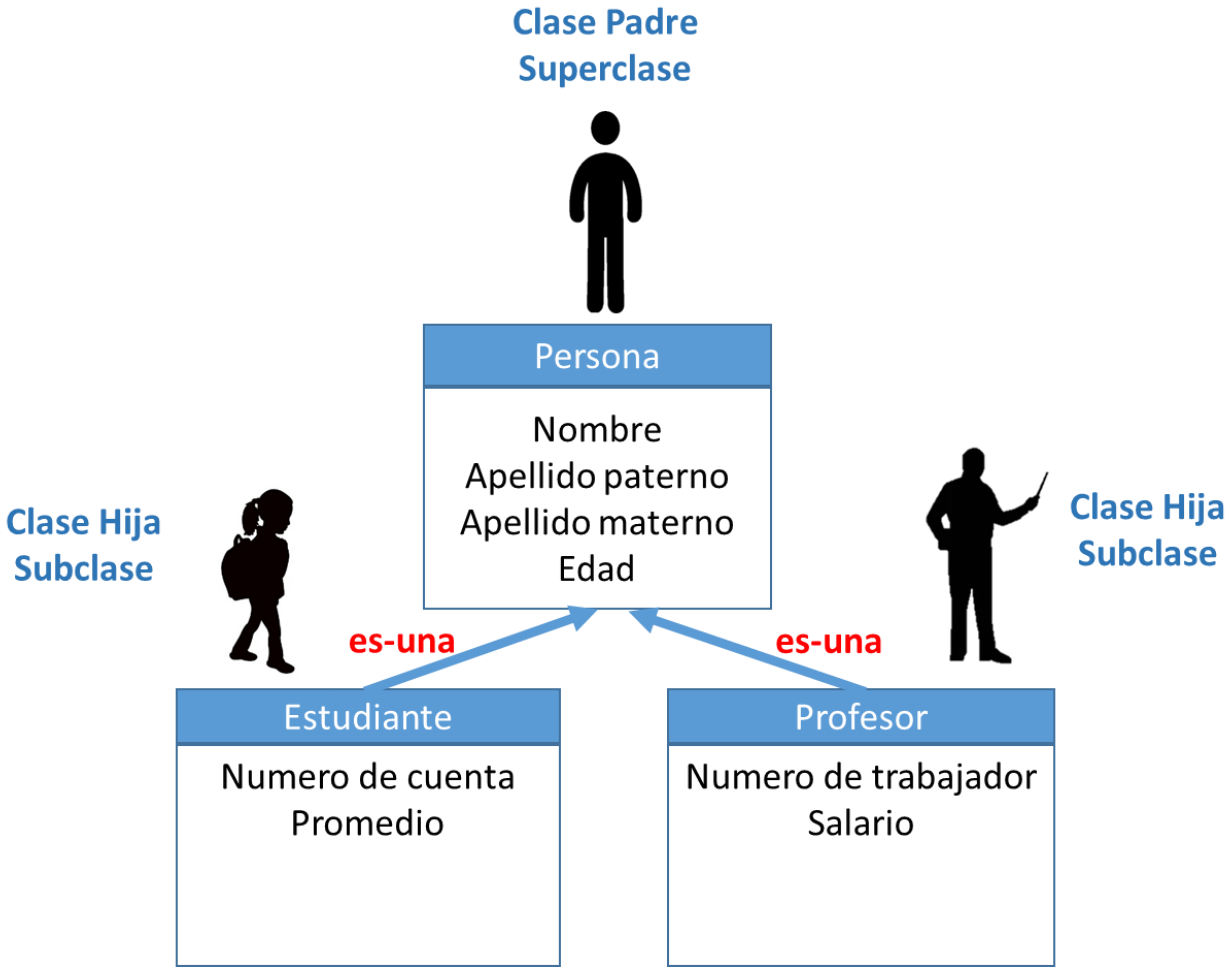
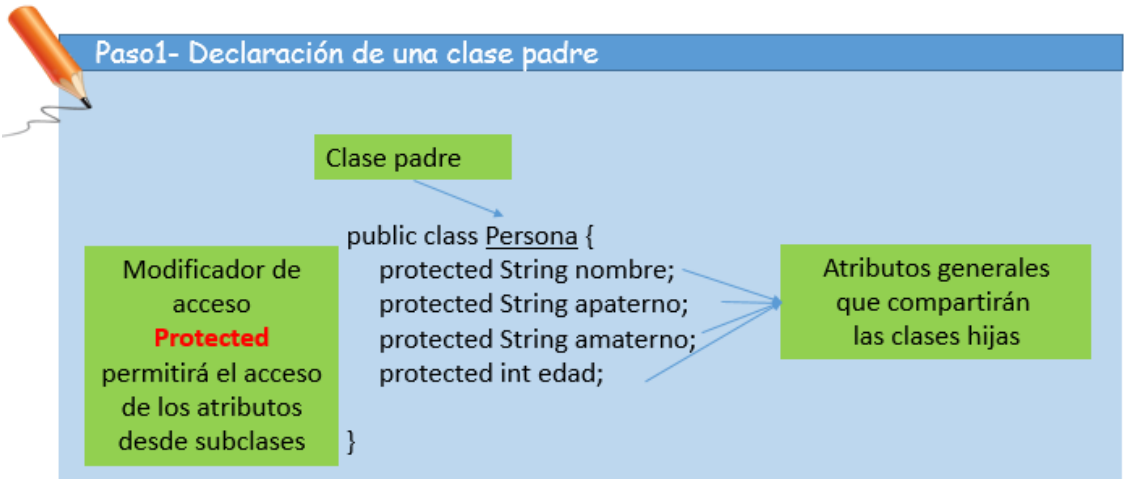
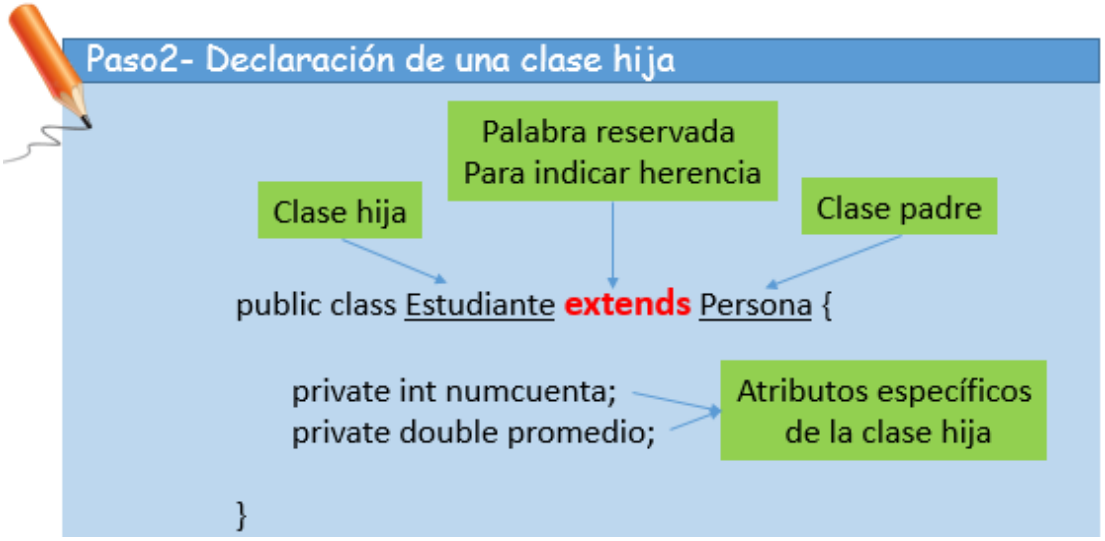
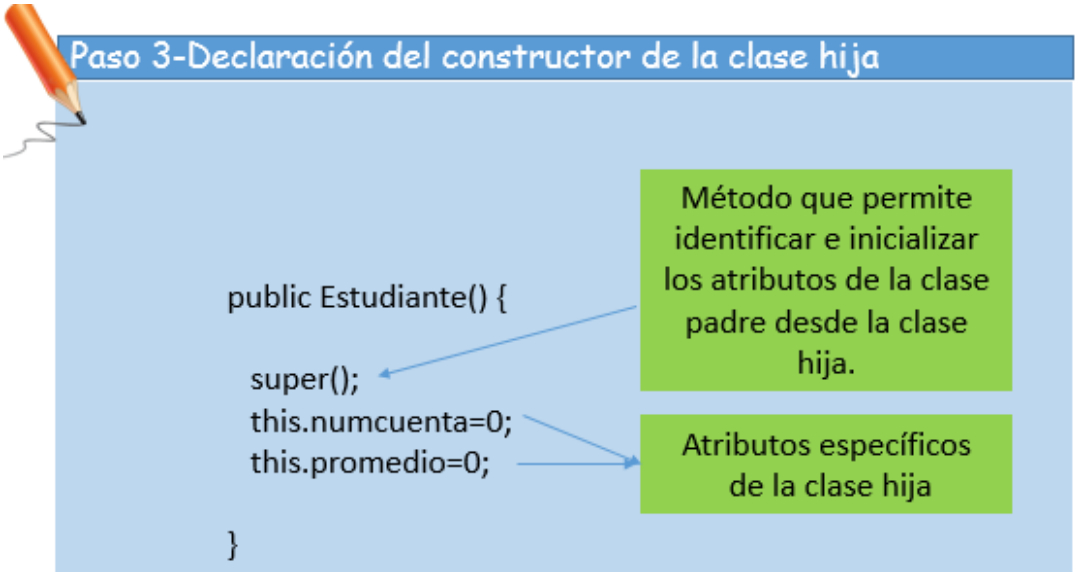


Figura 25. Superclase y subclase.

Los pasos que se deben seguir para codificar en ***Java*** el esquema anterior serían:







ÍNDICE

3.4.2 Autoevaluación del aprendizaje:

Instrucciones: Lee con atención la columna de Definición y realiza la relación correspondiente con su respuesta correcta, según la Palabra reservada o Término al que corresponda.

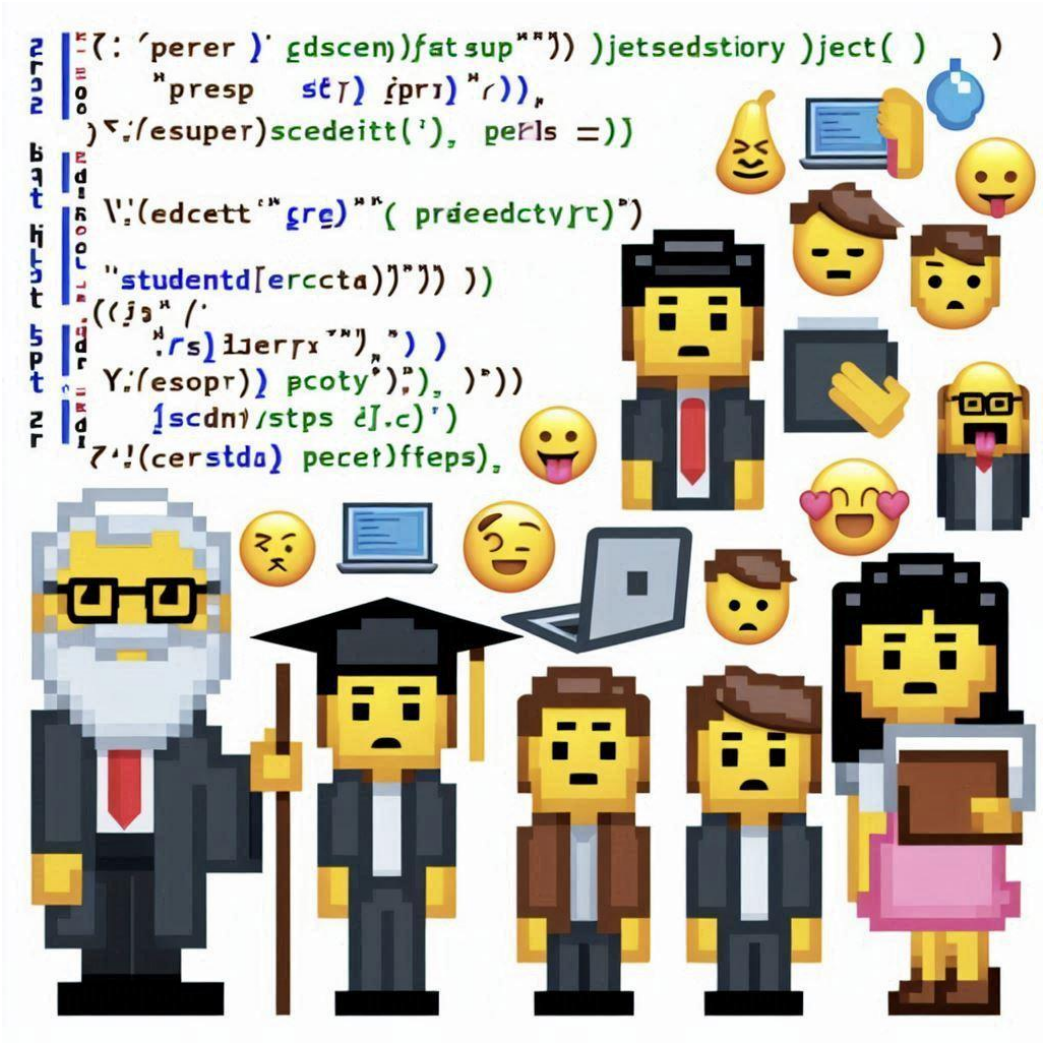
Definición	Palabra reservada/Término
I. Nombre como se le conoce a una clase hija en programación orientada a objetos. II. Nombre como se le conoce a una clase padre en programación orientada a objetos. III. Modificador de acceso que permite el acceso de los atributos de la clase padre desde las clases hijas. IV. Palabra reservada para indicar herencia en la creación de una clase. V. Método que permite identificar e inicializar los atributos de la clase padre desde la clase hija.	a) ***protected*** b) ***new*** c) ***extends*** d) subclase e) ***super***(); f) superclase

A) I:d, II: f, III: c, IV: a, V: e
B) I:f, II: d, III: a, IV: c, V: b
C) I:d, II: f, III: a, IV: c, V: e
D) I:f, II: d, III: b, IV: c, V: a

ÍNDICE

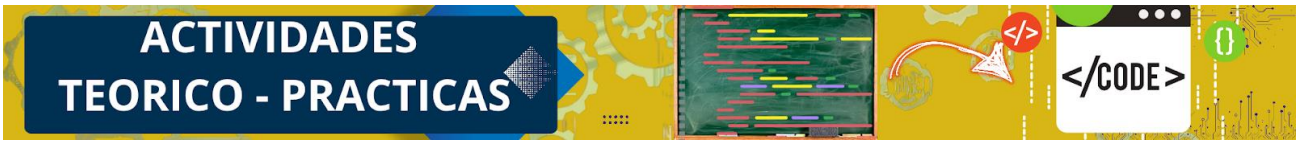
APRENDIZAJE 3.5

Desarrolla programas que involucren la herencia de clases.



Temática:

Implementación de la herencia de Clases.



ÍNDICE

3.5.1. Actividades de aprendizaje teórico–prácticas.

Retomando el ejemplo de la herencia vista en el aprendizaje anterior, donde a partir de una clase padre llamada Persona, se crea la clase hija llamada Estudiante, y una segunda clase hija llamada Profesor, se implementará el siguiente programa en java cuyo propósito es permitir a un alumno la Gestión de la calificación de cada una de sus materias, explicación del Código:

- Persona: Es la superclase con los atributos comunes (nombre, edad).
- Estudiante: Hereda de Persona, tiene una lista de Materias.
- Profesor: Hereda de Persona, enseña una materia específica.
- Materia: Representa la relación entre Profesor y Estudiante, guardando la calificación.
- Ciclo ***`do-while`***: Permite ingresar múltiples materias y profesores.
- Uso de ***`ArrayList`***: Para almacenar todas las materias y calificaciones del estudiante.

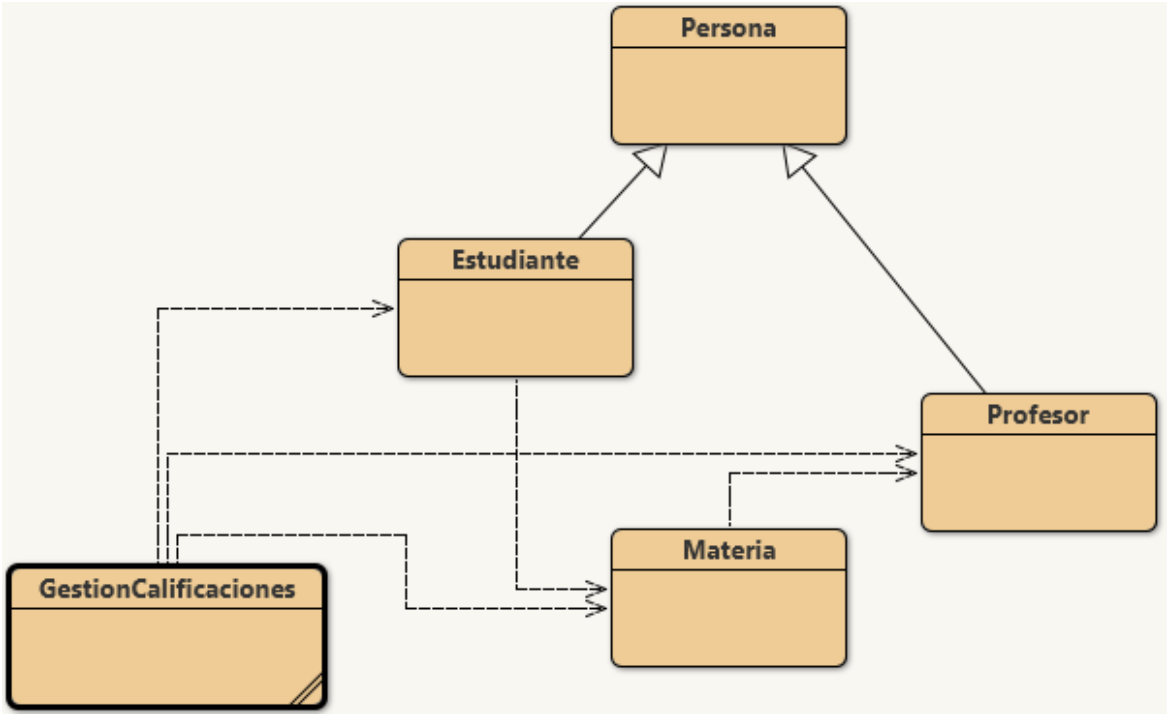


Figura 26. Diagrama de clases

```java
import java.util.ArrayList;
import java.util.Scanner;

// Superclase Persona (Clase Padre)
class Persona {
    protected String nombre;  // Atributo protegido (accesible en clases hijas)

    // Constructor de Persona
    public Persona(String nombre) {
        this.nombre = nombre;
    }

    // Método para mostrar información básica de la Persona
    public void mostrarInfo() {
        System.out.println("👤 Nombre: " + nombre);
    }
}
```

Figura 27. Clase Persona

```java
import java.util.ArrayList;
// Subclase Estudiante (Hereda de Persona)
class Estudiante extends Persona {
    private ArrayList<Materia> materias; // Lista de materias con sus calificaciones

    // Constructor de Estudiante
    public Estudiante(String nombre) {
        super(nombre);  // Llamamos al constructor de Persona
        this.materias = new ArrayList<>();
    }

    // Método para agregar una nueva materia y calificación
    public void agregarMateria(Materia materia) {
        materias.add(materia);
    }

    // Método para mostrar todas las materias y calificaciones del estudiante
    public void mostrarMaterias() {
        System.out.println("\n📚 Materias y calificaciones de " + nombre + ":");
        for (Materia materia : materias) {
            System.out.println("📝 " + materia.getNombre() + " - Profesor: " + materia.getProfesor().getNombre()
                               + " | Calificación: " + materia.getCalificacion());
        }
    }
}
```

Figura 28. Clase Estudiante

```java
// Subclase Profesor (Hereda de Persona)
class Profesor extends Persona {
    private String materia;  // Materia que enseña el profesor

    // Constructor de Profesor
    public Profesor(String nombre, String materia) {
        super(nombre);
        this.materia = materia;
    }

    // Método para obtener el nombre del profesor
    public String getNombre() {
        return nombre;
    }

    // Método para obtener la materia que enseña
    public String getMateria() {
        return materia;
    }
}
```

Figura 29. Clase Profesor

```java
// Clase Materia que une Estudiante con Profesor y calificación
class Materia {
    private String nombre;
    private Profesor profesor;
    private double calificacion;

    // Constructor de Materia
    public Materia(String nombre, Profesor profesor, double calificacion) {
        this.nombre = nombre;
        this.profesor = profesor;
        this.calificacion = calificacion;
    }

    // Métodos para obtener la información de la materia
    public String getNombre() {
        return nombre;
    }

    public Profesor getProfesor() {
        return profesor;
    }

    public double getCalificacion() {
        return calificacion;
    }
}
```

Figura 30. Clase Materia

```java
import java.util.ArrayList;
import java.util.Scanner;
// Clase principal que ejecuta el programa
public class GestionCalificaciones {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // ♠ Solicitar datos del estudiante
        System.out.println("📝 Ingresa los datos del estudiante:");
        System.out.print("Nombre: ");
        String nombreEstudiante = scanner.nextLine();

        // Crear el objeto Estudiante
        Estudiante estudiante = new Estudiante(nombreEstudiante);

        // ♠ Ciclo para ingresar varias materias y profesores
        String respuesta;
        do {
            // Solicitar datos del profesor
            System.out.println("\n👨‍🏫 Ingresa los datos del profesor:");
            System.out.print("Nombre: ");
            String nombreProfesor = scanner.nextLine();
            System.out.print("Materia que enseña: ");
            String nombreMateria = scanner.nextLine();

            // Crear el objeto Profesor
            Profesor profesor = new Profesor(nombreProfesor, nombreMateria);

            // Solicitar la calificación de la materia
            System.out.print("Calificación en " + nombreMateria + ": ");
            double calificacion = scanner.nextDouble();
            scanner.nextLine(); // Consumir la línea pendiente

            // Crear la materia y asociarla con el estudiante
            Materia materia = new Materia(nombreMateria, profesor, calificacion);
            estudiante.agregarMateria(materia);

            // Preguntar si quiere ingresar otra materia
            System.out.print("¿Deseas ingresar otra materia? (S/N): ");
            respuesta = scanner.nextLine().toLowerCase(); //toLowerCase() convierte a minúsculas
        } while (respuesta.equals("s"));

        // ♠ Mostrar la información final del estudiante
        System.out.println("\n📌 Información del estudiante:");
        estudiante.mostrarInfo();
        estudiante.mostrarMaterias();

    }
}
```

Figura 31. Clase Gestión de Calificaciones – ***Main***

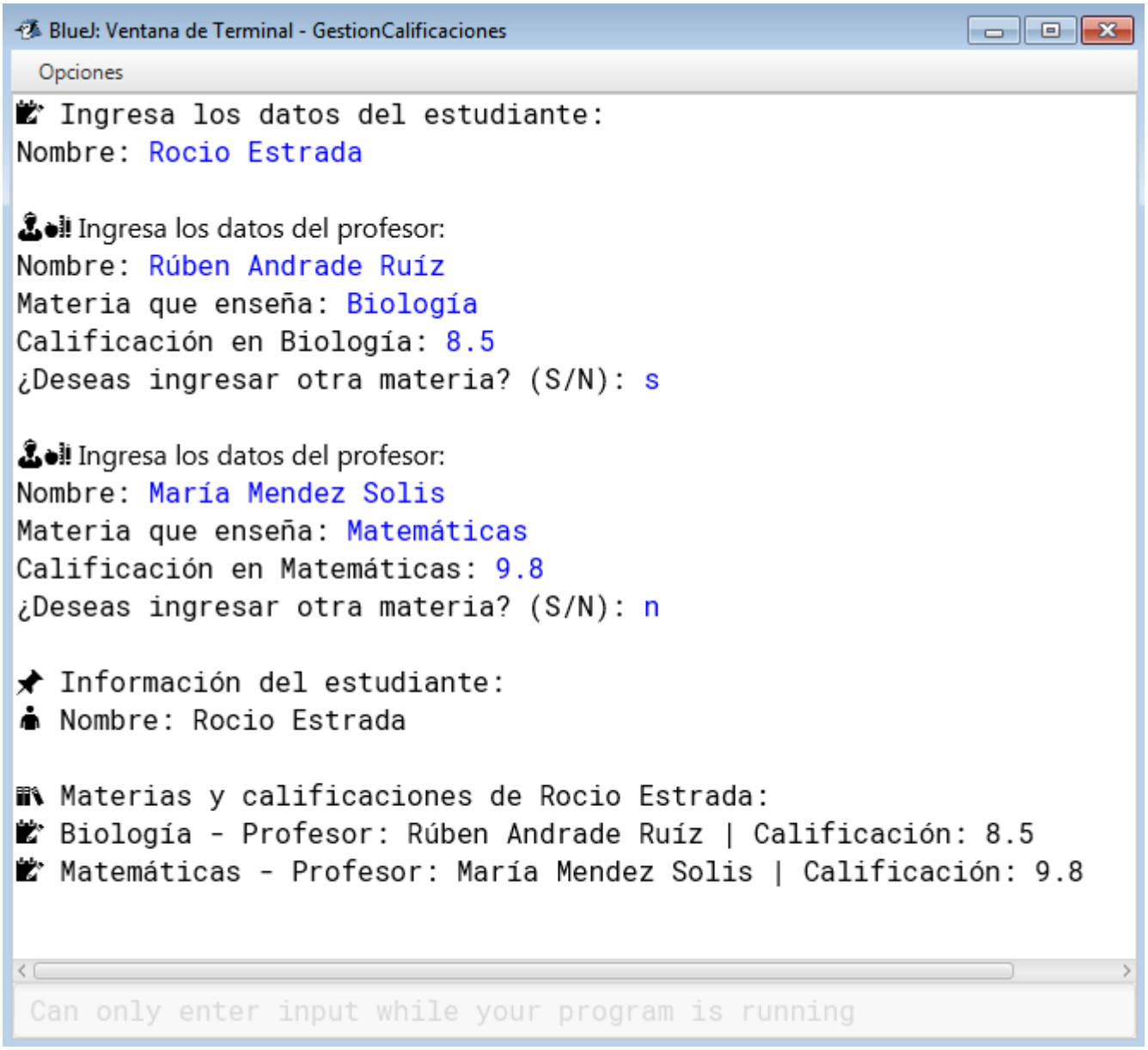


Figura 32. Ejecución del programa

La herencia nos permite evitar código repetido y aprovechar características comunes, como lo siguiente:

- Cada subclase (Estudiante y Profesor) tiene su propia funcionalidad adicional.
- Permite ingresar varios profesores y materias.
- Muestra toda la información al final.
- Usa herencia, colaboración de clases y polimorfismo (permite sobrescribir métodos (mostrarInfo()))

Los programas de esta Unidad utilizan emojis imprimibles generados por ***OpenAi (OpenAi, 2025)***, mismos que, al descargar los programas en la sección "Códigos" de esta guía, estarán disponibles para usarlos. Además, se pueden generar también mediante la sentencia **System.out.println()** utilizando su correspondiente ***Unicode*** en formato ***escape Java***; para esto, proporcionamos la lista de los que se utilizaron en los programas de esta unidad y un ejemplo de cómo usarlos.

Emoji	Descripción	Unicode en formato escape Java
	Persona	\uD83D\uDC64
	Libros	\uD83D\uDCDA
	Nota con lápiz	\uD83D\uDCDD
	Profesor	\uD83D\uDC68\u200D\uD83C\uDFEB
	Tachuela	\uD83D\uDCCC
	Huellas de patas	\uD83D\uDC3E
	Hospital	\uD83C\uDFE5
	Jeringa	\uD83D\uDC89
	Perro	\uD83D\uDC36
	Gato	\uD83D\uDC31
	Médico veterinario	\uD83D\uDC68\u200D\u2695\uFE0F
	Lupa	\uD83D\uDD0D

A continuación, se muestra una ejemplo de código para generar emojis mediante ***Unicode*** en formato ***escape Java***:

```
public class Emojis
{
        public static void main(String[] args) {
            //Unicode en formato escape Java para imprimir emojis
                System.out.println("\uD83D\uDC64 Persona");
                System.out.println("\uD83D\uDCDA Libros");
                System.out.println("\uD83D\uDCDD Nota de Lapiz");
                System.out.println("\uD83D\uDC68\u200D\uD83C\uDFEB Profesor");
                System.out.println("\uD83D\uDCCC Tachuela");
                System.out.println("\uD83D\uDC3E Huellas de patas");
                System.out.println("\uD83C\uDFE5 Hospital");
                System.out.println("\uD83D\uDC89 Jeringa");
                System.out.println("\uD83D\uDC36 Perro");
                System.out.println("\uD83D\uDC31 Gato");
                System.out.println("\uD83D\uDC68\u200D\u2695\uFE0F Médico veterinario");
                System.out.println("\uD83D\uDD0D Lupa");
                System.out.println("\uD83D\uDCE2 Bocina");
                System.out.println("\uD83D\uDEAB No pasar ");
        }
}
```

Figura 33. Clase Emojis

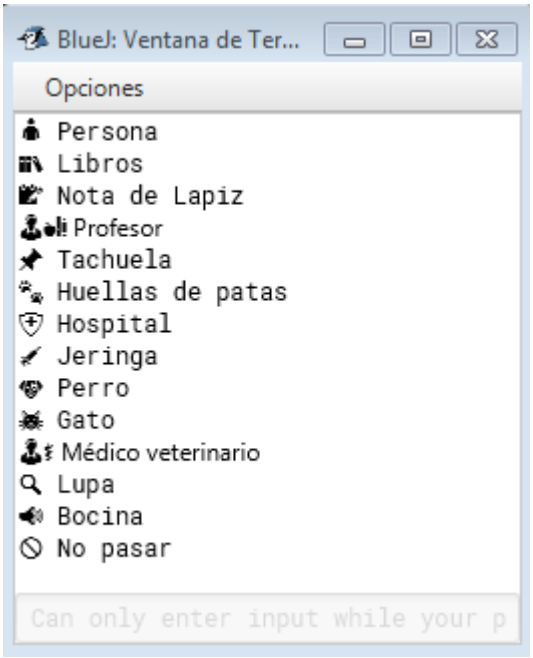


Figura 34. Ejecución del programa

En el enlace siguiente puedes acceder a los códigos respectivos:

Emojis con Unicode

ÍNDICE

3.5.2 Autoevaluación del aprendizaje:

Instrucciones: Completa los espacios en blanco colocando el número del código correspondiente y selecciona la respuesta que muestra la correspondencia correcta.

1. ***getCalificacion***	2. ***extends***	3. Materia
4. Estudiante	5. ***String***	6. ***super***

```
import java.util.ArrayList;
// Subclase Estudiante (Hereda de Persona)
a) class Estudiante ________ Persona {
    private ArrayList<Materia> materias; // Lista de materias con sus calificaciones

    // Constructor de Estudiante
b)  public _________(String nombre) {
c)      _____(nombre);  // Llamamos al constructor de Persona
        this.materias = new ArrayList<>();
    }

    // Método para agregar una nueva materia y calificación
d)  public void agregarMateria(________ materia) {
        materias.add(materia);
    }

    // Método para mostrar todas las materias y calificaciones del estudiante
    public void mostrarMaterias() {
        System.out.println("\n Materias y calificaciones de " + nombre + ":");
        for (Materia materia : materias) {
            System.out.println("  " + materia.getNombre() + " - Profesor: "
                               + materia.getProfesor().getNombre()
e)                             + " | Calificación: " + materia._______._______());
        }
    }
}
```

Opciones de respuesta:

A) a:6 - b:4 - c:1 - d:3 - e:2
B) a:2 - b:3 - c:6 - d:5 - e:1
C) a:6 - b:4 - c:1 - d:5 - e:2
D) a:2 - b:4 - c:6 - d:3 - e:1

ÍNDICE

3.6 RESPUESTAS A LAS AUTOEVALUACIONES.

AUTOEVALUACIÓN	RESPUESTA CORRECTA
3.1.2	Polimorfismo: 1:B , 2:B, 3:D, 4:C
3.3.2	Colaboración de clases: B) I:d, II: f, III: c, IV: a, V: b
3.4.2	Herencia de clases: C) I:d, II: f, III: a, IV: c, V: e
3.5.2	Herencia de clases en la solución de problemas: D) a:2 - b:4 - c:6 - d:3 - e:1

```
   import java.util.ArrayList;
   // Subclase Estudiante (Hereda de Persona)
a) class Estudiante 2. extends Persona {
       private ArrayList<Materia> materias; // Lista de materias con sus calificaciones

       // Constructor de Estudiante
b)     public 4. Estudiante(String nombre) {
c)         6. super(nombre);  // Llamamos al constructor de Persona
           this.materias = new ArrayList<>();
       }

       // Método para agregar una nueva materia y calificación
d)     public void agregarMateria(3. Materia materia) {
           materias.add(materia);
       }

       // Método para mostrar todas las materias y calificaciones del estudiante
       public void mostrarMaterias() {
           System.out.println("\n Materias y calificaciones de " + nombre + ":");
           for (Materia materia : materias) {
               System.out.println(" " + materia.getNombre() + " - Profesor: "
                                  + materia.getProfesor().getNombre()
e)                                + " | Calificación: " + materia. 1. getCalificacion ());
           }
       }
   }
```

ÍNDICE

3.7 REFERENCIAS.

Bing Image Creator. (2025). *Bing Image Creator*. Recuperado de https://www.bing.com/images/create

If Geek then. (2020). *Polimorfismo en Java: Programación orientada a objetos – E l blog de programadores geek*. Recuperado de https://ifgeekthen.nttdata.com/es/polimorfismo-en-java-programaci%C3%B3n-orientada-objetos

Joyanes, L. & Zahonero, I. (2018). *Programación en C, C++, Java y UML*. 2ª Ed. Mac Graw Hill. CDMX. México. Recuperado de https://ebookcentral.proquest.com/lib/bibliodgbmhe/detail.action?docID=3225314

OpenAI. (2025). *Chat GPT*. Recuperado de https://chatgpt.com/

RAE (2023). *Diccionario de la lengua española*. Recuperado de https://dle.rae.es/implementar

Sznajdleder, P. (2000). *Java a fondo - Estudio del lenguaje y desarrollo de aplicaciones.* Ed. Alfa-Omega. Recuperado de hhttps://unam-bibliotecasdigitales-com.pbidi.unam.mx:2443/read/9786077079064/back

Tutoriales, Ya (2024). *Colaboración de clases*. Recuperado de https://www.tutorialesprogramacionya.com/javaya/detalleconcepto.php?codigo=101&punto=&inicio=

U. Rioja. (2024). *Tema 2 – Relaciones entre clases*. Universidad de Rioja. Recuperado de https://www.unirioja.es/cu/jearansa/0910/archivos/EIPR_Tema02.pdf

ÍNDICE

Propósito.

Al finalizar la unidad, cualquier estudiante desarrollará programas en Java utilizando interfaces gráficas de usuario para aplicar y ampliar sus conocimientos de la programación orientada a objetos.

ÍNDICE

4.0 Conceptos Clave.

AWT (Librería).

Es el acrónimo del **Abstract Window Toolkit** para Java. Se trata de un conjunto de clases Java para el desarrollo de las Interfaces Gráficas de Usuario.

Clase.

Término que se usa para crear programas en Java, ya sea una aplicación o un applet; así como para agrupar un conjunto de operaciones y para posibilitar que los usuarios creen sus propios tipos de datos. (Malik, 2013, pág. 71).

Clase *`Graphics`*.

Conjunto de recursos contenidos en el paquete ***`java.awt`***, que generalmente proporcionan métodos para dibujar, entre otros, elementos como líneas, óvalos y rectángulos en la pantalla.

Clase *`Graphics drawLine`*.

Recurso y componente de la **Clase *`Graphics`***, que dibuja una línea, en la pantalla.

Clase *`Graphics drawOval`*.

Recurso y componente de la **Clase *`Graphics`***, que dibuja un óvalo, en la pantalla.

Clase *`Graphics drawPolygon`*.

Recurso y componente de la **Clase *`Graphics`***, que dibuja un polígono, en la pantalla.

Clase *`Graphics drawRect`*.

Recurso y componente de la **Clase *`Graphics`***, que dibuja un rectángulo, en la pantalla.

Clase *`Graphics drawRoundRect`*.

Recurso y componente de la **Clase *`Graphics`***, que dibuja un rectángulo con sus esquinas redondeadas, en la pantalla.

Clase *`Graphics setColor`*.

Recurso y componente de la **Clase *`Graphics`***, que establece el color actual en un componente gráfico.

Clase *Graphics fillOval*.

Recurso y componente de la **Clase *Graphics***, que dibuja un óvalo relleno, en la pantalla.

Clase *Graphics fillPolygon*.

Recurso y componente de la **Clase *Graphics***, que dibuja un polígono relleno, en la pantalla.

Clase *Graphics fillRect*.

Recurso y componente de la **Clase *Graphics***, que dibuja un rectángulo relleno, en la pantalla.

Clase *Graphics fillRoundRect*.

Recurso y componente de la **Clase *Graphics***, que dibuja un rectángulo con sus esquinas redondeadas y relleno, en la pantalla.

Clase *Swing*.

Librería cuya característica principal es que sus componentes GUI (***Graphic User Interface***), favorecen la portabilidad de las aplicaciones (ver *portabilidad*); es decir, es un conjunto de clases que proporciona alternativas poderosas y flexibles a los componentes estándar del ***AWT***. La mayoría de las clases de componentes ***swing***, comienzan con una ***j***.

Clase *Swing JButton*.

Componente botón, ***GUI***.

Clase *Swing JComboBox*.

Componente cuadro combo, ***GUI***, que permite al usuario seleccionar un artículo de una lista de artículos.

Clase *Swing JCheckBox*.

Componente ***GUI***, que puede estar seleccionado o no y que muestra su estado.

Clase *Swing JFrame*.

Componente ventana, ***GUI***.

Clase *Swing JLabel*.

Componente etiqueta, ***GUI***, que muestra una sola línea de texto que se ha especificado.

Clase *Swing JTextArea*.

Componente área de texto, ***GUI***, que actúa como pequeño cojín borrable, sobre el que es posible escribir cualquier texto en cualquier parte.

Clase *Swing JTextField*.

Componente área de texto, ***GUI***, que muestra un rectángulo y permite que el usuario introduzca texto en él.

Clase *Swing JMenu*.

Componente de un menu, ***GUI***.

Clase *Swing JMenuBar*.

Componente ***GUI***, que consiste en una Barra superior del programa que contiene ***JMenu's***.

Clase *Swing JMenuItem*.

Componente ***GUI***, que consiste en cada uno de los elementos de un menú.

Clase Swing JRadioButton.

Componente ***GUI***, que se usa junto con ***ButtonGroup***, y sólo puede haber uno seleccionado.

***GUI* (*Graphic User Interface*).**

Véase **Interfaz Gráfica de Usuario**.

Interfaz Gráfica de Usuario - *GUI* (*Graphic User Interface*).

Es la parte del programa que está conformada por todos los elementos gráficos visuales, mismos que facilitan la comunicación entre el usuario y el sistema, lo que propicia la interacción *usuario-programa* (Luna, 204, pág. 1; Sznajdleder, 2013, pág. 162; e Ingteleco).

java.awt.* **y** ***java.awt.event.****

Véase ***AWT***.

Portabilidad.

Una "pieza" de software es portátil si se puede utilizar en muchos tipos de computadoras.

Proyecto.

Consisten en descripciones del problema cuyas soluciones son programas completos.

ÍNDICE

APRENDIZAJE 4.1

Conoce las características de la Clase *Swing*.

Temática:

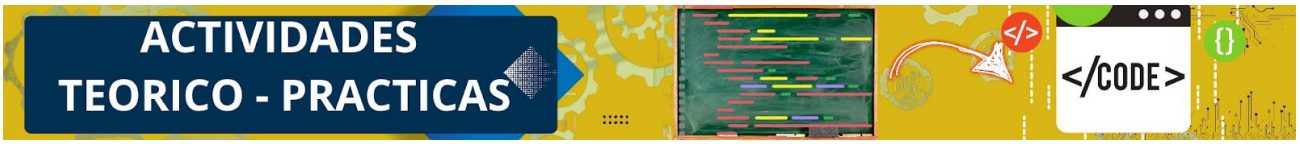
Concepto de interfaz gráfica de usuario (***GUI***).

La Clase ***Swing***.

Componentes ***javax.swing***.

La Clase ***AWT*** como antecedente de la Clase ***Swing***.

Relación de la Clase ***Swing*** con la Librería ***AWT***: ***java.awt***.* y ***java.awt.event***.*.



ÍNDICE

4.1.1. Actividades de aprendizaje teórico–prácticas.

La clase ***swing*** es el núcleo de las Clases de Fundación Java (***JFK***), por sus siglas en inglés: ***Java Foundation Classes***), en el que el componente básico es la ***GUI*** (Interfaz gráfica de Usuario, es decir, ***GUI - Graphic User Interface***), y, generalmente, una ***GUI*** se monta sobre una ventana (***Frame***); Este será el contenedor (***container***) principal que contendrá los componentes de la interfaz gráfica (*ver* ***Figura 1.*** *Jerarquía de Clases* ***Swing***).

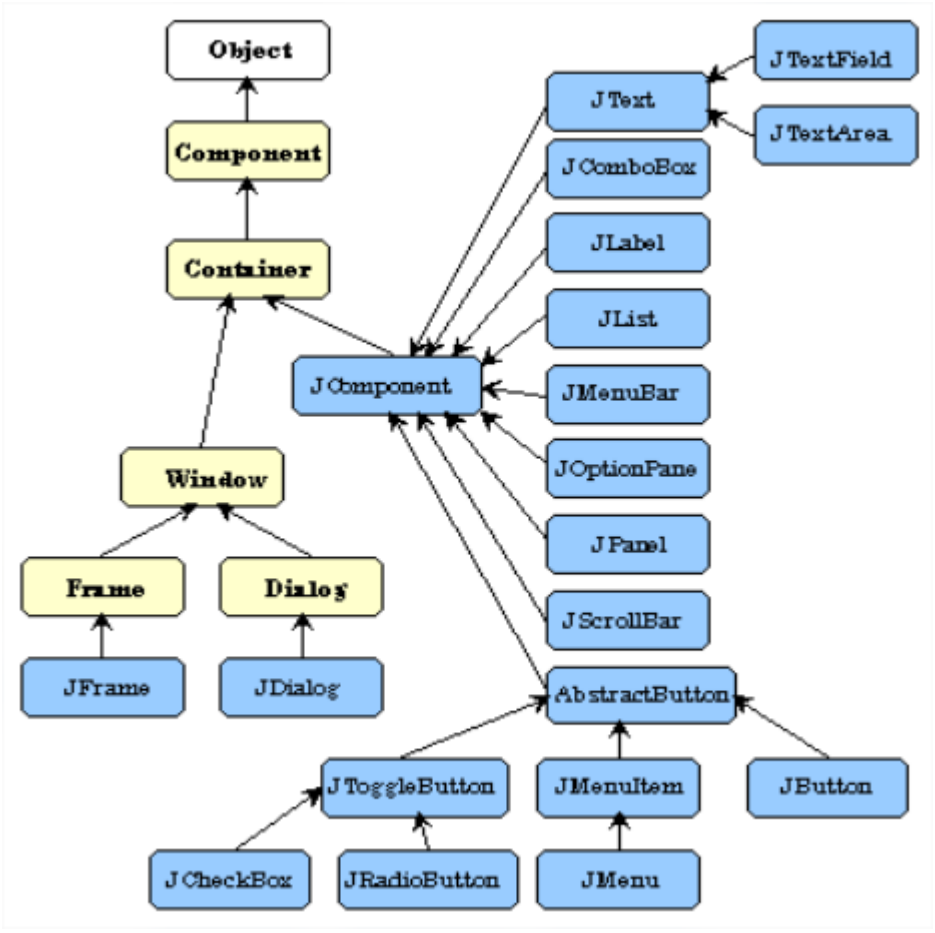


Figura 1. *Jerarquía de Clases* ***Swing****.*

Vale la pena resaltar que el importante antecedente de swing, es ***AWT***; dado que cuando Java fue presentado al mundo, surgió algo llamado kit de herramientas de ventana abstracta, o ***AWT*** (***Abstract Window Toolkit***), lo que marcó la llegada del primer conjunto de ***widgets*** nativos; lo que a su vez, trajo los primeros problemas, ya que su funcionalidad dependía de la plataforma subyacente, porque ***Java*** exigía que si una función no estaba disponible en todas las plataformas ***Java***, entonces, en la práctica, significaba que se debía lidiar con las diferencias ***plataforma-navegador***. No obstante, de lo anterior sobresale que ***AWT*** era extremadamente lento, poco confiable y no permitía hacer mucho con sus componentes nativos. Finalmente, Sun Microsystems, en sociedad con Netscape Communication y otros, crean una biblioteca llamada Java Foundation Classes, o JFC, la que algunas veces, brevemente, se denomina Conjunto de Componentes Swing (Zukowski, 2005; pág. 1).

Para entender mejor este complemento ***AWT-SWING***, veamos algunas ideas relevantes:

- ***AWT*** (***Abstract Windows Toolkit***) y ***SWING*** son librerías de clases, para el desarrollo de ***GUI***.
- Algunos componentes de ***AWT*** usan código nativo y por ello es dependiente de la plataforma.
- ***Swing*** está escrito completamente en Java por lo que es independiente de la plataforma.

Vale la pena resaltar, con base en lo anterior, que las aplicaciones ***Swing*** distribuidas entre varias plataformas tienen la misma apariencia y se pueden considerar como el reemplazo de ***AWT*** (Corcuera, s/f).

Gráficamente, veamos el alcance de ***AWT*** (*ver* ***Figura 2.*** *Jerarquía de Clases* ***AWT***).

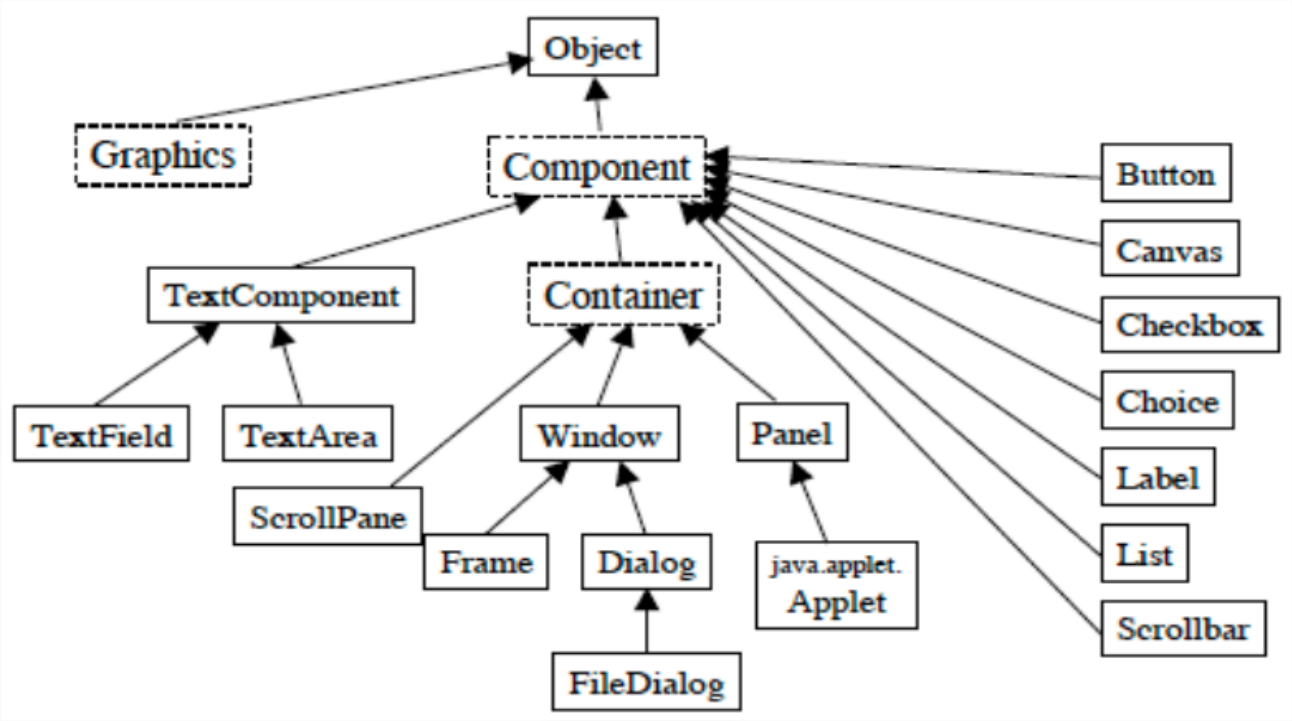


Figura 2. *Jerarquía de Clases* ***AWT****.*

Volvamos a nuestro tema relevante, la Clase ***Swing***.

La ***Figura 1.***, por una parte, muestra que como los ***containers*** también son componentes, entonces un ***container*** podría contener a otros ***containers***; y por otra, muestra un panorama general y completo de los componentes ***Java swing***, aunque en este trabajo sólo se mencionan aquellos que refiere el Programa de Estudios de la asignatura Cibernética y Computación II (*UNAM-CCH,* 2016).

El término ***Interfaz Gráfica de Usuario***, o ***GUI*** (***Graphical User Interface***), entró en existencia, porque las primeras interfaces ***usuario-computadora***, no eran gráficas. Para clarificar esto, veamos algunos antecedentes de ***Interfaz***, en el cuadro siguiente:

N°	CARACTERÍSTICAS	DENOMINACIÓN
1°	Está orientada a texto y teclado y, por lo general, consiste en comandos que el usuario tiene que recordar, y las respuestas de la computadora son breves.	Interfaz de comandos.
2°	Está basada en menús no gráficos, y generalmente se usan las teclas de flechas para elegir una opción.	Interfaz basada en menús.
3°	Está basada en elementos gráficos visuales, mismos que facilitan la comunicación entre el usuario y el sistema.	***Interfaz Gráfica de Usuario (GUI)***

*Evolución de las **Interfaces**.*

A fin de familiarizarte con algunos otros ejemplos de ***interfaz***, acomoda las vocales siguientes donde corresponde, para conocer su denominación:

[***i***]	[***o***]	[***á***]

N°	CARACTERÍSTICAS	DENOMINACIÓN
a)	Una pantalla permite al usuario interactuar directa y físicamente con los elementos del sistema.	Interfaz ***t__ct__l***.
b)	El sistema recibe comandos o solicitudes verbales del usuario y puede responder en consecuencia.	Interfaz de ***V__z***.

*Ejemplos de **Interfaces**.*

En este orden de ideas, los objetos definen su interacción con su entorno exterior a través de los métodos que exponen; así que, los métodos forman la interfaz del objeto con su entorno exterior; por ejemplo, la interfaz entre el usuario y el cableado eléctrico al interior de la carcasa de un televisor, son los botones que están enfrente de la televisión. Ahora, nuevamente, a fin de familiarizarte con algunos dispositivos que sirven como ejemplos de

interfaz, acomoda las letras siguientes donde corresponde, para conocer su denominación:

[***t***]	[***o***]	[***a***]	[***m***]	[***r***]	[***ó***]	[***o***]

N°	CARACTERÍSTICAS	INTERFAZ
i)	El **usuario** y el **sistema** interactúan mediante un dispositivo que visualmente se identifica mediante una flecha en la pantalla.	***R__t__n***.
ii)	El **usuario**, mediante un dispositivo físico con botones, controla inalámbricamente lo que el **sistema** muestra en la pantalla.	***C__ntr__l __e__o__o***.

*Dispositivos que sirven de **Interfaz**.*

Como seguramente ya lo has vivido, la evolución de la interfaz no queda aquí, porque actualmente la multi mencionada ***Interfaz Gráfica de Usuario***, o ***GUI***, antes de nuestro "pasado" ***universo***, ya ha evolucionado al ***metaverso***; así que, seguramente puedes ayudarnos a completar, con las letras siguientes donde corresponde, el Cuadro siguiente, para conocer la ***interfaz*** correspondiente:

[***d***]	[***a***]	[***l***]	[***u***]	[***r***]	[***i***]	[***d***]

N°	CARACTERÍSTICAS	DENOMINACIÓN
4°	Hace uso de gráficos 3D fotorrealistas en entornos increíblemente detallados y realistas, con texturas y efectos visuales que se sienten casi reales, en el que la tecnología de seguimiento de movimiento y la retroalimentación óptica mejoran susceptiblemente la experiencia del usuario; lo que permite interacciones más precisas y realistas.	***__ea__i__a__ v__rt__ __l***.

*Evolución de las **Interfaces**. (Conclusión)*

ÍNDICE

4.1.2 Autoevaluación del aprendizaje:

Ejercicio ***Interfaz*.**

Para cada inciso, en la columna ***interfaz*** completar lo correspondiente, de acuerdo con el entorno exterior respectivo:

	ENTORNO	*interfaz*
a)	Entre la red alámbrica de un auto eléctrico y el conductor, una posible interfaz sería…	
b)	La interfaz, entre una computadora (monitor, teclado, CPU y ratón) y el sillón ejecutivo, sería…	
c)	Entre el código de programación ***java*** y el usuario que utiliza la aplicación, la interfaz sería…	

Cuestionario: Interfaz Gráfica de Usuario (***GUI***).

Instrucciones: Selecciona la opción que contiene todas las respuestas correctas.

I. ¿Cuál de las siguientes clases de ***Swing*** se utiliza para crear una ventana en una ***interfaz gráfica*** de ***Java***?

a) ***JFrame***

b) ***JPanel***

c) ***JButton***

d) ***JLabel***

II. ¿Cuál de las siguientes opciones es un componente de ***interfaz gráfica*** en ***Java*** que permite al usuario ingresar datos?

a) ***JLabel***

b) ***JTextField***

c) ***JComboBox***

d) ***JButton***

III. ¿Qué significa ***AWT*** en ***Java***?

a) ***Advanced Window Toolkit***

b) ***Abstract Window Toolkit***

c) ***Application Window Tools***

d) ***Abstract Widget Technology***

IV. ¿Cuál de las siguientes opciones define mejor una ***interfaz gráfica*** de ***usuario*** (***GUI***) en ***Java***?

a) Un conjunto de clases que permiten crear aplicaciones de consola.

b) Un conjunto de herramientas que permiten diseñar páginas web interactivas.

c) Un conjunto de componentes visuales que permiten la creación de atributos de una clase.

d) Un conjunto de instrucciones que se ejecutan en segundo plano para realizar tareas.

V. ¿Cuál de las siguientes opciones es un ejemplo de contenedor en una ***interfaz gráfica*** de ***Java***?

a) Botón

b) Etiqueta

c) Panel

d) Campo de texto

Opciones de respuesta:

A) I:c - II:b - III:a - IV:c - V:a

B) I:b - II:a - III:a - IV:b - V:a

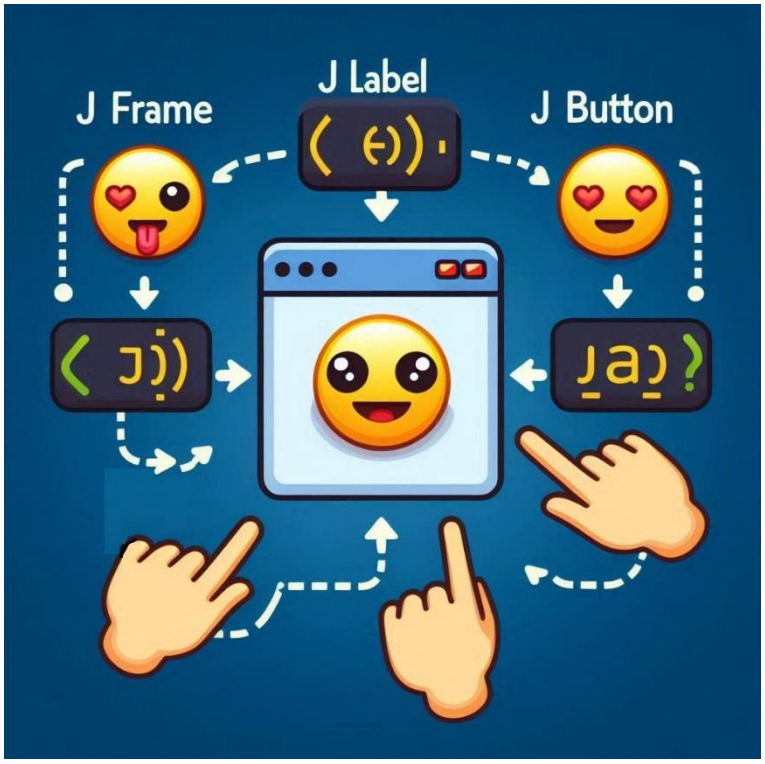
C) I:a - II:b - III:b - IV:c - V:c

D) I:a - II:a - III:c - IV:c - V:b

ÍNDICE

APRENDIZAJE 4.2

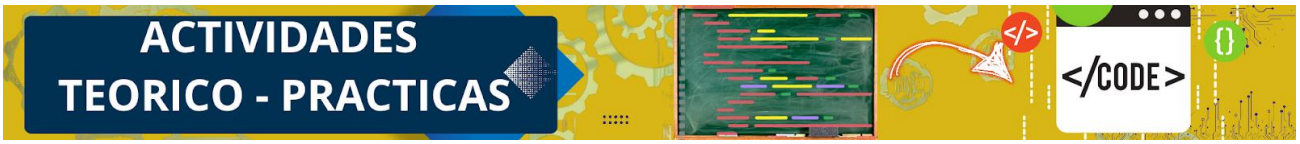
Elabora programas con una interfaz gráfica de usuario, aplicando las Clases: *JFrame*, *JLabel* y *JButton*.



Temática:

Clase ***Swing***:

- ***JFrame***
- ***JLabel***
- ***JButton***



ÍNDICE

4.2.1. Actividades de aprendizaje teórico–prácticas.

Clase ***JFrame***:

Representa el elemento básico para iniciar el desarrollo de una interfaz gráfica. Para utilizarla se deberá crear ***clases*** que hereden de la clase ***JFrame***, por lo tanto, heredará de esta clase sus métodos; a continuación, se listan los métodos básicos:

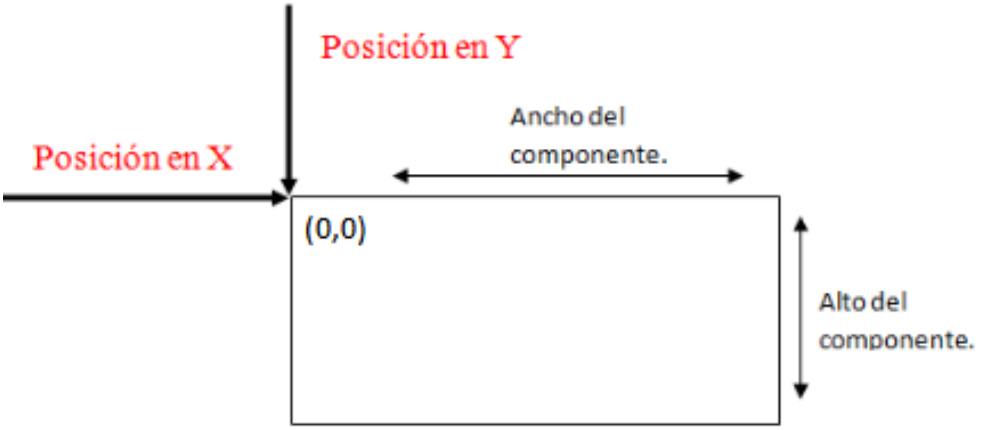
- ***setTitle*(*String* titulo);** Establece el título de la ventana
- ***setSize*(*int* ancho, *int* alto);** Define el tamaño de la ventana
- ***setVisible*(*boolean visible*);** Hace visible o invisible la ventana
- ***setDefaultCloseOperation*(*int* operacion);** Define qué hacer cuando se cierra la ventana
- ***setLocationRelativeTo*(*null*);** Establece la posición de la ventana. Si se le pasa ***null*** como parámetro se posiciona en el centro de la pantalla.
- ***add*(*Component* componente);** Añade elementos a la ventana
- ***setBounds(x,y,ancho, alto)***; se usa para posicionar y dimensionar componentes en una ventana Java. Sus parámetros indican:

  x: es la posición en las coordenadas X de la pantalla.

  y: es la posición en las coordenadas Y de la pantalla.

  ancho: es el ancho que tendrá el componente que estamos manipulando.

  alto: es la altura que tendrá el componente que estamos manipulando.



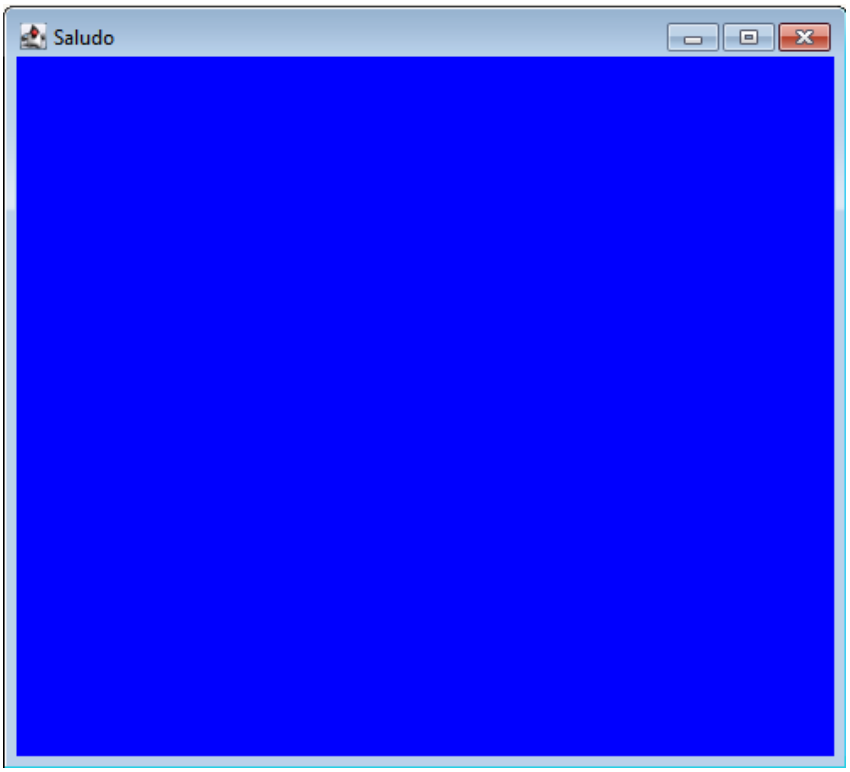
- ***getContentPane().setBackground()***; permite cambiar el color de fondo de una ventana, los colores básicos son:

El siguiente programa muestra el código para crear una ventana cuyas dimensiones son 500 pixeles de ancho y 450 pixeles de alto, de color azul, y centrada.

```java
//Importar las librerias de swing y awt que se van a utilizar.
import javax.swing.*;
import java.awt.Color;
//Se crea la clase Ventana que hereda de la clase JFrame
public class Ventana extends JFrame
{
    //Constructor
    public Ventana()
    {
        //creara una ventana en las coordenadas 0,0 de un
        //ancho de 500 y alto de 450 pixeles.
        setBounds(0,0,500,450);
        //Establece el titulo de la ventana
        setTitle("Saludo");
        //Establece el color de fondo en Azúl
        getContentPane().setBackground(Color.blue);
        //Establece la posición de la ventana al centro
        setLocationRelativeTo(null);


    }


    public static void main(String args[]){
        //Se instancia el objeto ventana1 de la clase Ventana
        Ventana ventana1 = new Ventana();
        //Se hace visible en pantalla la ventana
        ventana1.setVisible(true);
    }
}
```

El resultado correspondiente se muestra en la página siguiente.

Resultado



Clase `JLabel`:

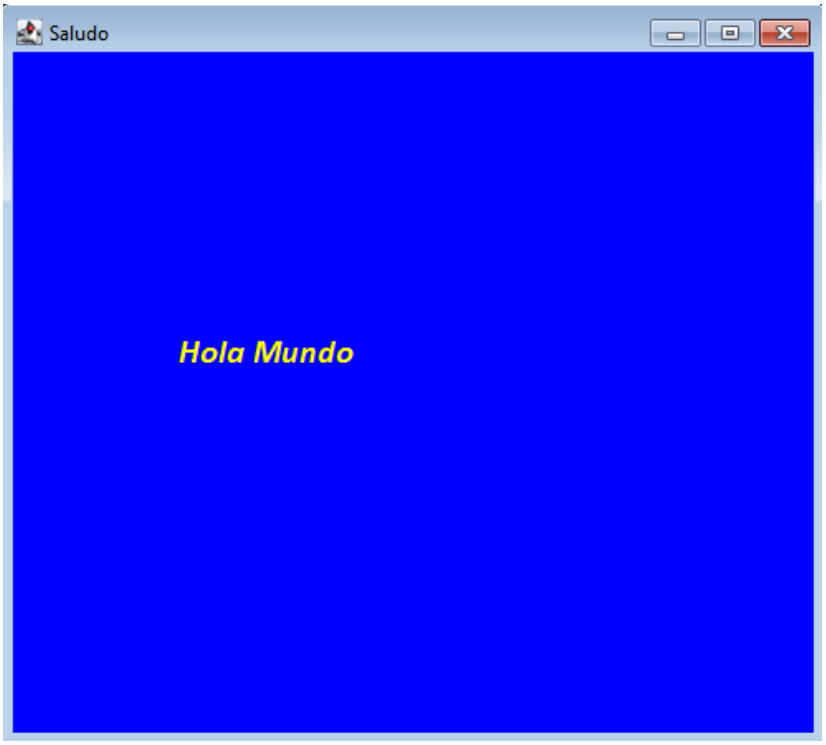
Son textos o también llamadas etiquetas que se muestran en la ventana. Para utilizarla se deberá crear una clase que herede de la clase ***`JLabel`***, por lo tanto, heredará de esta clase sus métodos; a continuación, se listan los métodos básicos:

- ***`setText`*(`String` texto);** Cambia el texto de la etiqueta
- ***`setFont`*(`Font` fuente);** Cambia el tipo de letra, requiere el uso de la biblioteca ***`import java.awt.Font`***;
- ***`setForeground`*(`Color` color);** Cambia el color del texto
- ***`setLayout(null)`*;** permite posicionar componentes de manera absoluta en un contenedor. Esto se hace sin utilizar un administrador de diseño, por lo que el programador es responsable de configurar las posiciones y tamaños de los componentes.

El siguiente programa es el código para mostrar una etiqueta con el texto “Hola mundo”, en las coordenadas ***x***=100, ***y***=175 de la ventana, con dimensiones son 200 pixeles de ancho y 20 pixeles de alto, tipo de letra Calibri de 20 puntos de tamaño en color amarillo.

```java
//Importar las librerias de swing y awt que se van a utilizar.
import javax.swing.*;
//libreria awt.Font requerida para el tamaño y tipo de fuente
import java.awt.Font;
//libreria awt.Color requerida para el color de la ventana y de la fuente
import java.awt.Color;
public class Ventana extends JFrame
{
    //Se declara el atributo lblHola del tipo JLabel es una etiqueta
    private JLabel lblHola;

    //Constructor
    public Ventana()
    {
        //Se establece tamaño, encabezado, color y posición de la ventana
        setBounds(0,0,500,450);
        setTitle("Saludo");
        getContentPane().setBackground(Color.blue);
        setLocationRelativeTo(null);

        //Se habilita el modo gráfico para iniciar la posición de los componentes
        setLayout(null);
        //Se crea el objeto etiqueta llamado lblHola de la clase JLabel
        lblHola = new JLabel();
        //Se establece el texto que mostrara la etiqueta
        lblHola.setText("Hola Mundo");
        //Se establece la pocición x,y el ancho y el alto de la etiqueta
        lblHola.setBounds(100,175,200,20);
        //Se crea un objeto de la clase Font del tipo de fuente y tamaño
        Font fuente1 = new Font("Calibri", 3, 20);
        //Se establece la fuente y el tamaño para la etiqueta lblHola
        lblHola.setFont(fuente1);
        //Se establece el color de la fuente en amarillo
        lblHola.setForeground(Color.YELLOW);
        //Se agrega el objeto etiqueta llamado lblHola a la ventana
        add(lblHola);
    }

    public static void main(String args[]){
        Ventana ventana1 = new Ventana();
        ventana1.setVisible(true);
    }
}
```

Resultado



Clase `JButton`:

Son botones que el usuario puede presionar. Para utilizarlos se deberá crear una clase que herede de la clase ***`JButton`***, por lo tanto, heredará de esta clase sus métodos; a continuación, se listan los métodos básicos:

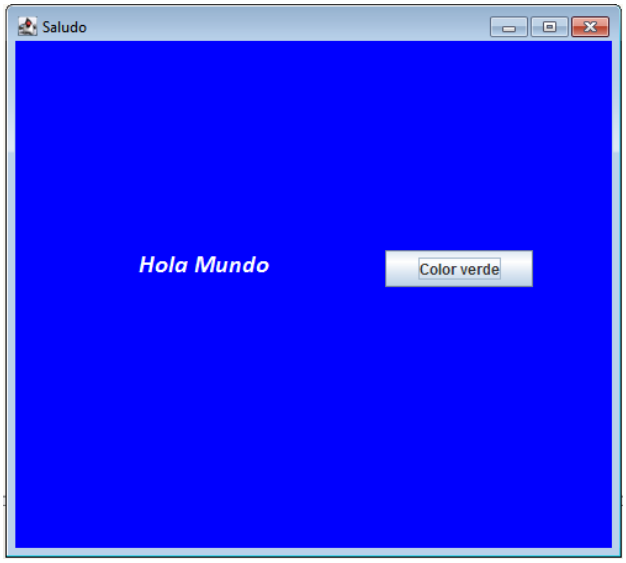
- ***`setText`*****(*****`String`*** **texto*****)*****;** Cambia el texto del botón
- ***`setBounds(int`*** **x,** ***`int`*** **y,** ***`int`*** **ancho,** ***`int`*** **alto*****)*****;** Define la posición y tamaño
- ***`AddActionListener(ActionListener`*** **oyente*****)*****;** Añade un evento cuando se hace clic.
- ***`setEnabled(boolean`*** **activado*****)*****;** Activa o desactiva el botón

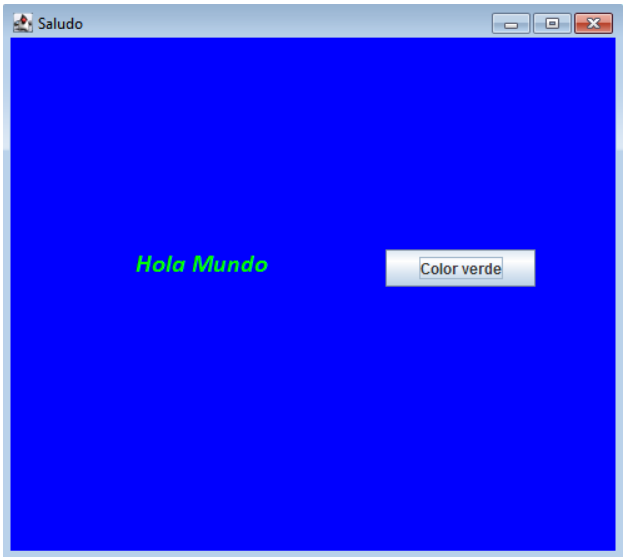
El siguiente programa muestra el código para agregar al programa anterior un botón que cambie el color del mensaje a color verde. El botón se mostrará en las coordenadas ***x***=300, ***y***=170 de la ventana, con dimensiones de 120 píxeles de ancho y 30 píxeles de alto, mostrando el texto “Color verde”.

```java
//Importar las librerias de swing y awt que se van a utilizar.
import javax.swing.*;
//libreria awt.Font requerida para el tamaño y tipo de fuente
import java.awt.Font;
//libreria awt.Color requerida para el color de la ventana y de la fuente
import java.awt.Color;
//libreria awt.event requerida para el uso de eventos tipo clic que tendra el botón
import java.awt.event.*;
//libreria .event.ChangeListener requerida para agregar eventos al botón.
import javax.swing.event.ChangeListener;
public class Ventana extends JFrame implements ActionListener
{
    //Se declara el atributo lblHola del tipo JLabel es una etiqueta
    private JLabel lblHola;
    //Se declara el atributo btnColor del tipo JButton es un Boton
    private JButton btnColor;

    //Constructor
    public Ventana()
    {
        //Se establece tamaño, encabezado, color y posición de la ventana
        setBounds(0,0,500,450);
        setTitle("Saludo");
        getContentPane().setBackground(Color.blue);
        setLocationRelativeTo(null);

        //Se habilita el modo gráfico para iniciar la posición de los componentes
        setLayout(null);
        //Se crea el objeto etiqueta llamado lblHola de la clase JLabel
        lblHola = new JLabel();
        //Se establece el texto que mostrara la etiqueta
        lblHola.setText("Hola Mundo");
        //Se establece la pocición x,y el ancho y el alto de la etiqueta
        lblHola.setBounds(100,175,200,20);
        //Se crea un objeto de la clase Font del tipo de fuente y tamaño
        Font fuente1 = new Font("Calibri", 3, 20);
        //Se establece la fuente y el tamaño para la etiqueta lblHola
        lblHola.setFont(fuente1);
        //Se establece el color de la fuente en blanco
        lblHola.setForeground(Color.WHITE);
        //Se agrega el objeto etiqueta llamado lblHola a la ventana
        add(lblHola);

        //Se crea el objeto botón llamado btlColor de la clase JButton
        btnColor = new JButton();
        //Se establece el texto que mostrara el botón
        btnColor.setText("Color verde");
        //Se establece la pocición x,y el ancho y el alto del botón
        btnColor.setBounds(300,170,120,30);
        //Se agrega el objeto botón llamado btnColor a la ventana
        add(btnColor);
        //Se agrega el evento clic como parte del botón
        btnColor.addActionListener(this);
    }
```

```
    //Método que detecta el clic del botón y si este pertenece a btnColor cambiara
    //el color del objeto lblHola a color verde
    public void actionPerformed(ActionEvent e){
        if(e.getSource() == btnColor){
           lblHola.setForeground(Color.GREEN);
            }
        }

    public static void main(String args[]){
        Ventana ventana1 = new Ventana();
        ventana1.setVisible(true);
    }
}
```

Resultado





ÍNDICE

4.2.2 Autoevaluación del aprendizaje:

Sopa de letras.

Instrucciones: Encuentra la lista de palabras clave en la sopa de letras.

Interface Gráfica de Usuario

C	O	P	J	Ñ	C	G	N	E	V
S	O	J	N	H	O	B	O	L	I
I	E	N	I	N	L	G	T	E	S
J	N	V	T	K	O	C	T	B	I
Í	F	T	E	E	R	B	U	A	B
G	L	R	E	N	N	F	B	L	L
Ú	N	A	A	R	T	E	J	J	E
Ñ	W	I	H	M	F	O	D	X	H
T	W	T	W	F	E	A	B	O	G
Ñ	G	C	R	S	U	X	Z	D	R

- ***JFrame***
- ***Contenedor***
- ***Visible***
- ***JLabel***
- ***Color***
- ***Swing***
- ***Interfaz***
- ***Evento***
- ***Jbutton***
- ***AWT***

ÍNDICE

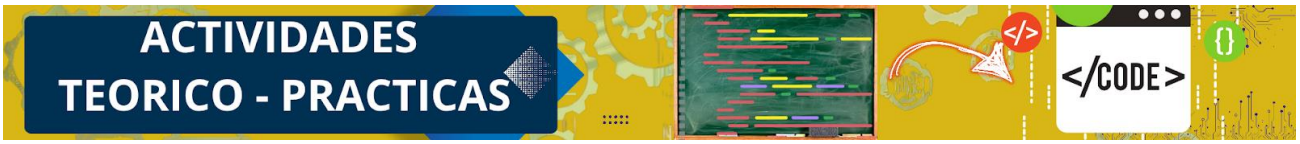
APRENDIZAJE 4.3

Propone un proyecto que utilice las Clases *JFrame*, *JLabel* y *JButton*.

Temática:

Clase ***Swing***:

- ***JFrame***
- ***JLabel***
- ***JButton***.



ÍNDICE

4.3.1. Actividades de aprendizaje teórico–prácticas.

Se desea un programa que, al oprimir un botón, la ventana cambie a Verde, Rojo o Azul, según el botón seleccionado y mande el mensaje con la leyenda del color seleccionado (“Verde”, “Rojo“ o “Azul”).

Respuesta:

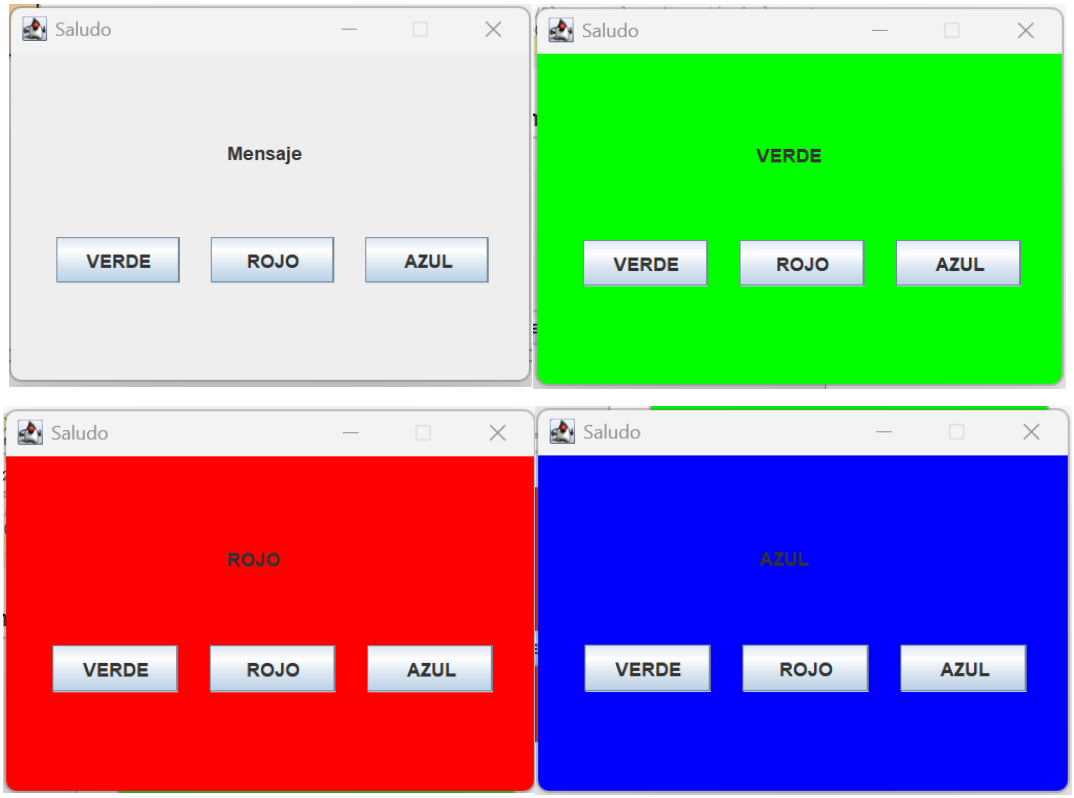
a. Escribir el siguiente código.

Clase Formulario 2:

```
//Importamos la libreria para el desarrollo de nuestro formulario
import javax.swing.*;
//Importamos la libreria para el evento de los botones
import java.awt.event.*;
//Librerai para el color
import java.awt.Color;
//Creamos la clase formulario
public class Formulario2 extends JFrame implements ActionListener{
    //Declaramos los atributos de mi formulario
    private JLabel label1;
    private JButton boton1;
    private JButton boton2;
    private JButton boton3;
    //Creamos el construtor para la estructura de mi formulario
    public Formulario2(){
        setLayout(null);
        //Creamos la etiqueta JLabel1
        label1 = new JLabel("Mensaje"); //Texto de la etiqueta
        label1.setBounds(140,50,100,30);//Dimensiones de la etiqueta
        add(label1);//Agregamos la etiqueta
        //Creamos el botón JBoton1
        boton1 = new JButton("VERDE");//Texto del botón
        boton1.setBounds(30,120,80,30);//Dimensiones del bóton
        add(boton1);//Agregamos el botón
        boton1.addActionListener(this);//Mandamos a llamar al evento del botón
        //Creamos el botón JBoton2
        boton2 = new JButton("ROJO");//Texto del botón
        boton2.setBounds(130,120,80,30);//Dimensiones del bóton
        add(boton2);//Agregamos el botón
        boton2.addActionListener(this);//Mandamos a llamar al evento del botón
        //Creamos el botón JBoton3
```

```
        boton3 = new JButton("AZUL");//Texto del botón
        boton3.setBounds(230,120,80,30);//Dimensiones del bóton
        add(boton3);//Agregamos el botón
        boton3.addActionListener(this);//Mandamos a llamar al evento del botón
    }
    //Creamos el evento para los botones,para cambiar el color al JFrame
    public void actionPerformed(ActionEvent e){
        if(e.getSource() == boton1){
            //Mandamos a imprimir VERDE al JLabel1
            label1.setText("VERDE");
            getContentPane().setBackground(Color.GREEN);
        }if(e.getSource() == boton2){
            //Mandamos a imprimir ROJO al JLabel1
            label1.setText("ROJO");
            getContentPane().setBackground(Color.RED);
        }
        if(e.getSource() == boton3){
            //Mandamos a imprimir AZUL al JLabel1
            label1.setText("AZUL");
            getContentPane().setBackground(Color.BLUE);
        }
    }
    //Creamos el método main
    public static void main(String args[]){
        Formulario2 formulario1 =  new Formulario2();
        formulario1.setTitle("Saludo");//Titulo de la ventana
        formulario1.setBounds(0, 0, 350, 250);//Dimensiones de la ventana
        formulario1.setVisible(true);//Mostramos la ventana
        formulario1.setResizable(false);//Bloquemos la redimensión de la ventana
        formulario1.setLocationRelativeTo(null);//Centramos la venta
    }
}
```

b. Compilar y ejecutar el programa (Realizar pruebas).

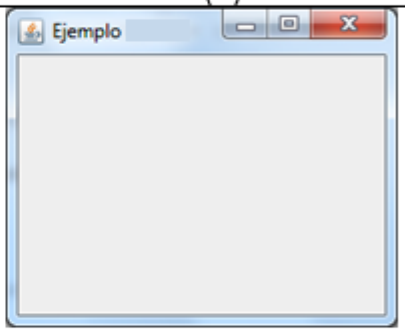
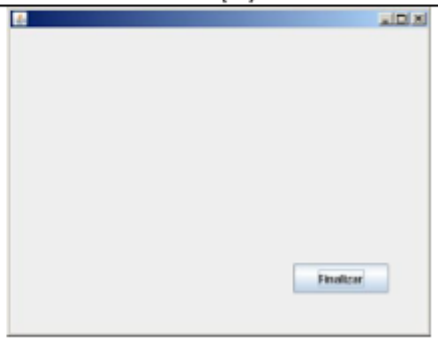


ÍNDICE

4.3.2 Autoevaluación del aprendizaje:

Relación de columnas.

Instrucciones: Relaciona cada una de las imágenes con la descripción de cada clase y selecciona la respuesta que muestra la correspondencia correcta.

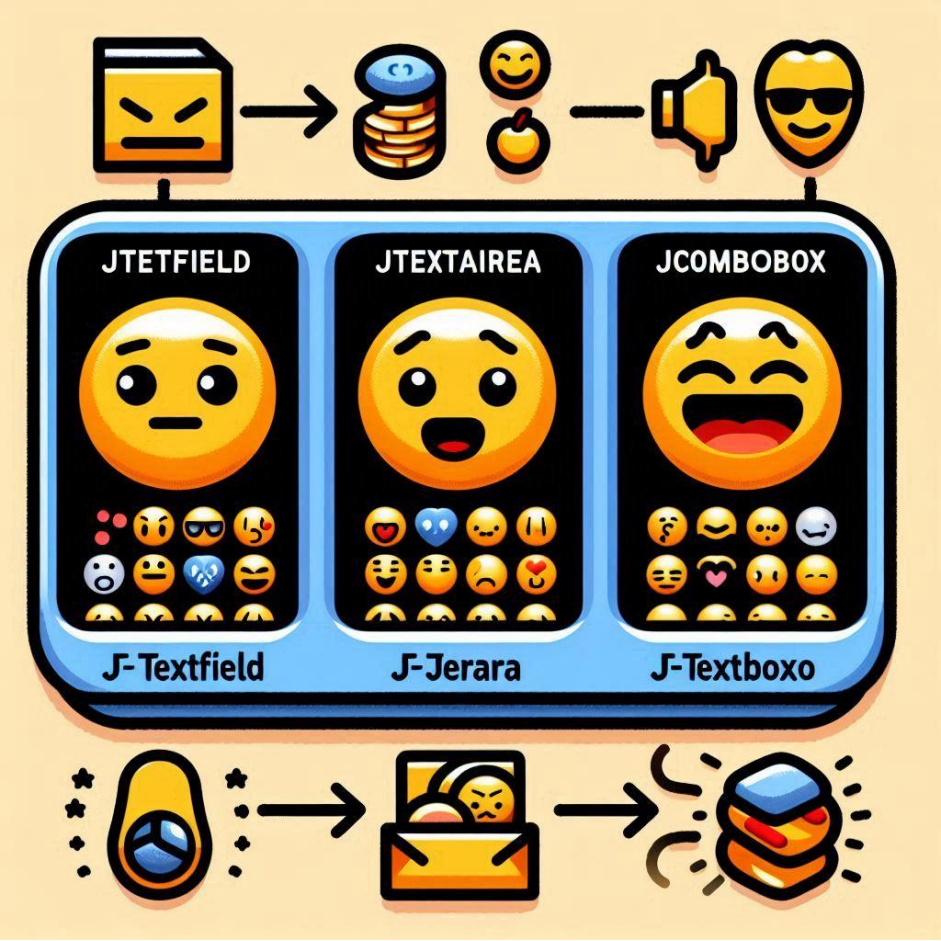
a) JFrame: Contiene todos los demás componentes de la interfaz. Es la base de la ventana visible para el usuario.	 1.- ( )
b) JButton: Permite al usuario realizar una acción al hacer clic. Es un elemento interactivo.	 2.- ( )
c) JLabel: Muestra texto o imágenes estáticas. No es interactiva.	 3.- ( )

A) 1:b - 2:a - 3:c
B) 1:c - 2:a - 3:b
C) 1:a - 2:c - 3:b
D) 1:a - 2:b - 3:c

ÍNDICE

APRENDIZAJE 4.4

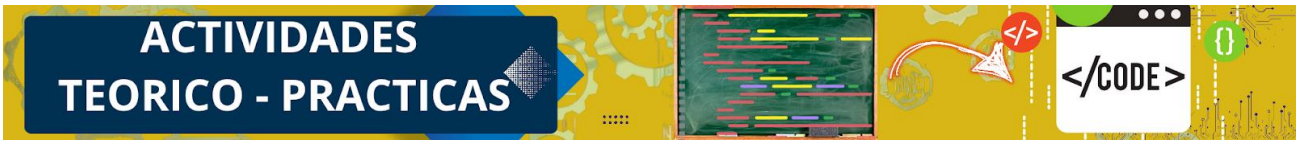
Elabora programas con interfaz gráfica de usuario, aplicando las Clases: *JTextField*, *JTextArea* y *JComboBox*.



Temática:

Clase ***Swing***:

- ***JTextField***
- ***JTextArea***
- ***JComboBox***



ÍNDICE

4.4.1. Actividades de aprendizaje teórico–prácticas.

Clase `JTextField`:

Es un componente que permite al usuario ingresar una sola línea de texto. Es comúnmente utilizado para entradas cortas como nombres, correos electrónicos, o números, sus principales características son: puede habilitar o deshabilitar la edición del campo, puede mostrar un texto predeterminado, puede capturar el texto ingresado por el usuario; a continuación, se listan sus métodos básicos:

- ***`setText(String text)`***; Establece el texto. txtNombre.***`setText`***("Escribe aquí");
- ***`getText()`*;** Obtiene el texto. String texto = txtNombre.***`getText`***();
- ***`setColumns(int columns);`*** Define el ancho. txtNombre.***`setColumns`***(20);
- ***`setEditable(boolean` editable`);`*** Habilita/deshabilita edición. txtNombre.setEditable(***`false`***);
- ***`setForeground(Color` color`);`*** Cambia el color del texto txtNombre.setForeground(Color.BLUE);
- ***`setBackground(Color` color`)`*** → Cambia el color de fondo txtNombre.***`setBackground`***(***`Color`***.LIGHT_GRAY);
- ***`setFont(Font` font`)`*** → Cambia la fuente del texto. → txtNombre.setFont(***`new Font`***("Arial", ***`Font.BOLD`***, 14));

Ejemplo:

```
JTextField txtNombre = new JTextField(20); // Campo con 20 columnas
txtNombre.setText("Escribe tu nombre");  // Texto predeterminado
```

..

Clase `JTextArea`

Permite al usuario ingresar varias líneas de texto, ideal para comentarios, descripciones o cualquier entrada extensa, sus principales características son: establecer un número específico de filas y columnas, le es posible ajustar el texto en el área (***`setLineWrap(true)`***), agrega una barra de desplazamiento (***`JScrollPane`***), puede ser

usado para mostrar texto sin permitir su edición; a continuación, se listan sus métodos básicos:

- ***setText*(*String* text);** Establece el texto.
  `txtComentario.setText("Escribe tu comentario aquí...");`
- ***getText*();** Obtiene el texto.
  `String comentario = txtComentario.getText();`
- ***setRows*(*int rows*);** Define las filas visibles.
  `txtComentario.setRows(5);`
- ***setColumns*(*int columns*);** Define las columnas visibles.
  `txtComentario.setColumns(30);`
- ***setLineWrap*(*boolean wrap*);** Activa salto de línea automático.
  `txtComentario.setLineWrap(true);`
- ***setWrapStyleWord*(*boolean word*);** Salto de línea por palabra completa.
  `txtComentario.setWrapStyleWord(true);`
- ***setEditable*(*boolean* editable);** Habilita/deshabilita edición.
  `txtComentario.setEditable(false);`

Ejemplo:

```
JTextArea txtComentario = new JTextArea(5, 30); // 5 filas, 30 columnas
txtComentario.setLineWrap(true); // Activar salto de línea automático
txtComentario.setWrapStyleWord(true); // Salto por palabras completas
JScrollPane scrollPane = new JScrollPane(txtComentario); // Agregar scroll
```

Clase JComboBox

Permite seleccionar un solo elemento de una lista desplegable. Es útil cuando se tienen opciones predefinidas, sus principales características son: puede agregar o eliminar elementos dinámicamente, permite establecer una opción seleccionada por defecto, permite recuperar el elemento seleccionado, se puede configurar para permitir la edición manual, a continuación se listan sus métodos básicos:

- ***addItem*(*String item*);** Agrega un elemento. cmbColores.*addItem*("Rojo");

- ***getSelectedItem()*;** Obtiene el elemento seleccionado. → ***String*** color = (***String***) cmbColores.***getSelectedItem***();
- ***setSelectedIndex(int index)*;** Selecciona un índice. cmbColores.***setSelectedIndex***(1);
- ***removeItem(String item)*;** Elimina un elemento. cmbColores.***removeItem***("Azul");
- ***removeAllItems()*;** Elimina todos los elementos. cmbColores.***removeAllItems()***;
- ***setEditable*(*boolean* editable);** Permite escribir en la lista. cmbColores.***setEditable(true)***;
- ***getItemCount()*;** Obtiene la cantidad de elementos. ***int count*** = cmbColores.***getItemCount***();

Ejemplo:

```
JComboBox<String> cmbColores = new JComboBox<>(new String[]{"Rojo", "Verde", "Azul"});
cmbColores.setSelectedIndex(1); // Selecciona "Verde" por defecto
String colorSeleccionado = (String) cmbColores.getSelectedItem();
```

A continuación, se elabora un programa con interfaz gráfica de usuario, aplicando las Clases: ***JTextField***, ***JTextArea*** y ***JComboBox***.

Se desea un programa que le solicitarle al usuario su nombre en un campo de texto ***(JTextField)***, después debe seleccionar un elemento (AZUL, ROJO, VERDE, AMARILLO) de una lista desplegable (***JComboBox***), y mostrar un mensaje en un área de texto (***JTextArea***), que le dará un saludo personalizado con su nombre y le dirá el color que seleccionó.

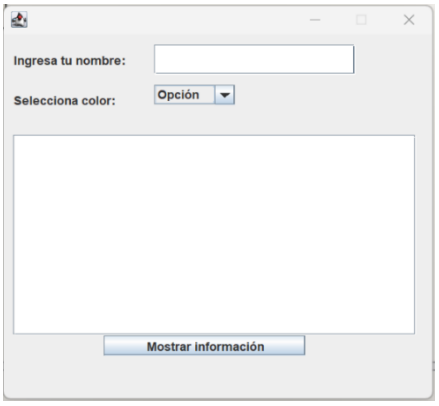
Escribir el siguiente código.

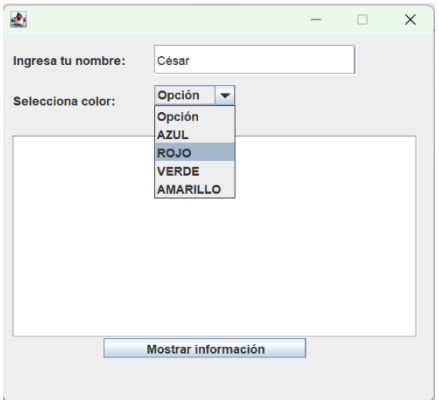
Clase Formulario:

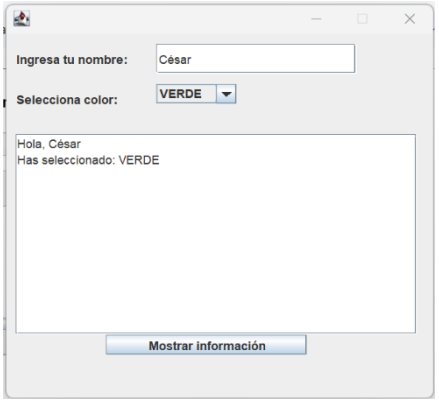
```
import javax.swing.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener{
    //Declaramos los atributos de mi formulario
    private JTextField textfield1;
    private JLabel textlabel1;
    private JLabel textlabel2;
    private JComboBox combo1;
    private JButton boton1;
    private JScrollPane scrollpane1;
    private JTextArea textarea1;
    String nombre = "";
    public Formulario(){
        setLayout(null);
        //Creamos el JTextField
        textfield1 = new JTextField();
        textfield1.setBounds(150,10,200,30);
        add(textfield1);
        //Creamos las etiquetas JLabel1
        JLabel textlabel1 = new JLabel("Ingresa tu nombre: ");
        textlabel1.setBounds(10,10,200,30);
        add(textlabel1);
        JLabel textlabel2 = new JLabel("Selecciona color: ");
        textlabel2.setBounds(10,50,200,30);
        add(textlabel2);
        //Creamos el JComboBox
        combo1 = new JComboBox();
        combo1.setBounds(150,50,80,20);
        add(combo1);
        combo1.addItem("Opción");
        combo1.addItem("AZUL");
        combo1.addItem("ROJO");
```

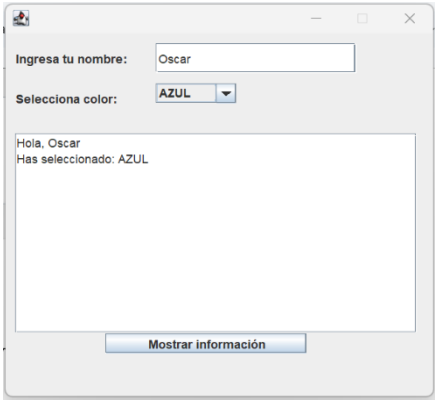
```
        combo1.addItem("VERDE");
        combo1.addItem("AMARILLO");
        //Creamos el botón JBoton1
        boton1 = new JButton("Mostrar información");
        boton1.setBounds(100,300,200,20);
        add(boton1);
        boton1.addActionListener(this);
        //Creamos el JTextArea1
        textarea1 = new JTextArea();
        scrollpane1 = new JScrollPane(textarea1);
        scrollpane1.setBounds(10,100,400,200);
        add(scrollpane1);
    }
    //Creamos el evento para el botón
    public void actionPerformed(ActionEvent e){
        if(e.getSource() == boton1){
            String nombre = textfield1.getText();
            String selecOpcion = (String) combo1.getSelectedItem();
            textarea1.setText("Hola, " + nombre + " \nHas seleccionado: " + selecOpcion);
        }
    }
    //Creamos el método main
    public static void main(String args[]){
        Formulario formulario1 = new Formulario();
        formulario1.setBounds(0,0,440,400);
        formulario1.setVisible(true);
        formulario1.setResizable(false);
        formulario1.setLocationRelativeTo(null);
    }
}
```

b. Compilar y ejecutar el programa (Realizar pruebas).









ÍNDICE

4.4.2 Autoevaluación del aprendizaje:

Completa los espacios en blanco.

Instrucciones: En los espacios en blanco, escribe el número que corresponda a la declaración básica de cada clase y selecciona la respuesta que muestra la correspondencia correcta.

1.- ***JButton***	2.- ***JComboBox***	3.- ***JTextArea***	4.- ***JTextField***

a)
b)
c)

```
import javax.swing.*;
public class EjercicioComponentes {
    public static void main(String[] args) {
        // Crear un campo de texto para ingresar un nombre
        _________ nombre = new __________ ("Ingrese su nombre");
        // Crear un área de texto para escribir una dirección
        _________ direccion = new __________ (5, 20);
        // Crear una lista desplegable con opciones de países
        String[] paises = {"Colombia", "Argentina", "México", "España"};
        _________ <String> pais = new _________ (paises);
    }
}
```

A) a:3 - b:4 - c:1

B) a:4 - b:2 - c:3

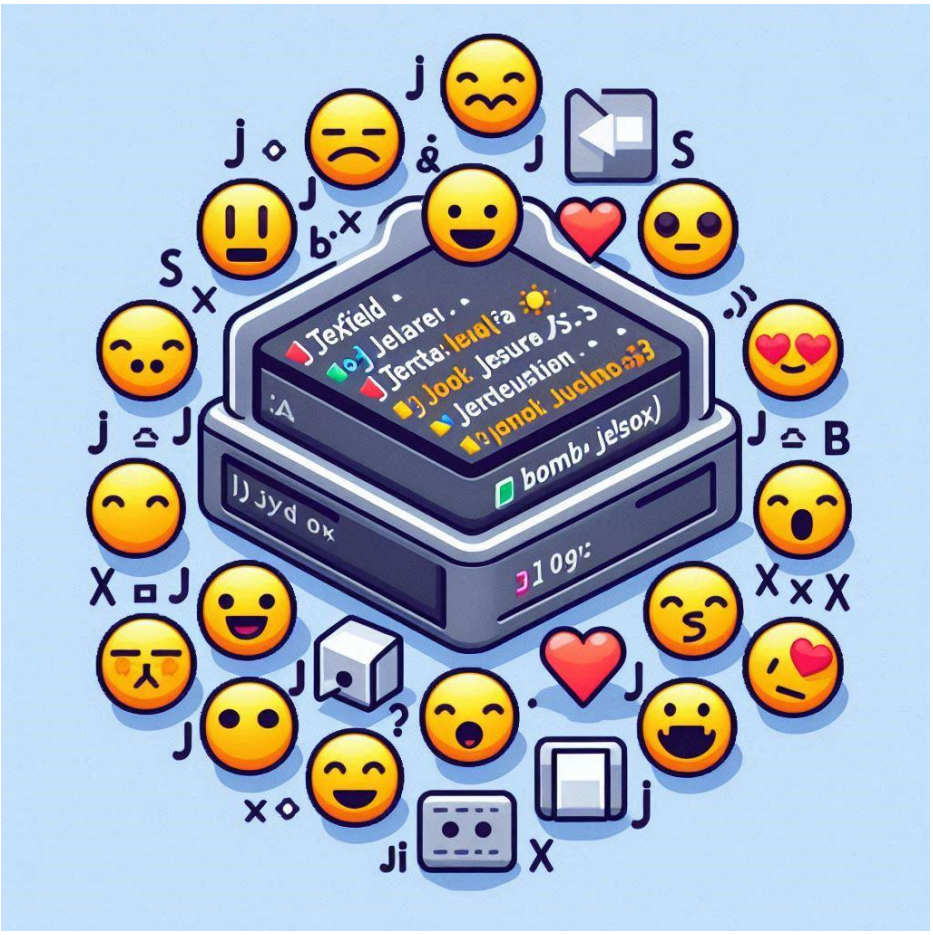
C) a:1 - b:2 - c:3

D) a:4 - b:3 - c:2

ÍNDICE

APRENDIZAJE 4.5

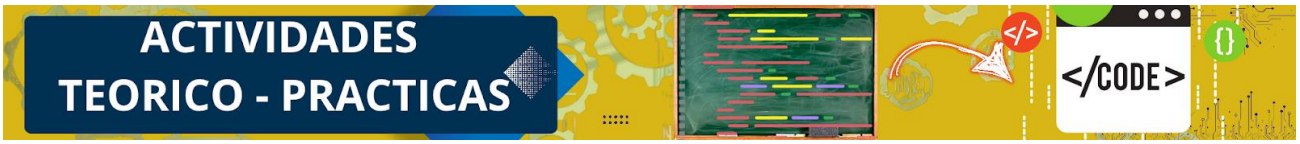
Propone un proyecto que utilice las Clases *JTextField*, *JTextArea* y *JComboBox*.



Temática:

Clase ***Swing***:

- ***JTextField***
- ***JTextArea***
- ***JComboBox***



ÍNDICE

4.5.1. Actividades de aprendizaje teórico–prácticas.

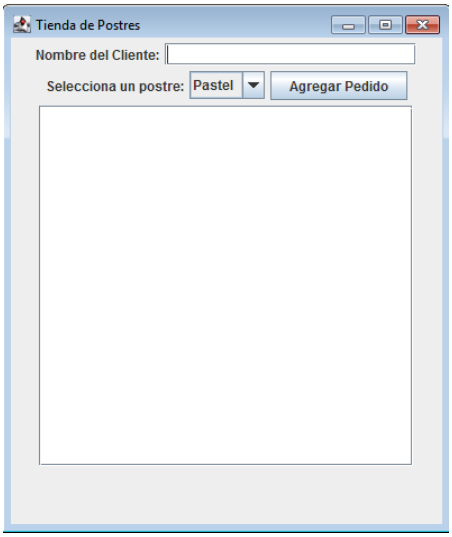
Se desea un proyecto de una **Tienda de Postres** que le solicitarle al cliente su nombre en un campo de texto (***JTextField***), después debe permitir seleccionar postres (Pastel, Helado, Galleta, Flan) mediante una lista desplegable (***JComboBox***), y al pulsar el botón “Agregar Pedido”, se mostrará en el área la selección el nombre del cliente, el postre elegido y su costo (***JTextArea***), de la misma manera cada que agregue otro postre.

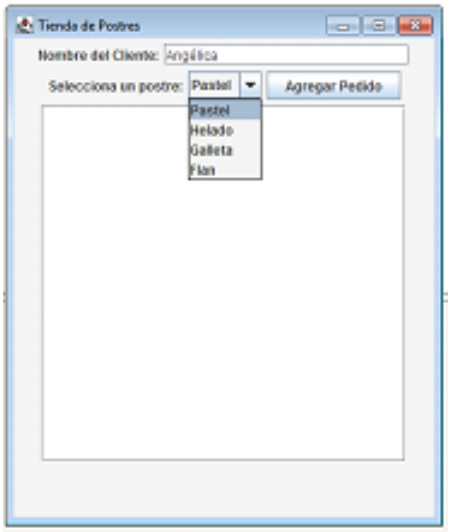
Escribir el siguiente código.

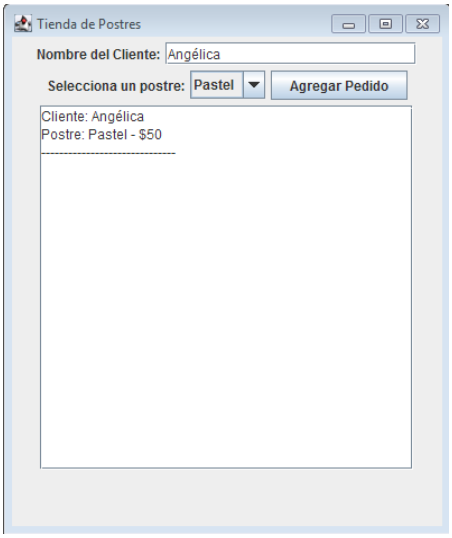
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class TiendaPostres {
    public static void main(String[] args) {
        new VentanaPostres().setVisible(true); // Solo hace visible la ventana
    }
}

class VentanaPostres extends JFrame {
    public VentanaPostres() {
        // Configuración de la ventana
        setTitle("Tienda de Postres");
        setSize(400, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());

        // Etiqueta y campo de texto para el nombre del cliente
        JLabel lblNombre = new JLabel("Nombre del Cliente:");
        JTextField txtNombre = new JTextField(20);

        // Etiqueta y lista desplegable de postres
        JLabel lblPostres = new JLabel("Selecciona un postre:");
        JComboBox<String> cmbPostres = new JComboBox<>(new String[]{"Pastel", "Helado", "Galleta", "Flan"});
        // Área de texto para mostrar los detalles de la compra
        JTextArea txtDetalle = new JTextArea(20, 30);
        txtDetalle.setEditable(false); // Solo lectura
        JScrollPane scrollPane = new JScrollPane(txtDetalle);

        // Botón para calcular el total
        JButton btnCalcular = new JButton("Agregar Pedido");

        // Acción del botón
        btnCalcular.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String nombre = txtNombre.getText(); // Obtener el nombre del cliente
                String postreSeleccionado = (String) cmbPostres.getSelectedItem(); // Obtener el postre seleccionado
                int precio = 0;

                // Estructura condicional para asignar el precio del postre
                if (postreSeleccionado.equals("Pastel")) {
                    precio = 50;
                } else if (postreSeleccionado.equals("Helado")) {
                    precio = 30;
                } else if (postreSeleccionado.equals("Galleta")) {
                    precio = 15;
                } else if (postreSeleccionado.equals("Flan")) {
                    precio = 40;
                }
```

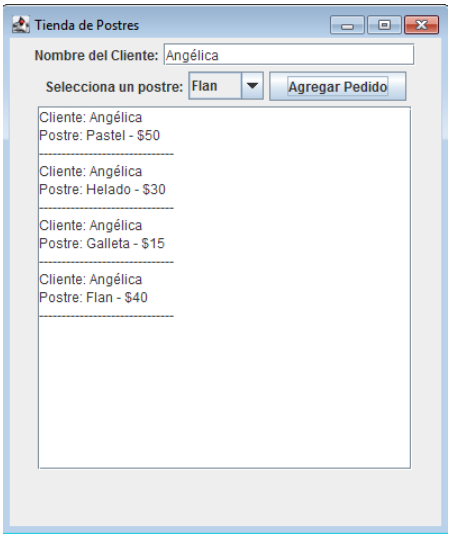
```
            // Mostrar la selección y el total en el área de texto
            txtDetalle.append("Cliente: " + nombre + "\n");
            txtDetalle.append("Postre: " + postreSeleccionado + " - $" + precio + "\n");
            txtDetalle.append("------------------------------\n");
        }
    });

    // Agregar componentes a la ventana
    add(lblNombre);
    add(txtNombre);
    add(lblPostres);
    add(cmbPostres);
    add(btnCalcular);
    add(scrollPane);
  }
}
```

Compilar y ejecutar el programa (Realizar pruebas).





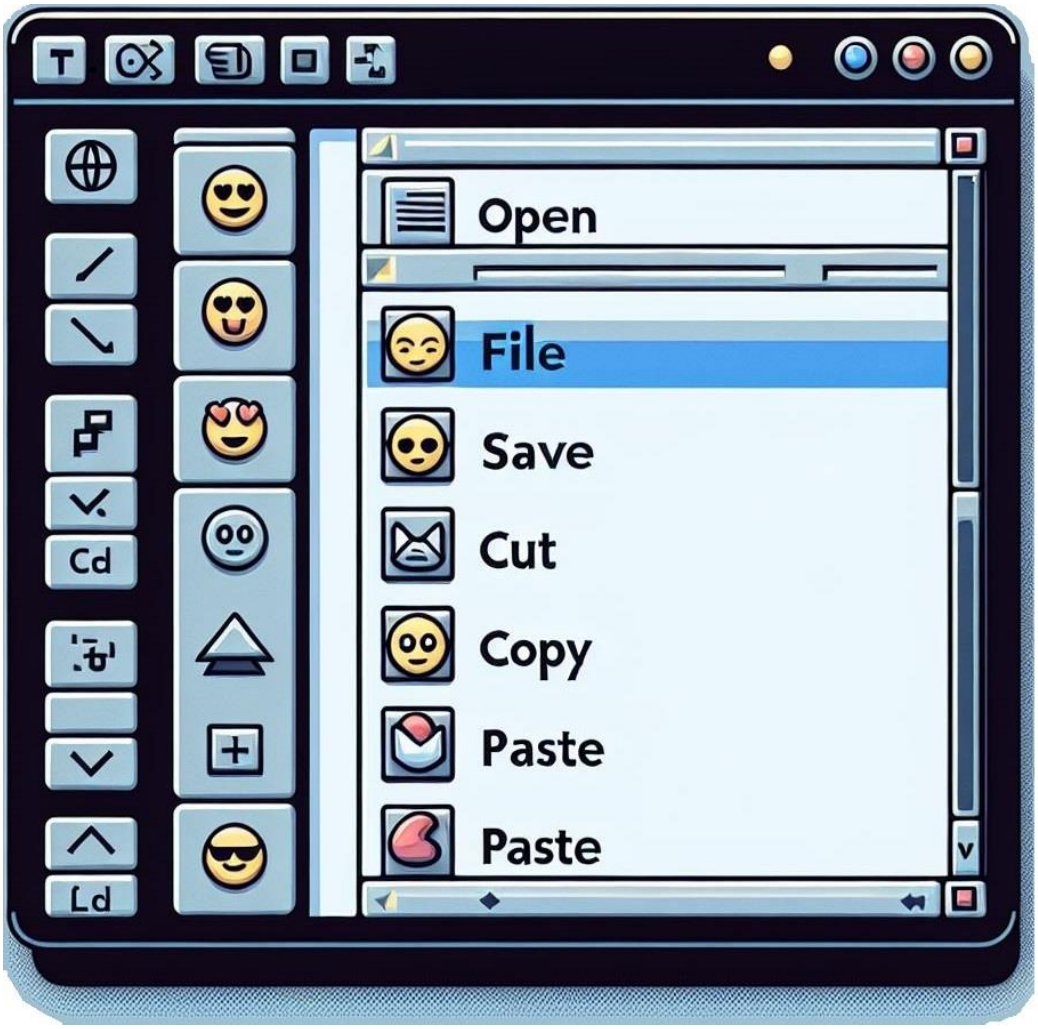




ÍNDICE

APRENDIZAJE 4.6

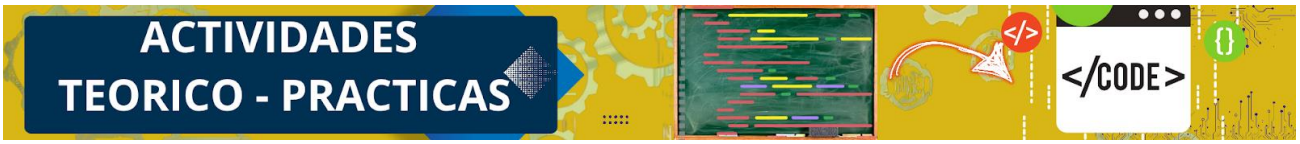
Elabora programas con interfaz gráfica de usuario, aplicando las Clases: *JMenuBar*, *JMenu* y *JMenuItem*.



Temática:

Clase ***Swing***:

- ***JMenuBar***
- ***JMenu***
- ***JMenuItem***



4.6.1. Actividades de aprendizaje teórico–prácticas.

La manera en que todos los componentes de menú se interrelacionan, se muestra en la figura ***Figura 3*** (Zukowski, 2005):

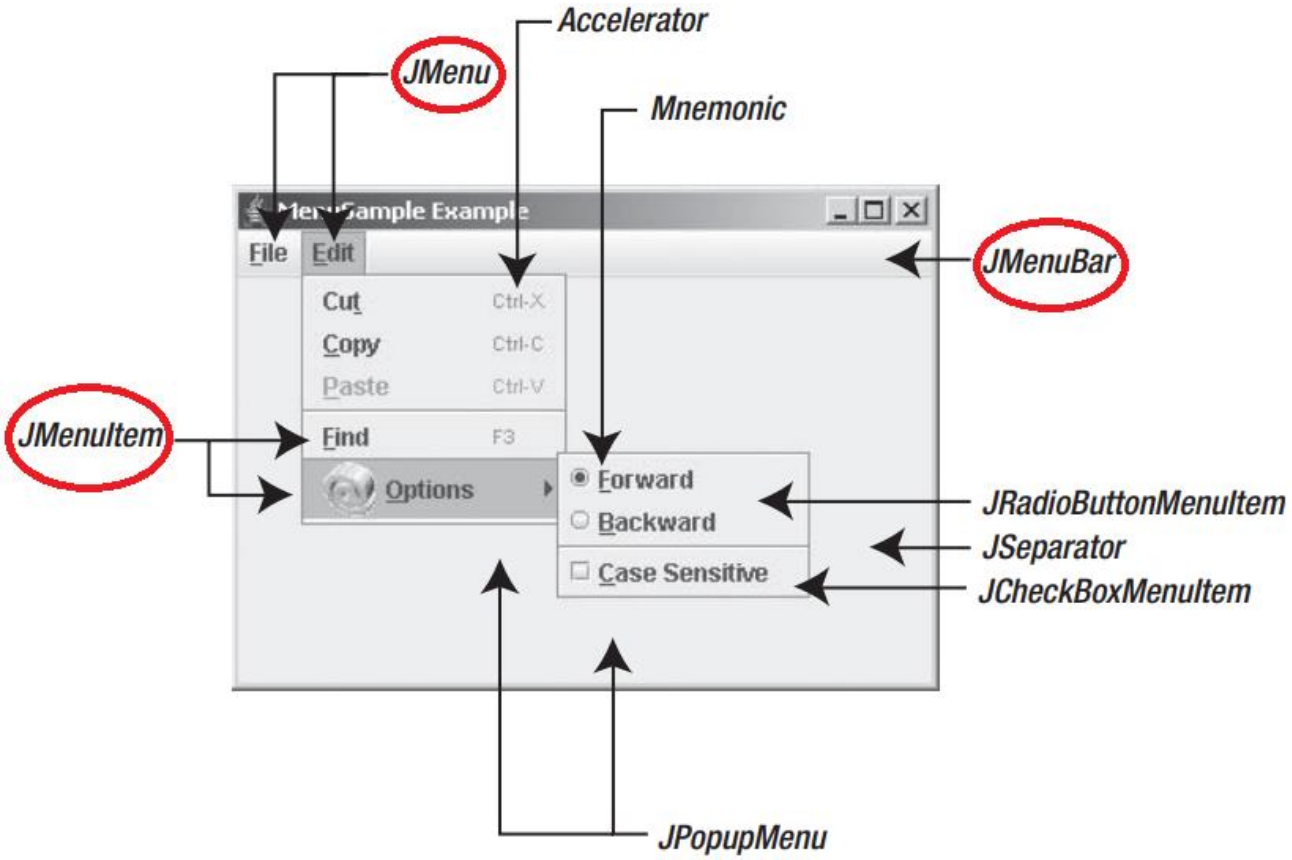


Figura 3. *Ejemplo Componentes de* ***Menu****.*

No obstante que en la ***Figura 3*** se muestran todos los componentes que se interrelacionan con menús, en este apartado únicamente se abordarán los componentes que se resaltan en **círculos rojos**:

JMenuBar	Es la barra en la que se alojan los menús (***JMenu***).
JMenu	Es una ventana emergente que, por una parte, contiene ***JMenuItems*** que se muestran cuando el usuario selecciona un elemento en ***JMenuBar***; y por otra, contiene entradas o elementos del menú (***JMenuItem***).
JMenuItem	Es un elemento o una entrada de un menú.

Características de las Clases ***JMenuBar***, ***JMenu*** y ***JMenuItem****.*

En palabras más digeribles, dentro de la clase ***Swing***, el componente "barra de menú" es ***JMenuBar*** , y para su funcionamiento, como se ve en la figura **Figura *4-6-1***, primero se requiere que se llene con elementos **menú** (***JMenu***), y este último, a su vez, puede contener **elementos** o **entradas** (***JMenuItem***); segundo, agregar el menu a la "barra de menú" a un ***JFrame*** o algún otro componente de la interfaz de usuario que acepte una barra de menú (***JMenuBar***):

Para la creación, y habilitación, de las Clases ***JMenuBar***, ***JMenu*** y ***JMenuItem***, dentro de un programa, se cuenta con dos alternativas:

1. Directamente en la plataforma de desarrollo, ya sea ***BlueJ***, ***NetBeans***, etc., basta con "arrastrar" los componentes requeridos hacia donde se requieran, ya que normalmente su operación es muy intuitiva.
2. Mediante la escritura del código adecuado, lo que resulta un poco retador.

Para afrontar el reto que representa escribir el código necesario para crear, y habilitar, las Clases ***JMenuBar***, ***JMenu*** y ***JMenuItem*** dentro de un programa, primero revisaremos un ejemplo cuya imagen se muestra en la ***Figura 4***:

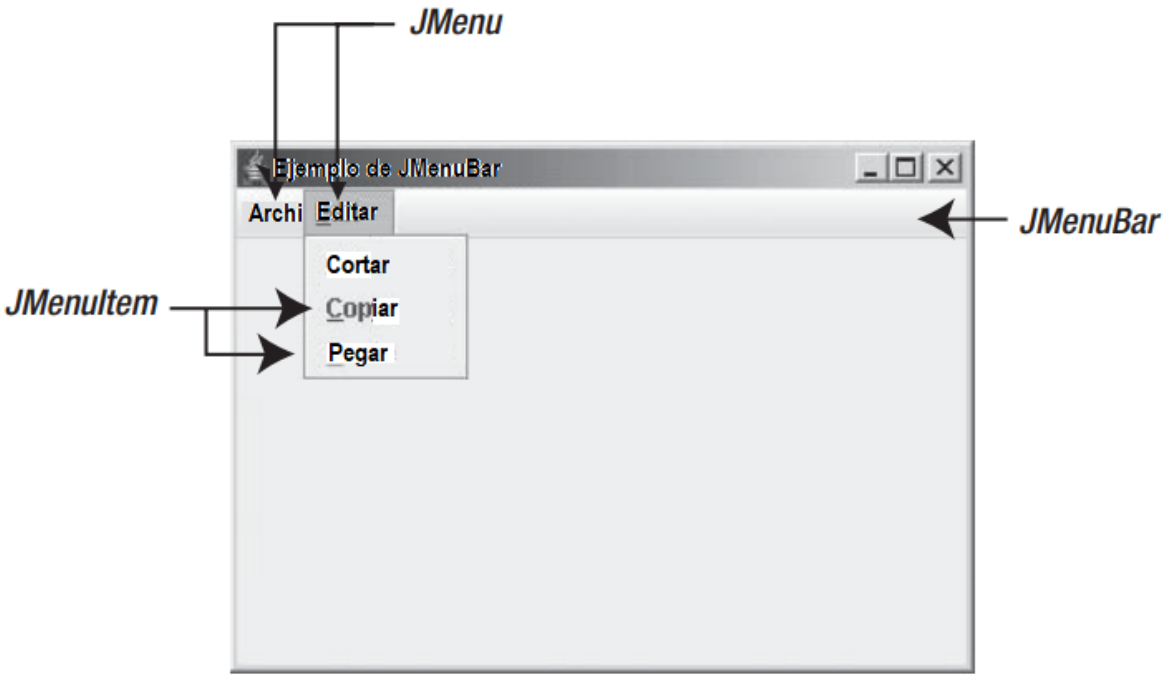


Figura 4. *Ejemplo de* ***`MenuBar`*** *(****`JMenuBar`****,* ***`JMenu`*** *y* ***`JMenuItem`****).*

Los **requerimientos**, el código y su **explicación**, respectivamente, son:

1. Crear una ventana básica **(*`JFrame`*)** con un tamaño de 400 300 píxeles, que será la ventana principal donde se muestra todo.
2. Agregar un menú **(*`JMenu`*)** de "Archivo" y otro de "Editar" en la barra de menú (***`JMenuBar`***).
3. Cada menú **(*`JMenu`*)** tiene varios elementos o entradas **(*`JMenuItem`*)**:
    1. Archivo: [**Abrir**], [**Guardar**] y [**Salir**].
    2. Editar: [**Cortar**], [**Copiar**] y [**Pegar**].
4. Asigna la barra de menú (***`JMenuBar`***) a la ventana **(*`JFrame`*)** y hace visible la interfaz.

El código correspondiente, se muestra a continuación:

```
//Importar las librerias de swing y awt que se van a utilizar.
import javax.swing.*;
import java.awt.event.*;
//import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import java.awt.Color;
public class Ventana extends JFrame
{
    private JMenuBar menuBar;

    //Constructor
    public Ventana()
    {
        // Se establece la pocición y titulo de la ventana
        setBounds(0,0,500,450);
        setTitle("Saludo");
        getContentPane().setBackground(Color.blue);
        setLocationRelativeTo(null);

        // Crear el JMenuBar
        menuBar = new JMenuBar();

        // Crear los menús
        JMenu archivoMenu = new JMenu("Archivo");
        JMenu editarMenu = new JMenu("Editar");

        // Crear los elementos del menú "Archivo"
        JMenuItem abrirItem = new JMenuItem("Abrir");
        JMenuItem guardarItem = new JMenuItem("Guardar");
        JMenuItem salirItem = new JMenuItem("Salir");
```

```
        // Agregar acciones a los elementos para la opción salir
        salirItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.exit(0);  // Cerrar la aplicación
            }
        });

        // Añadir los elementos al menú "Archivo"
        archivoMenu.add(abrirItem);
        archivoMenu.add(guardarItem);
        archivoMenu.addSeparator();  // Añadir un separador
        archivoMenu.add(salirItem);

        // Crear los elementos del menú "Editar"
        JMenuItem cortarItem = new JMenuItem("Cortar");
        JMenuItem copiarItem = new JMenuItem("Copiar");
        JMenuItem pegarItem = new JMenuItem("Pegar");

        // Añadir los elementos al menú "Editar"
        editarMenu.add(cortarItem);
        editarMenu.add(copiarItem);
        editarMenu.add(pegarItem);

        // Añadir los menús al JMenuBar
        menuBar.add(archivoMenu);
        menuBar.add(editarMenu);

        // Asignar el JMenuBar a la ventana
        setJMenuBar(menuBar);
    } // fin constructor

    public static void main(String args[]){
        //Se instancia el objeto ventana1
        Ventana ventana1 = new Ventana();
        //Se hace visible
        ventana1.setVisible(true);

    }
}
```

*Código del Ejemplo **Menu (`JMenuBar`, `JMenu` y `JMenuItem`).***

Considérese que, en esta interfaz gráfica, los resultados serán de acuerdo con la ejecución de los procesos, mediante la selección que se haga con el ratón (“mouse”) de la computadora.

Resultados:

En la **Tabla** siguiente se muestran los resultados que se generan al compilar y ejecutar el código anterior:

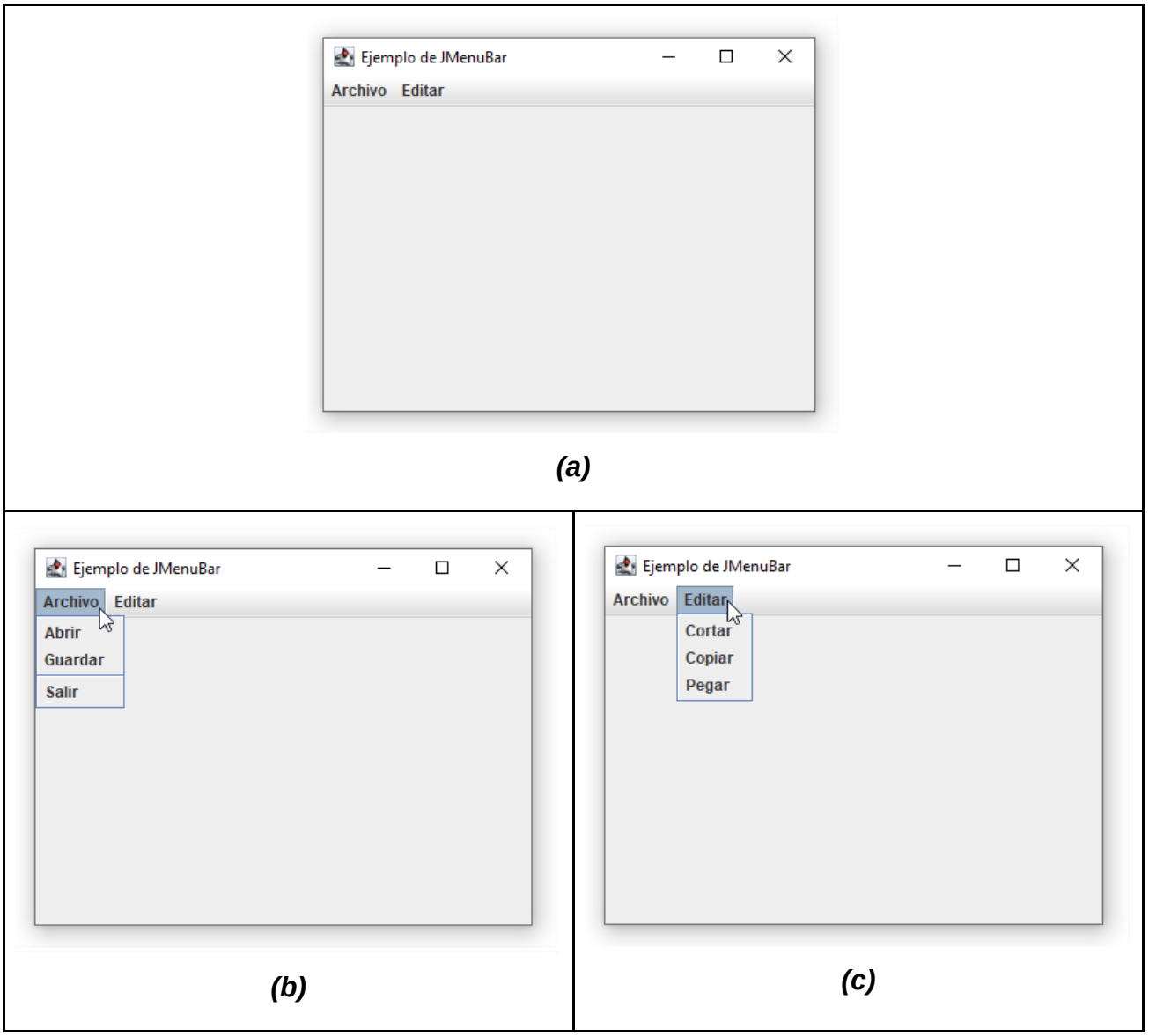


Tabla *Resultados del código* ***Menu (`JMenuBar`***, ***`JMenu`*** y ***`JMenuItem`***).

Inicialmente, aparece el resultado que se muestra en ***(a)***, y el sistema espera que se haga la elección con el puntero del ratón (“mouse”). Si se hace click en el menu ***Archivo***, la aplicación muestra el resultado que se muestra en ***(b)***; si por el contrario, se hace click en el menu ***Editar***, la aplicación muestra el resultado que se muestra en ***(c)***.

ÍNDICE

4.6.2 Autoevaluación del aprendizaje:

Ejercicio 1. Relación de columnas.

Instrucciones: Relaciona las clases con su definición correspondiente y selecciona la respuesta que muestra la correspondencia correcta.

CLASE	DEFINICIÓN
I. ***JMenuBar*** II. ***JMenu*** III. ***JMenuItem***	a) Elemento individual dentro de un menú, representa una opción seleccionable. b) Permite al usuario ejecutar una acción o un procedimiento. c) Es la barra en la que se alojan los menús d) Contenedor que agrupa varios ***JMenuItem*** y se muestra como un menú desplegable.

A) I:c - II:d - III:a

B) I:a - II:b - III:c

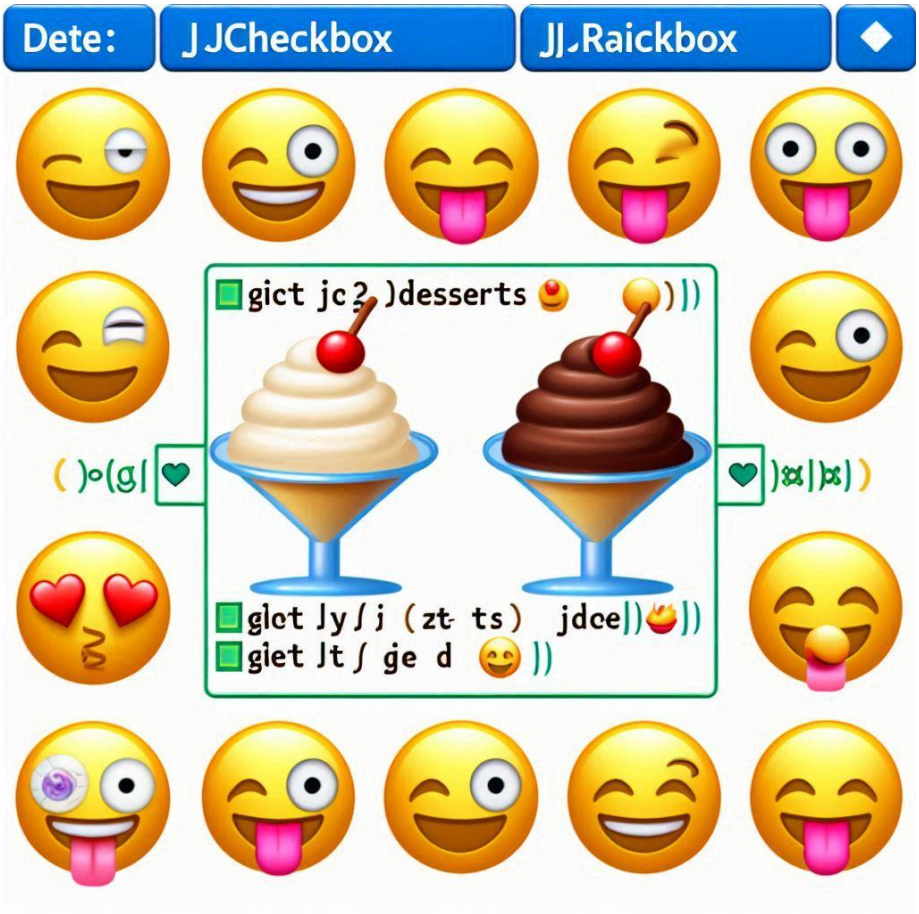
C) I:d - II:c - III:a

D) I:c - II:b - III:a

ÍNDICE

APRENDIZAJE 4.7

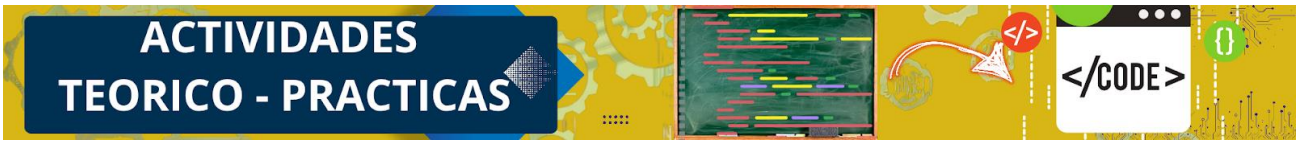
Elabora programas con interfaz gráfica de usuario, aplicando las Clases *JCheckBox* y *JRadioButton*.



Temática:

Clase ***Swing***:

- ***JCheckBox***
- ***JRadioButton***



ÍNDICE

4.7.1. Actividades de aprendizaje teórico–prácticas.

Clase *JCheckBox*.

La clase ***JCheckBox*** en ***Java*** pertenece al paquete ***javax.swing*** y se utiliza para crear casillas de verificación (“checkboxes”) en una ***Interfaz Gráfica de Usuario*** (***GUI***).

Características de *JCheckBox*.

- Tiene forma de cuadrito.
- Dado que puede estar en dos estados, seleccionado o no seleccionado, permite que el usuario seleccione simultáneamente varias opciones de manera independiente.
- Se pueden agrupar varios ***JCheckBox***, pero cada uno funciona de manera independiente.
- Es útil para opciones que pueden combinarse, como agregar ingredientes a un platillo o seleccionar preferencias en una configuración.

Ejemplo de uso de ***JCheckBox***.

```
// Crea una casilla con etiqueta "Opción 1":
JCheckBox checkBox = new JCheckBox("Opción 1");
// Establece la casilla como seleccionada por defecto:
checkBox.setSelected(true);
```

Clase *JRadioButton*.

La clase ***JRadioButton***, también es parte del paquete ***javax.swing***, se emplea para crear botones de opción (“radio buttons”) en una ***GUI***. A diferencia de ***JCheckBox***, los ***JRadioButton*** funcionan en grupos, lo que significa que solo una opción dentro del grupo puede estar seleccionada a la vez.

Características de ***JRadioButton***.

- Tiene forma de un circulito.
- Permite que el usuario elija sólo una única opción de varias posibles.
- Se pueden agrupar varios ***JRadioButton*** dentro de una clase ***ButtonGroup***.
- Es útil cuando se debe seleccionar sólo una opción, por ejemplo, en categorías como métodos de pago se debe seleccionar sólo una (efectivo, transferencia, tarjeta de crédito, etc.); en el género en un formulario (masculino, femenino u otro); o en tipos de postres en un menú (pastel, flan, etc.).

Ejemplo de uso de ***JRadioButton***:

```
// Crea dos JRadioButton, uno con etiqueta "Opción A" y otro con "Opción B":
JRadioButton opcionA = new JRadioButton("Opción A");
JRadioButton opcionB = new JRadioButton("Opción B");
// Agrupar los JRadioButton para que sólo se pueda seleccionar uno a la vez
ButtonGroup grupo = new ButtonGroup();
grupo.add(opcionA);
grupo.add(opcionB);
```

Diferencias entre *JCheckBox* y *JRadioButton*.

Característica	*JCheckBox*	*JRadioButton*
Selección	Puede haber varias opciones seleccionadas a la vez.	Solo una opción puede estar seleccionada dentro de un grupo.
Independencia	Cada casilla funciona de manera independiente.	Necesita agruparse dentro de un ***ButtonGroup*** para asegurar la selección única.
Uso común	Selección de múltiples opciones (ingredientes, preferencias).	Elección única entre varias opciones (género, tipo de pago, etc.).

Ejemplo combinando ***JCheckBox*** y ***JRadioButton***.

A continuación, se presenta un programa en Java que permite al usuario seleccionar un postre y agregar toppings opcionales.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class PostresApp {
    public static void main(String[] args) {
        // Crear ventana
        JFrame ventana = new JFrame("Selecciona tu postre");
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.setSize(400, 300);
        ventana.setLayout(new FlowLayout());

        // Etiqueta de postres
        JLabel etiquetaPostre = new JLabel("Elige un postre:");
        ventana.add(etiquetaPostre);

        // Botones de radio (Postres)
        JRadioButton helado = new JRadioButton("Helado");
        JRadioButton duraznos = new JRadioButton("Duraznos en almíbar");
        JRadioButton fresas = new JRadioButton("Fresas con crema");

        ButtonGroup grupoPostres = new ButtonGroup();
        grupoPostres.add(helado);
        grupoPostres.add(duraznos);
        grupoPostres.add(fresas);

        ventana.add(helado);
        ventana.add(duraznos);
        ventana.add(fresas);

        // Etiqueta de toppings
        JLabel etiquetaToppings = new JLabel("Elige tus toppings:");
        ventana.add(etiquetaToppings);
```

```
// Casillas de verificación (Toppings)
JCheckBox chispas = new JCheckBox("Chispas de chocolate");
JCheckBox nuez = new JCheckBox("Nuez");
JCheckBox lechera = new JCheckBox("Lechera");

ventana.add(chispas);
ventana.add(nuez);
ventana.add(lechera);

// Botón de confirmación
JButton btnConfirmar = new JButton("Confirmar Pedido");
ventana.add(btnConfirmar);

// Área de texto para mostrar selección
JTextArea resultado = new JTextArea(5, 30);
resultado.setEditable(false);
ventana.add(resultado);

// Acción del botón
btnConfirmar.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        String pedido = "Tu postre: ";

        if (helado.isSelected()) {
            pedido += "Helado";
        } else if (duraznos.isSelected()) {
            pedido += "Duraznos en almíbar";
        } else if (fresas.isSelected()) {
            pedido += "Fresas con crema";
        } else {
            pedido += "Ninguno seleccionado";
        }
```

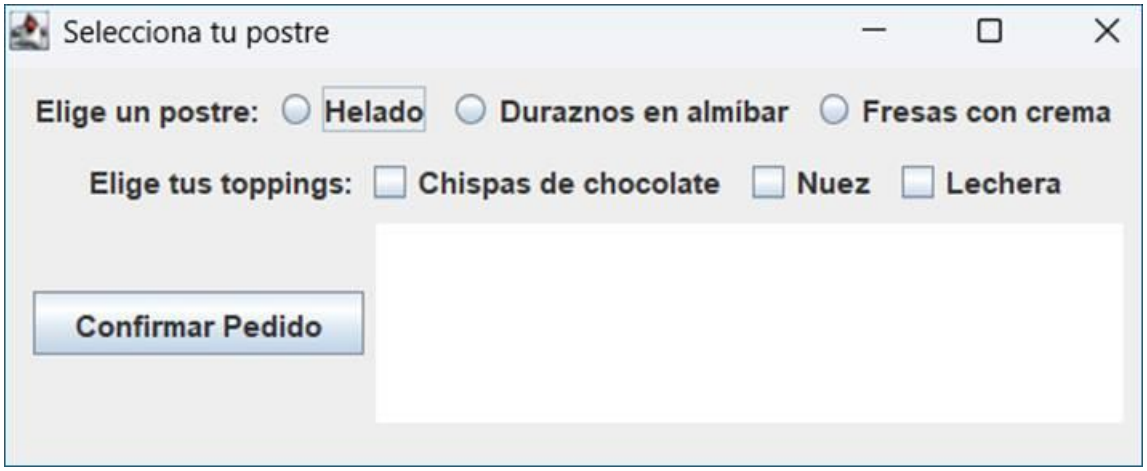
```
            pedido += "\nToppings: ";
            if (chispas.isSelected()) pedido += "Chispas de chocolate, ";
            if (nuez.isSelected()) pedido += "Nuez, ";
            if (lechera.isSelected()) pedido += "Lechera, ";

            // Eliminar la última coma si hay toppings seleccionados
            if (pedido.endsWith(", ")) {
                pedido = pedido.substring(0, pedido.length() - 2);
            }

            resultado.setText(pedido);
        }
    });

    // Hacer visible la ventana
    ventana.setVisible(true);
  }
}
```

Explicación del código.

- ***JRadioButton***: Permite al usuario elegir sólo un postre.
- ***ButtonGroup***: Asegura que sólo una opción de postre esté seleccionada.
- ***JCheckBox***: Permite seleccionar múltiples toppings sin restricciones.
- Interfaz gráfica con ***JFrame*** y ***JPanel***: Organiza visualmente los componentes.
- Botón de confirmación (***JButton***): Recoge la selección y la muestra en un ***JTextArea***.

Resultado:

En esta imagen el usuario no ha elegido su postre, ni sus toppings.



Ahora, ya hizo la selección del postre con 3 toppings.

ÍNDICE

4.7.2 Autoevaluación del aprendizaje:

Relaciona los elementos de la imagen con las clases.

Instrucciones: Clasifica los elementos de la imagen según su clase y elige la respuesta correcta.

Clases:

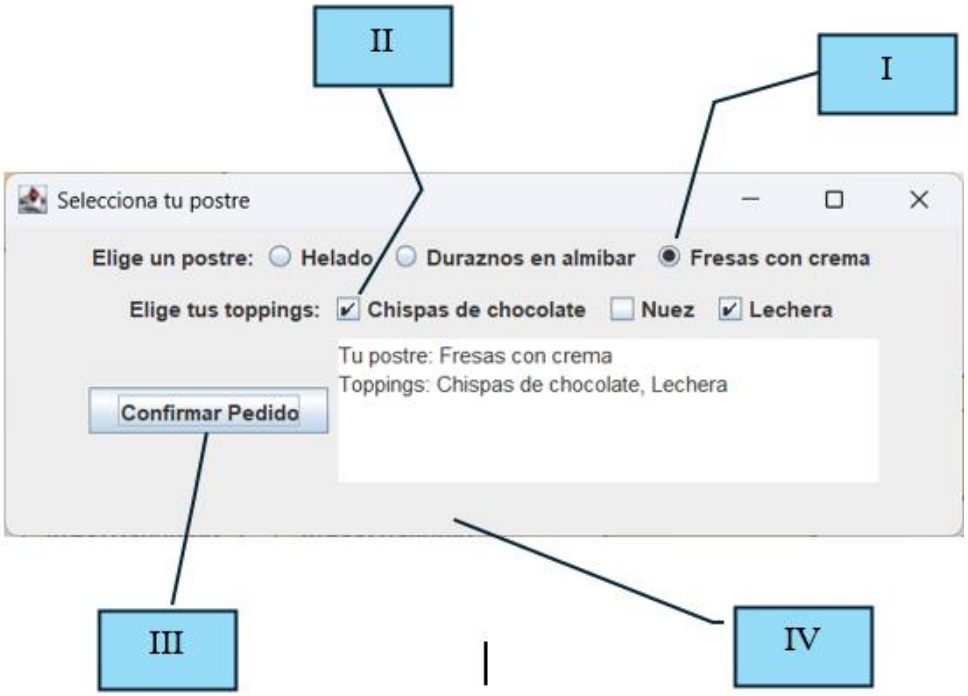
a) ***JButton***

b) ***JCheckBox***

c) ***JRadioButton***

d) ***JFrame***

e) ***JPanel***



Opciones de respuesta:

A) I:e - II:d - III:a - IV:b

B) I:a - II:b - III:e - IV:c

C) I:c - II:b - III:a - IV:d

D) I:a - II:b - III:e - IV:d

ÍNDICE

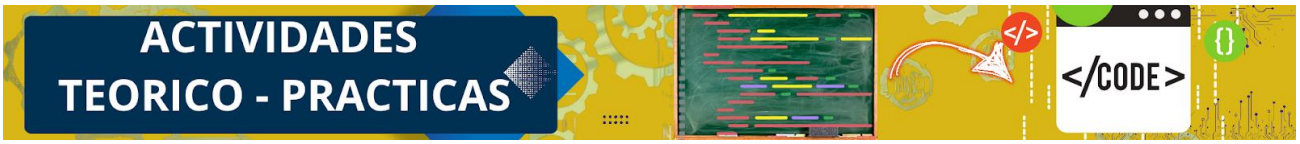
APRENDIZAJE 4.8

Elabora programas con interfaz gráfica de usuario, aplicando las Clases *setColor*, *drawLine*, *drawRect*, *drawRoundRect*, *drawOval* y *drawPolygon*.

Temática:

Clase ***Graphics***:

- ***setColor***
- ***drawLine***
- ***drawRect***
- ***drawRoundRect***
- ***drawOval***
- ***drawPolygon***



ÍNDICE

4.8.1. Actividades de aprendizaje teórico–prácticas.

En ***Java***, la clase ***Graphics*** nos permite dibujar formas en la pantalla. Para usar estos métodos, se necesita sobreescribir el método ***paint(Graphics g)*** en un ***JPanel*** o ***JFrame***. En la programación gráfica con Java ***Graphics***, las coordenadas en pantalla funcionan con un **sistema de coordenadas cartesianas invertidas**. Funciona de la siguiente manera:

- El punto (0,0) está en la esquina superior izquierda de la ventana.
- El eje X crece hacia la derecha y el eje Y crece hacia abajo.

```
(0,0) ——————————————————▸ X ➡ (Aumenta)
  |
  |
  |
  ▼ Y (Aumenta)
```

A Continuación, analizaremos los métodos más comunes para dibujar con la clase ***graphics*** en ***Java***:

drawLine(int x1, int y1, int x2, int y2)

Dibuja una línea recta desde un punto de inicio (x1, y1) hasta otro punto final (x2, y2), se usa para dividir áreas o como parte de dibujos más complejos, usa coordenadas en un sistema de ejes (x, y).

Ejemplo

```
g.drawLine(20, 30, 120, 30);
```

Dibuja una línea horizontal desde (20,30) hasta (120,30), se mantiene en el mismo nivel porque y1 = y2 = 30.

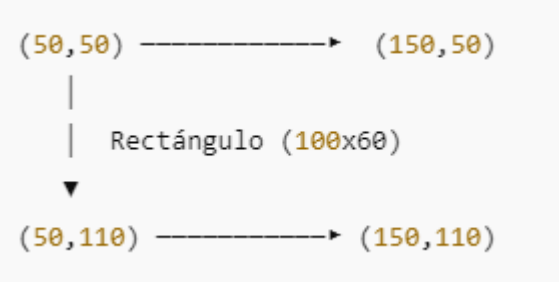
drawRect(int x, int y, int width, int height)

Dibuja un rectángulo con una posición inicial (x, y), con un ancho y una altura en pixeles dada por el usuario, se usa para representar cuadros, botones, marcos, etc., si ***width*** y ***height*** son iguales, se obtiene un cuadrado.

Ejemplo

```
g.drawRect(50, 50, 100, 60); // Rectángulo de 100x60 en (50,50)
```

Se dibuja un rectángulo en las coordenadas x, y (50,50), con un ancho de 100 píxeles y una altura de 60 píxeles.



drawRoundRect(int x,int y,int width,int height,int arcWidth,intarcHeight)

Dibuja un rectángulo con bordes redondeados, los valores ***arcWidth*** y ***arcHeight*** definen el redondeo de las esquinas, se usa en botones y marcos de interfaz gráfica.

Ejemplo

```
g.drawRoundRect(50, 130, 100, 60, 20, 20);
```

Coordenadas x, y (50,130) es la esquina superior izquierda, 100x60 define el tamaño del rectángulo, 20x20 hace que las esquinas sean más redondeadas.

drawOval(int x, int y, int width, int height)

Dibuja un óvalo dentro del rectángulo definido por ***(x, y, width, height)***, si ***width == height***, se dibuja un círculo, útil para gráficos como botones circulares o rostros.

Ejemplo

```
g.drawOval(50, 200, 80, 80);
```

Dibuja un círculo con centro aproximado en las coordenadas x, y (90,240), porque su rectángulo de referencia empieza en (50,200), con un diámetro de 80.

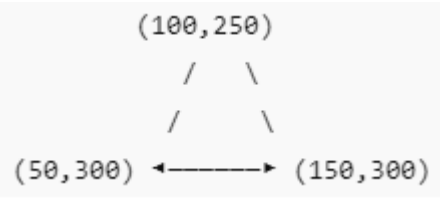
drawPolygon(int[] xPoints, int[] yPoints, int nPoints)

Dibuja un polígono conectando varios puntos, se usa para triángulos, estrellas, y figuras personalizadas, los arreglos ***xPoints*** y ***yPoints*** almacenan las coordenadas de X y Y respectivamente.

Ejemplo

```
int[] x = {50, 100, 150};
int[] y = {300, 250, 300};
g.drawPolygon(x, y, 3);
```

(50,300) → Primer vértice.
(100,250) → Segundo vértice (el más alto).
(150,300) → Tercer vértice.



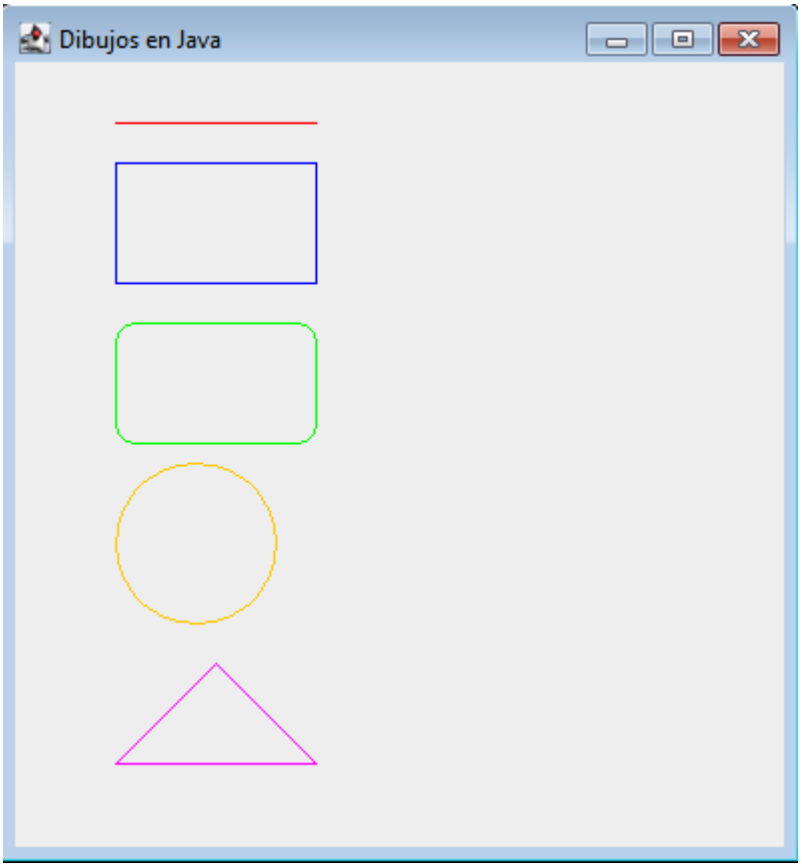
setColor(Color c)

Este método se usa para cambiar el color con el que se dibujan las formas, permite elegir cualquier color usando la clase Color, debe llamarse antes de dibujar para aplicar el color, se usa junto con métodos de dibujo (***drawLine***, ***drawRect***, etc.).

Ejemplo

```
g.setColor(Color.RED); // Cambia el color a rojo
g.drawLine(50, 50, 150, 50); // Dibuja una línea roja
```

El siguiente programa muestra **todos los métodos gráficos** explicados, y muestra en pantalla una línea color rojo, un rectángulo azul, un rectángulo con las esquinas redondeadas en color verde, un óvalo en color naranja y un polígono de tres vértices formando un triángulo en color magenta.



El código respectivo se muestra abajo.

En cada línea de código encontrarás comentarios explicando detalladamente su funcionamiento.

```java
// Importamos las librerías necesarias para la interfaz gráfica
import javax.swing.*;  // Para crear la ventana y el panel
import java.awt.*;     // Para dibujar figuras y manejar colores

// Creamos una clase que extiende JPanel para poder dibujar en ella
class DibujoPanel extends JPanel {

    // Sobreescribimos el método paintComponent para dibujar
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g); // Limpia la pantalla antes de dibujar

        // 🔹 1. Establecemos el color y dibujamos una línea
        g.setColor(Color.RED); // Cambia el color de la línea a rojo
        g.drawLine(50, 30, 150, 30); // Dibuja una línea desde coordenadas (x=50,y=30) hasta (x=120,y=30)

        // 🔹 2. Dibujamos un rectángulo azul
        g.setColor(Color.BLUE); // Cambia el color a azul
        g.drawRect(50, 50, 100, 60); // Rectángulo en (x=50,y=50) con tamaño 100Ancho x 60Alto

        // 🔹 3. Dibujamos un rectángulo con esquinas redondeadas
        g.setColor(Color.GREEN); // Color verde
        g.drawRoundRect(50, 130, 100, 60, 20, 20); // rectángulo redondeado en (x=50,y=130) con tamaño 100Ancho x 60Alto
                                                   // Bordes redondeados con radio de 20px
        // 🔹 4. Dibujamos un óvalo (círculo)
        g.setColor(Color.ORANGE); // Color naranja
        g.drawOval(50, 200, 80, 80); // Círculo de 80px de diámetro en (x=50,y=200)

        // 🔹 5. Dibujamos un polígono (triángulo)
        g.setColor(Color.MAGENTA); // Color magenta
        int[] x = {50, 100, 150}; // Coordenadas en X de los puntos
        //int[] y = {300, 250, 300}; // Coordenadas en Y de los puntos
        int[] y = {350, 300, 350}; // Coordenadas en Y de los puntos
        g.drawPolygon(x, y, 3); // Dibuja un triángulo con 3 puntos
    }
}

// Creamos la clase principal con el método main
public class DibujoFiguras {
    public static void main(String[] args) {
        // 🔹 6. Creamos una ventana (JFrame)
        JFrame ventana = new JFrame("Dibujos en Java"); // Nombre de la ventana
        DibujoPanel panel = new DibujoPanel(); // Creamos un objeto de nuestra clase personalizada

        ventana.add(panel); // Agregamos el panel a la ventana
        ventana.setSize(400, 430); // Establecemos el tamaño de la ventana Ancho=400 Alto=430
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Cierra el programa al cerrar la ventana
        ventana.setVisible(true); // Hacemos visible la ventana
    }
}
```

ÍNDICE

4.8.2 Autoevaluación del aprendizaje:

Relación de columnas.

Instrucciones: En la siguiente tabla, se presentan métodos de la clase Graphics en Java utilizados para dibujar figuras y modificar colores en pantalla. Relaciónalos con su descripción correcta y selecciona la respuesta que muestra la correspondencia correcta.

Método	Descripción
I. ***setColor***	a) Dibuja un polígono utilizando un conjunto de coordenadas ***(x, y)*** para definir sus vértices.
II. ***drawLine***	b) Dibuja un óvalo dentro de un rectángulo delimitador definido por sus coordenadas y dimensiones.
III. ***drawRect***	c) Dibuja un rectángulo con esquinas redondeadas, especificando el ancho y alto del redondeo.
IV. ***drawRoundRect***	d) Dibuja un rectángulo relleno con el color actual, especificando su posición, ancho y alto.
V. ***drawOval***	e) Dibuja una línea recta entre dos puntos dados por coordenadas ***(x1, y1)*** y ***(x2, y2)***.
VI. ***drawPolygon***	f) Cambia el color de dibujo a un color especificado.
	g) Dibuja un rectángulo con un ancho y alto especificados a partir de un punto ***(x, y)***.

A) I:b - II:g - III:c - IV:a - V:b - VI:d

B) I:a - II:c - III:b - IV:c - V:d- VI:g

C) I:f - II:e - III:g - IV:c - V:b - VI:a

D) I:c - II:d - III: c - IV:g - V:b - VI:a

ÍNDICE

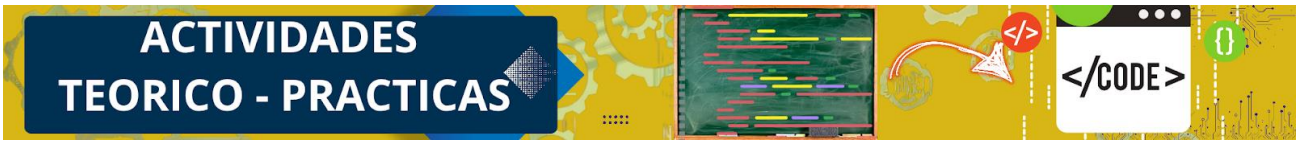
APRENDIZAJE 4.9

Elabora programas con interfaz gráfica de usuario, aplicando las Clases *fillRect*, *fillRoundRect*, *fillOval* y *fillPolygon*.

Temática:

Clase ***Graphics***:

- ***fillRect***
- ***fillRoundRect***
- ***fillOval***
- ***fillPolygon***



ÍNDICE

4.9.1. Actividades de aprendizaje teórico–prácticas.

Otra de las bondades que ofrece ***java***, en su interfaz gráfica, es que se pueden dibujar formas geométricas rellenas, lo que implica utilizar, de la clase ***java.awt.Graphics***, los métodos que comienzan con ***fill***, por ejemplo:

- Un rectángulo relleno (***fillRect***).
- Un rectángulo relleno con sus esquinas redondeadas (***fillRoundRect***).
- Un óvalo relleno (***fillOval***).
- Un polígono relleno (***fillPolygon***).

Entonces, cabe resaltar que, en ***Java***, para poder crear los componentes gráficos ***fillRect***, ***fillRoundRect***, ***fillOval*** y ***fillPolygon***, se debe utilizar la clase ***Graphics*** dentro de un componente ***JPanel*** o ***Canvas*** (Ingteleco, s/f).

El ejemplo siguiente utiliza un ***JPanel*** y sobrescribe el método ***paintComponent*** para dibujar un rectángulo relleno, un rectángulo relleno con esquinas redondeadas, un óvalo y un polígono rellenos:

Los **requerimientos**, el código y su **explicación**, respectivamente, son:

1. Dibujar un rectángulo relleno **(*fillRect*)**.
2. Dibujar un rectángulo relleno redondeado **(*fillRoundRect*)**.
3. Dibujar un óvalo relleno **(*fillOval*)**.
4. Dibujar un polígono relleno **(*fillPolygon*)**.

El código correspondiente, se muestra a continuación:

```java
import javax.swing.*;
import java.awt.*;
public class DibujarFiguras extends JPanel {
    // Sobrescribir el método paintComponent para dibujar las figuras
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        // Convertir Graphics a Graphics2D para un mejor control
        Graphics2D g2d = (Graphics2D) g;
        // Establecer el color de relleno
        g2d.setColor(Color.BLUE);
        // 1. Dibujar un rectángulo relleno
        g2d.fillRect(50, 50, 150, 100); // (x, y, ancho, alto)
        // 2. Dibujar un rectángulo redondeado relleno
        g2d.fillRoundRect(250, 50, 150, 100, 30, 30);
                    // ( x,   y, ancho, alto, radio de las esquinas)
        // 3. Dibujar un óvalo relleno
        g2d.fillOval(50, 200, 150, 100);  // (x, y, ancho, alto)
        // 4. Dibujar un polígono relleno
        int[] xPoints = {250, 320, 450};
        int[] yPoints = {300, 200, 250};
        g2d.fillPolygon(xPoints, yPoints, 3);  // Arrays de puntos y número de vértices
    }

    public static void main(String[] args) {
        // Crear el JFrame y agregar el panel
        JFrame frame = new JFrame("Dibujar Figuras RELLENAS");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(500, 400);
        frame.add(new DibujarFiguras());
        frame.setVisible(true);
    }
}
```

Código del Ejemplo ***fill (fillRect***, ***fillRoundRect***, ***fillOval*** y ***fillPolygon***).

Explicación:

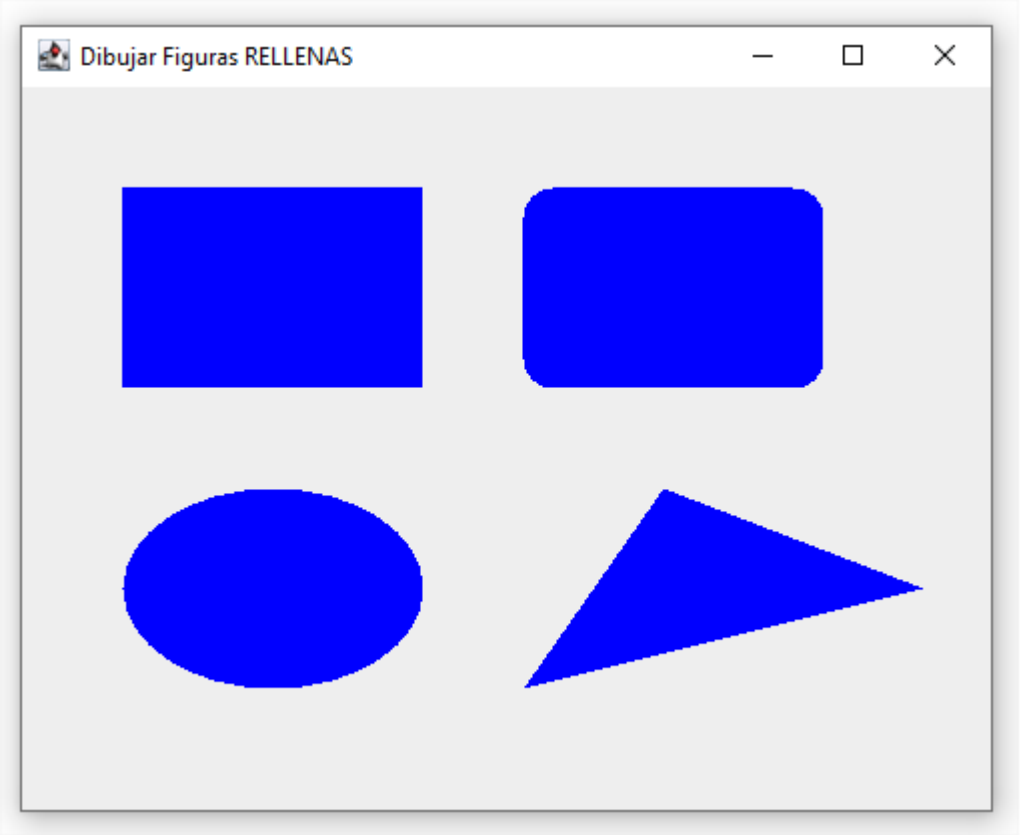
1. ***fillRect(x, y, width, height)***: Dibuja un rectángulo relleno con las coordenadas ***(x, y)*** como la esquina superior izquierda y con el tamaño especificado por ***width*** (ancho) y ***height*** (alto).
2. ***fillRoundRect(x, y, width, height, arcWidth, arcHeight)***: Dibuja un rectángulo redondeado relleno. Además de las coordenadas y el tamaño, debes especificar el radio de las esquinas con ***arcWidth*** y ***arcHeight***.
3. ***fillOval(x, y, width, height)***: Dibuja un óvalo relleno dentro de un rectángulo de las mismas dimensiones que ***width*** y ***height***.

4. ***fillPolygon(xPoints, yPoints, nPoints)***: Dibuja un polígono relleno con los puntos dados en los arreglos ***xPoints*** y ***yPoints***. ***nPoints*** es el número de puntos (vértices) que forman el polígono.

Resultados:

El código crea una ventana con varias figuras rellenas (un rectángulo, un rectángulo redondeado, un óvalo y un polígono) dibujadas en color azul.

A continuación, se muestran los resultados que se generan al compilar y ejecutar el código anterior:



Resultados del código ***fill*** *(****fillRect***, ***fillRoundRect***, ***fillOval*** y ***fillPolygon***).

Si deseas cambiar el color de las figuras o las coordenadas, solo necesitas ajustar los valores en las llamadas de los métodos correspondientes.

ÍNDICE

4.9.2 Autoevaluación del aprendizaje:

Relación de columnas.

Instrucciones: En la siguiente tabla, se presentan métodos de la clase ***Graphics*** en ***Java*** utilizados para dibujar figuras rellenas. Relaciónalos con su descripción correcta y selecciona la respuesta que muestra la correspondencia correcta.

MÉTODO	DESCRIPCIÓN
I. ***fillRect*** II. ***fillRoundRect*** III. ***fillOval*** IV. ***fillPolygon***	a) Dibuja un rectángulo con un ancho y alto especificados a partir de un punto ***(x, y)***. b) Dibuja un óvalo relleno dentro de un rectángulo delimitador definido por sus coordenadas y dimensiones. c) Dibuja un rectángulo con esquinas redondeadas relleno, especificando el ancho y alto del redondeo. d) Dibuja un rectángulo relleno con el color actual, especificando su posición, ancho y alto. e) Dibuja un polígono relleno utilizando un conjunto de coordenadas (x, y) para definir sus vértices.

A) I:d - II:c: - III:b - IV:e

B) I:a - II:b: - III:c - IV:d

C) I:e - II:c: - III:b - IV:a

D) I:b - II:e: - III:d - IV:c

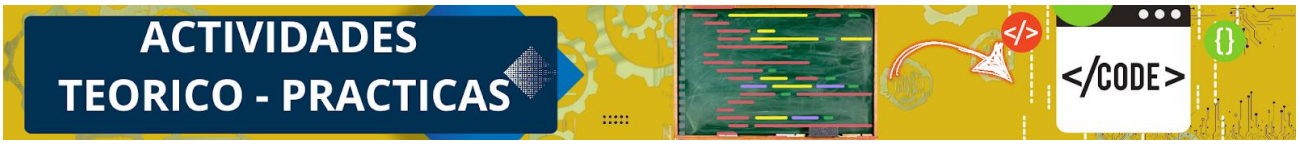
ÍNDICE

APRENDIZAJE 4.10

Propone un proyecto que utilice las Clases: *setColor*, *drawLine*, *drawRect*, *drawRoundRect*, *drawOval*, *drawPolygon*, *fillRect*, *fillRoundRect*, *fillOval* y *fillPolygon*.

Temática:

Preparación del proyecto.



ÍNDICE

4.10.1. Actividades de aprendizaje teórico–prácticas.

El programa que analizaremos hoy crea una tableta de dibujo interactiva utilizando ***Java Swing***. Permite dibujar líneas en una ventana moviendo un punto con las flechas del teclado y tiene un botón para borrar el dibujo.

Importación de Librerías

El código comienza importando las librerías necesarias:

javax.swing.*: Para crear la ***interfaz gráfica***. ***java.awt.****: Para manejar gráficos y diseño. ***java.awt.event.****: Para manejar eventos del teclado y del botón. ***java.util.ArrayList***: Para almacenar los puntos dibujados.
extends JPanel: La clase hereda de ***JPanel***, lo que permite dibujar en la ventana. ***implements KeyListener***: Permite detectar las teclas que presiona el usuario.

Variables Principales

x y ***y***: Representan la posición del punto que se dibuja en la pantalla. ***STEP***: Determina cuántos píxeles se mueve el cursor por cada pulsación de tecla. ***points***: Es una lista donde se guardan las coordenadas de los puntos dibujados.

¿Qué hace este código?

1. Crea una ventana (***JFrame***) con el título **"Tableta de Dibujo"**.
2. Crea una instancia de la clase DibujoTableta (el área donde se dibuja).
3. Crea un botón "Limpiar" y le asigna una acción para borrar el dibujo.
4. Organiza los elementos en la ventana:

1. El panel de dibujo está en el centro (***BorderLayout.CENTER***).
2. El botón está en la parte inferior (***BorderLayout.SOUTH***).

5. Define el tamaño de la ventana (500x500 píxeles).
6. Hace visible la ventana con ***frame.setVisible(true)***.

Método ***paintComponent(Graphics g)*** (Dibujo en la Pantalla)

¿Qué hace este método?

- ***super.paintComponent(g)***; limpia la pantalla antes de dibujar.
- ***g.setColor(Color.BLACK)***; establece el color de dibujo en negro.
- Luego, se recorre la lista de puntos (***points***) y se dibuja una línea entre cada par de puntos consecutivos con ***g.drawLine(p1.x, p1.y, p2.x, p2.y)***

Métodos Vacíos de KeyListener

keyReleased(): Se activa cuando el usuario suelta una tecla, pero no lo usamos.

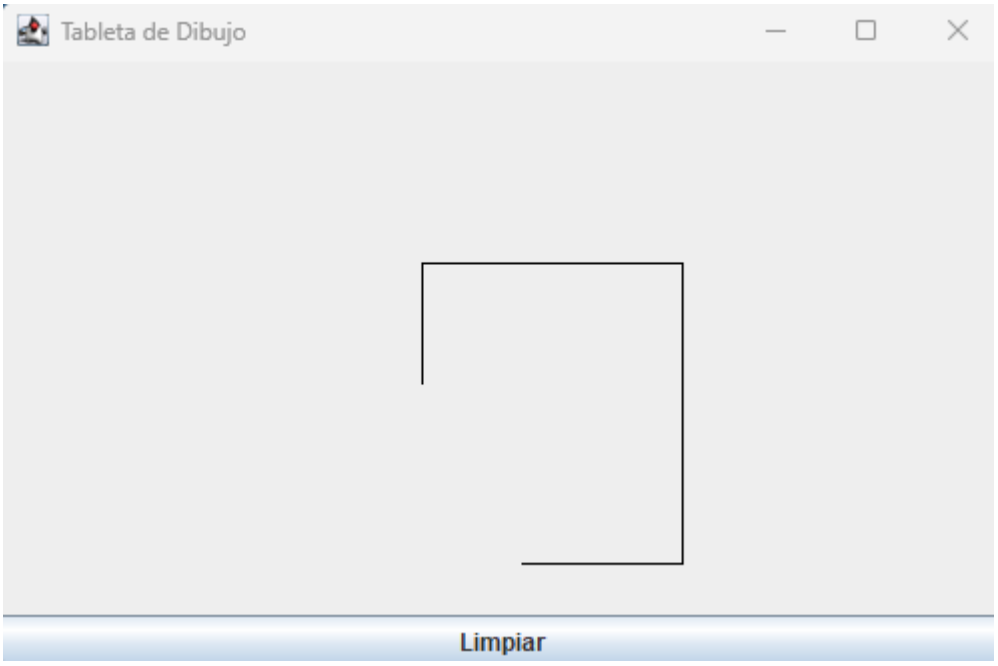
keyTyped(): Se activa cuando el usuario escribe un carácter, pero tampoco lo usamos.

*Dejamos estos métodos vacíos porque **Java** nos obliga a implementarlos al usar **KeyListener**.*

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;

public class DibujoTableta extends JPanel implements KeyListener {
    private int x = 250, y = 250; // Posición inicial del cursor
    private final int STEP = 10; // Tamaño del paso
    private ArrayList<Point> points = new ArrayList<>(); // Lista de puntos dibujados

    public static void main(String[] args) {
        JFrame frame = new JFrame("Tableta de Dibujo");
        DibujoTableta panel = new DibujoTableta();

        JButton clearButton = new JButton("Limpiar");
        clearButton.addActionListener(e -> panel.limpiarPantalla());

        frame.setLayout(new BorderLayout());
        frame.add(panel, BorderLayout.CENTER);
        frame.add(clearButton, BorderLayout.SOUTH);

        panel.setPreferredSize(new Dimension(500, 500));
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }

    public DibujoTableta() {
        setFocusable(true);
        addKeyListener(this);
    }
```

```
    @Override //sobrescritura de métodos
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.BLACK);
        for (int i = 1; i < points.size(); i++) {
            Point p1 = points.get(i - 1);
            Point p2 = points.get(i);
            g.drawLine(p1.x, p1.y, p2.x, p2.y);
        }
    }

    @Override //sobrescritura de métodos
    public void keyPressed(KeyEvent e) {
        int keyCode = e.getKeyCode();
        if (keyCode == KeyEvent.VK_UP) {
            y -= STEP;
        } else if (keyCode == KeyEvent.VK_DOWN) {
            y += STEP;
        } else if (keyCode == KeyEvent.VK_LEFT) {
            x -= STEP;
        } else if (keyCode == KeyEvent.VK_RIGHT) {
            x += STEP;
        }
        points.add(new Point(x, y));
        repaint();
    }

    public void limpiarPantalla() {
        points.clear();
        x = 250;
        y = 250;
        repaint();
    }

    @Override //sobrescritura de métodos
    public void keyReleased(KeyEvent e) {}

    @Override //sobrescritura de métodos
    public void keyTyped(KeyEvent e) {}

}
```

Resultado



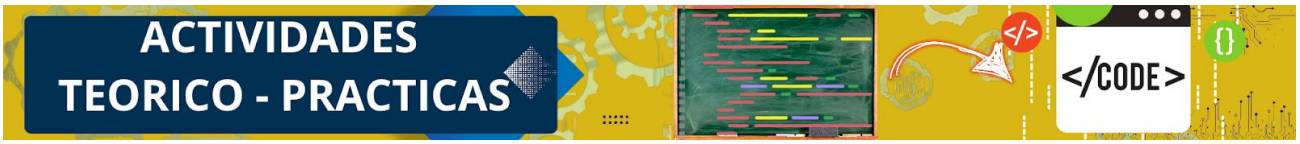
ÍNDICE

APRENDIZAJE 4.11

Desarrolla un proyecto que integre las Clases estudiadas en esta unidad.

Temática:

Desarrollo de un proyecto.



ÍNDICE

4.11.1. Actividades de aprendizaje teórico–prácticas.

El programa implementa una máquina expendedora simulada en ***Java*** utilizando ***Swing*** para la ***interfaz gráfica***. Su funcionamiento permite al usuario ingresar dinero mediante botones de diferentes denominaciones ($1, $2, $5, $10) y seleccionar productos para comprar.

Comportamiento del Código

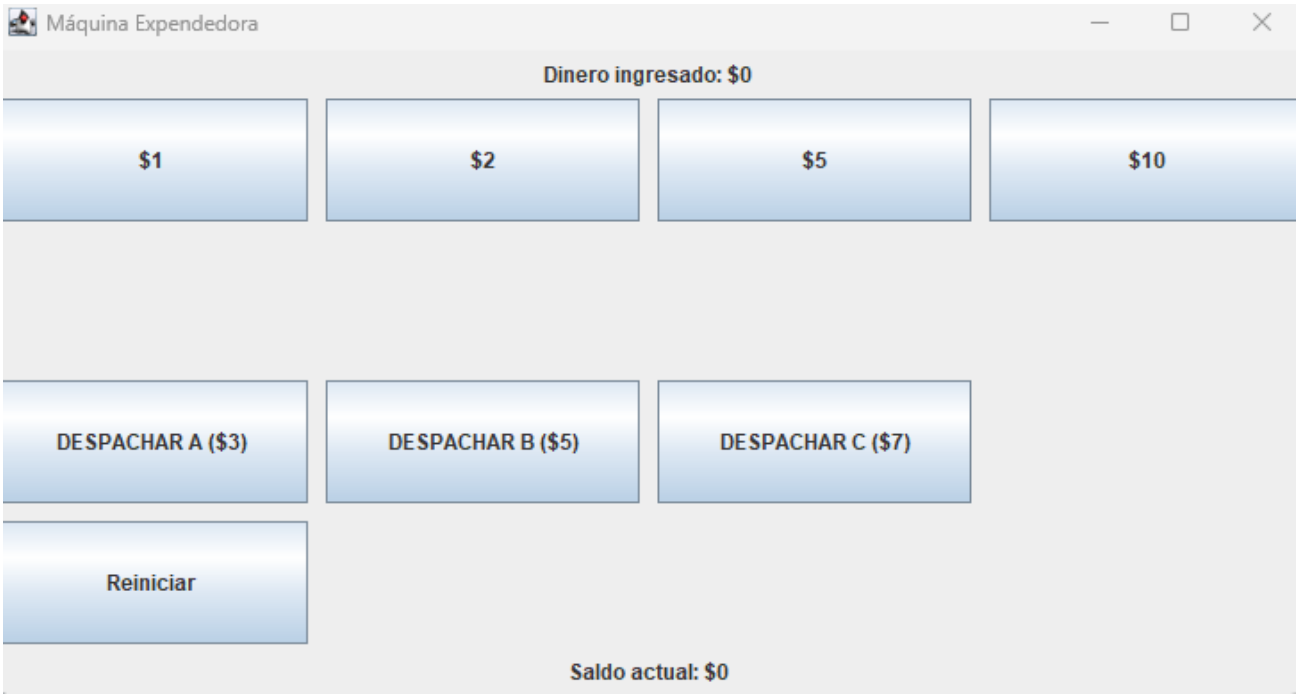
1. ***Interfaz gráfica*****:** Se compone de un panel con distribución en tres secciones principales:
    1. Un ***label*** superior muestra la cantidad de dinero ingresado.
    2. Un ***panel*** central con un diseño de cuadrícula 3x3 que contiene botones para ingresar monedas y seleccionar productos.
    3. Un ***label*** inferior que indica el saldo actual después de cada compra.
2. **Ingreso de dinero**:
    1. Al presionar un botón de moneda, el programa suma la cantidad correspondiente al saldo total.
    2. Si el usuario intenta ingresar más de $10, se muestra un mensaje de error.
3. **Compra de productos**:
    1. Existen tres botones etiquetados como "DESPACHAR A ($3)", "DESPACHAR B ($5)" y "DESPACHAR C ($7)".
    2. Si el saldo es suficiente, la compra se efectúa y el saldo se descuenta automáticamente.
    3. Si el saldo es insuficiente, se muestra un mensaje de advertencia.
4. **Reinicio de saldo**:
    1. Se incluye un botón "Reiniciar", que devuelve el saldo a $0.

Este código es útil para simular el funcionamiento de una máquina expendedora básica y puede servir como base para proyectos más avanzados, agregando funcionalidades como más productos, animaciones o una lógica más compleja de pago.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class MaquinaExpendedora extends JPanel {
    private int dineroIngresado = 0;
    private JLabel labelDinero;
    private JLabel labelSaldo;

    public static void main(String[] args) {
        JFrame frame = new JFrame("Máquina Expendedora");
        MaquinaExpendedora panel = new MaquinaExpendedora();
        frame.add(panel);
        frame.setSize(750, 400); // Ajuste del tamaño de la ventana
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }

    public MaquinaExpendedora() {
        setLayout(new BorderLayout());
        setBackground(Color.LIGHT_GRAY);

        // Panel superior (mostrar saldo)
        JPanel panelSuperior = new JPanel();
        labelDinero = new JLabel("Dinero ingresado: $0");
        panelSuperior.add(labelDinero);
        add(panelSuperior, BorderLayout.NORTH);

        // Panel central con un GridLayout de 4x4 para alinear los botones
        JPanel panelBotones = new JPanel();
        panelBotones.setLayout(new GridLayout(4, 4, 10, 10)); // 4x4 con espacio entre botones

        // Agregar botones de monedas
        agregarBotonMoneda(panelBotones, "$1", 1);
        agregarBotonMoneda(panelBotones, "$2", 2);
        agregarBotonMoneda(panelBotones, "$5", 5);
        agregarBotonMoneda(panelBotones, "$10", 10);
```

```
        // Espacios vacíos para mantener la alineación en el grid
        panelBotones.add(new JLabel(""));
        panelBotones.add(new JLabel(""));
        panelBotones.add(new JLabel(""));
        panelBotones.add(new JLabel(""));

        // Agregar botones de productos alineados
        agregarBotonProducto(panelBotones, "DESPACHAR A ($3)", 3);
        agregarBotonProducto(panelBotones, "DESPACHAR B ($5)", 5);
        agregarBotonProducto(panelBotones, "DESPACHAR C ($7)", 7);
        panelBotones.add(new JLabel("")); // Espacio vacío para ajuste

        // Botón de reinicio alineado en el grid
        JButton btnReiniciar = new JButton("Reiniciar");
        btnReiniciar.addActionListener(e -> reiniciarDinero());
        panelBotones.add(btnReiniciar);
        panelBotones.add(new JLabel("")); // Espacios vacíos para ajuste visual
        panelBotones.add(new JLabel(""));
        panelBotones.add(new JLabel(""));

        add(panelBotones, BorderLayout.CENTER);

        // Panel inferior para mostrar saldo restante
        JPanel panelSaldo = new JPanel();
        labelSaldo = new JLabel("Saldo actual: $0");
        panelSaldo.add(labelSaldo);
        add(panelSaldo, BorderLayout.SOUTH);
    }

    private void agregarBotonMoneda(JPanel panel, String texto, int valor) {
        JButton btn = new JButton(texto);
        btn.addActionListener(e -> agregarDinero(valor));
        panel.add(btn);
    }
```

Resultado


Máquina Expendedora
Dinero ingresado: $0
$1
$2
$5
$10
DESPACHAR A ($3)
DESPACHAR B ($5)
DESPACHAR C ($7)
Reiniciar
Saldo actual: $0

ÍNDICE

4.12 RESPUESTAS A LAS AUTOEVALUACIONES.

<table>
<tr><th>AUTOEVALUACIÓN</th><th>RESPUESTA CORRECTA</th></tr>
<tr><td>4.1.2</td><td>EJERCICIO 4.1 INTERFAZ:
<table><tr><td>a)</td><td>Cualquier botón, control, interruptor que sirva para controlar el movimiento del auto (llaves de encendido, control de los limpiadores, botón de cristales, etc.)</td></tr><tr><td>b)</td><td>El usuario.</td></tr><tr><td>c)</td><td>El entorno o área de trabajo de la aplicación que, generalmente, se compone de menús, botones, frames, cajas de texto, casillas de opciones, etc.</td></tr></table>
Cuestionario: Interfaz Gráfica de Usuario (GUI).

C) I:a - II:b - III:b - IV:c - V:c</td></tr>
<tr><td>4.2.2</td><td>Sopa de letras: Palabras clave.
<table><tr><td>N</td><td>*</td><td>*</td><td>*</td><td>*</td><td>S</td><td>G</td><td>S</td><td>*</td><td>*</td></tr><tr><td>E</td><td>*</td><td>*</td><td>*</td><td>*</td><td>E</td><td>R</td><td>E</td><td>*</td><td>N</td></tr><tr><td>X</td><td>*</td><td>*</td><td>*</td><td>T</td><td>T</td><td>E</td><td>T</td><td>R</td><td>E</td></tr><tr><td>T</td><td>N</td><td>*</td><td>T</td><td>*</td><td>T</td><td>N</td><td>V</td><td>O</td><td>X</td></tr><tr><td>D</td><td>*</td><td>E</td><td>*</td><td>*</td><td>E</td><td>N</td><td>A</td><td>L</td><td>T</td></tr><tr><td>O</td><td>R</td><td>*</td><td>X</td><td>*</td><td>R</td><td>A</td><td>L</td><td>A</td><td>L</td></tr><tr><td>U</td><td>*</td><td>*</td><td>*</td><td>T</td><td>*</td><td>C</td><td>O</td><td>V</td><td>I</td></tr><tr><td>B</td><td>*</td><td>*</td><td>*</td><td>*</td><td>I</td><td>S</td><td>R</td><td>T</td><td>N</td></tr><tr><td>L</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>N</td><td>*</td><td>E</td><td>E</td></tr><tr><td>E</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>T</td><td>G</td><td>*</td></tr></table></td></tr>
<tr><td>4.3.2</td><td>Relacionar columnas: Clase - Imágenes.
B) 1:c - 2:a - 3:b</td></tr>
</table>

4.4.2	Completar espacio en blanco, del código. D) a:4 - b:3 - c:2
4.6.2	Relaciona columnas: Clase/Definición. A) I:c - II:d - III:a
4.7.2	Relacionar Elementos-Imagen: C) I:c - II:b - III:a - IV:d
4.8.2	Relacionar Columnas: Métodos/Descripción. C) I:f - II:e - III: g - IV:c - V:b - VI:a
4.9.2	Relacionar Columnas: Métodos (***fill***)/Descripción. A) I:d - II:c: - III:b - IV:e

ÍNDICE

4.13 REFERENCIAS.

Corcuera, P. (s/f). Interfaces de usuario con AWT y Swing. España. Recuperado el 16 de febrero 2025, de https://personales.unican.es/corcuerp/java/Slides/GUIs_AWT_Swing.pdf

Dean, J., Et al. (2009). Introducción a la programación con Java. 1a Ed. Ed. Mc Graw Hill. CDMX, México. Recuperado el 5 de febrero 2025, de https://portafoliodeevidenciasderick.weebly.com/uploads/2/5/4/9/25498533/introduccion_a_la_programacion_con_java.pdf

IA, Canva. (S/F). Plataformas para generar las imágenes. Recuperado el 12 de marzo 2025, de https://www.canva.com/ y https://copilot.microsoft.com/onboarding

Ingteleco. (S/F). Apuntes de Java - Tema 7: AWT. Recuperado el 31 de enero 2025, de https://triunfadorsucre2010.wordpress.com/wp-content/uploads/2011/11/tema-7-awt.pdf

Joyanes, L. (2011). Programación en Java. Mc. Graw Hill. CDMX, México. Recuperado el 13 de febrero, 225, de https://ebookcentral.proquest.com/lib/bibliodgbmhe/reader.action?docID=3215489&ppg=5

Luna, L. (2004). El diseño de interfaz gráfica de usuario para publicaciones digitales. Revista Digital Universitaria. CDMX, México. Recuperado el 31 de enero, 225, de https://www.revista.unam.mx/vol.5/num7/art44/ago_art44.pdf

LaTorre, A. (s/f). GUIs en Java. UPM. España. Recuperado el 16 de febrero 2025, de https://laurel.datsi.fi.upm.es/_media/docencia/cursos/java/2012/guis_en_java-1pp_2012_.pdf

Malik, D. (2013). Programación Java - Del análisis de problemas al diseño de programas. 5a. ed. Cengage, Learning. CDMX, México. Recuperado el 31 de enero 2025, de http://ebookcentral.proquest.com/lib/bibliodgbsp/detail.action?docID=3430513

Schildt, H. (207). Fundamentos de Java. 3a. Ed. Mac Graw Hill. Cdmx. México. Recuperado el 13 de febrero 2025, de https://maycolmo.github.io/pe/java.pdf

Sznajdleder, P. (2013). Java a fondo - Estudio del lenguaje y desarrollo de aplicaciones. Ed. Alfa-Omega. Buenos Aires, Argentina. Recuperado el 31 de enero 2025, de https://unam-bibliotecasdigitales-com.pbidi.unam.mx:2443/read/9786077079064/index

UNAM-CCH. (2016). Programa de Estudios - Área de Matemáticas - Cibernética y Computación II. 1a. Ed. UNAM - CCH. México. Recuperado el 16 de febrero 2025, de https://www.cch.unam.mx/sites/default/files/programas2016/CIBERNETICA_COMPUTACION_I_II.pdf

Zukowski, J. (2005). La guía definitiva para Java Swing. 3a. Apress. EUA. Recuperado el 16 de febrero 2025, de https://www.cs.buap.mx/~ygalicia/Libros/JavaSwing.pdf

ÍNDICE

ENLACES DE CÓDIGO.

Los enlaces o vínculos necesarios de los códigos incluidos en este trabajo, y que se refieren a los programas, ya sea segmento o versión completa de dicho programa, se enlistan, a continuación, y se agrupan por Unidad.

Unidad 1. Lenguaje de programación orientada a objetos con Java.

Aprendizaje 1.1. Organización General de un programa en Java

Aprendizaje 1.2. Clase Carro con atributos

Autoevaluación.

Aprendizaje 1.2. El mayor de tres números

Aprendizaje 1.3. Clase Persona, setter y getter

Aprendizaje 1.4. Teclea tu nombre y edad

Unidad 2. Estructuras de control de secuencia en Java.

Aprendizaje 2.1. Ejemplo 1. Prueba de divisibilidad *if*.

Aprendizaje 2.1. Ejemplo 2. Prueba de divisibilidad *if-else*.

Aprendizaje 2.1. Ejemplo 3. Comparación de dos números *if - else* anidada.

Aprendizaje 2.2. Ejemplo 4. Asignación de calificación con switch.

Aprendizaje 2.3. Ejemplo 5. Tabla de multiplicar con *for*.

Aprendizaje 2.4. Ejemplo 6. Tabla de multiplicar con *while*.

Aprendizaje 2.5. Ejemplo 7. Tabla de multiplicar con *do - while*.

Aprendizaje 2.7. Ejemplo 8. Inicialización de un arreglo Unidimensional.

Aprendizaje 2.9. Ejemplo 9. Inicialización de un arreglo Bidimensional.

Aprendizaje 2.10. Proyecto Final.

Autoevaluación.

Aprendizaje 2.1. Ejercicio 1. El mayor de tres números.

Aprendizaje 2.1. Ejercicio 2. Mayor o menor de edad.

Aprendizaje 2.1. Ejercicio 3. Aprobado o reprobado.

Aprendizaje 2.2. Calculadora.

Aprendizaje 2.5. Promedio de números positivos.

Unidad 3. Polimorfismo, constructores, colaboración y herencia de clases.

Aprendizaje 3.1. Polimorfismo-Mascota

Aprendizaje 3.1. Polimorfismo-MascotaSonido

Aprendizaje 3.2. Colaboración de Clases-DueñoMascota

Aprendizaje 3.2. Colaboración de Clases-Veterinaria

Aprendizaje 3.3. Colaboración de Clases-Gestión de Biblioteca

Aprendizaje 3.4. Herencia Clases-Persona/Estudiante

Aprendizaje 3.5. Herencia Clases-Gestión de Calificaciones

Autoevaluación.

Aprendizaje 3.5. Herencia Clases-Clase Materia

Unidad 4. Interfaz gráfica de usuario.

Aprendizaje 4.2. JFrame, JLabel y JButton-Hola Mundo

Aprendizaje 4.3 Proyecto JFrame, JLabel y JButton.

Aprendizaje 4.4 JTextField, JTextArea y JComboBox-Opciones de colores

Aprendizaje 4.5. Proyecto JTextField, JTextArea y JComboBox-Tienda de Postres

Aprendizaje 4.6. JMenuBar, JMenu y JMenuItem-Menú Archivo Editar

Aprendizaje 4.7 JCheckBox y JRadioButton-App de Postres.

Aprendizaje 4.8. Clase Graphics-Dibujos

Aprendizaje 4.9. Clase Graphics-Rellenos

Aprendizaje 4.10. Tableta Digital

Aprendizaje 4.11 Máquina de Dulces

ÍNDICE

EXAMEN EXTRAORDINARIO

Con la finalidad de que tengas una perspectiva de cómo ha sido tu preparación para presentar el EXAMEN EXTRAORDINARIO de la asignatura CIBERNÉTICA Y COMPUTACIÓN II, enseguida podrás probar tus aprendizajes, conocimientos y habilidades adquiridos con la presente guía, ya que en las páginas siguientes se comparte un *Simulador de EXAMEN EXTRAORDINARIO*. Te recomendamos contestar el simulador examen y, en caso de alguna duda con tus respuestas, en la última página de este trabajo podrás verificarlas con la *HOJA DE RESPUESTAS* respectiva.

¡¡Te deseamos bastante suerte y éxito!!

Estudiante: _____ No. De cuenta: _____

INSTRUCCIONES: LEE CON CUIDADO CADA UNA DE LAS PREGUNTAS, SELECCIONA Y MARCA LA RESPUESTA QUE CONSIDERES CORRECTA, **RECUERDA PARA QUE TUS RESPUESTAS SEAN CONSIDERADAS, DEBERÁS REALIZAR Y ENTREGAR EL PROCEDIMIENTO** NECESARIO PARA LLEGAR A SU SOLUCIÓN.

Unidad 1. Lenguaje de programación orientada a objetos con Java.

1. Son identificadores predefinidos que tienen un significado específico, generalmente corresponden a nombres de instrucciones del lenguaje de programación y no se pueden utilizar como variables en un programa.
   a. Comentarios.
   b. Identificadores.
   c. Palabras reservadas.
   d. Sentencias.

2. El método _____ (también llamado método de acceso) es un método que permite obtener el valor de un atributo privado desde fuera de la clase.
   a. setter
   b. super
   c. getter
   d. main

3. Elige la forma correcta de declarar los métodos set y get.

a.
```
public int getCalificacion( ) {
        return calificacion;
    }
   public setCalificacion(int numero){
       this.calificacion= new numero;
    }
```

b.
```
public int getCalificacion( ) {
        return calificacion;
    }
   public setCalificacion(int numero){
       this.calificacion=numero;
    }
```

c.
```
public int getCalificacion(int numero ){
        return calificacion;
    }
   public setCalificacion(){
       this.calificacion=numero;
    }
```

d.
```
public int getCalificacion( ){
        this.calificacion;
    }
   public setCalificacion(int numero){
       this.calificacion=numero;
    }
```

4. Es una propiedad de los objetos que lo distingue de los demás, dentro de la programación orientada a objetos.
    a. Identidad.
    b. Comportamiento.
    c. Atributo.
    d. Comportamiento.

5. Es la clase que me permite la entrada de datos (interacción de ingreso de datos entre el usuario y el programa) en la creación de un programa.
    a. Clase principal.
    b. Clase de datos
    c. Método System.in
    d. Clase Scanner.

Unidad 2. Estructuras de control de secuencia en Java.

6. Identifica la salida correcta del siguiente segmento de programa, si el valor de ***num1 = 2*** y ***num2 = 2***

```
if (num1 % num2 == 0) {
        System.out.println( num1 +" es divisible por "+ num2);
    }

else {
       System.out.println( num1 +" no es divisible por "+ num2);
}
```

    a. El resultado es 0.
    b. 2 es divisible por 2.
    c. El resultado es 2.
    d. 2 no es divisible por 2.

7. Identificar salida correcta a partir del siguiente pseudocódigo, si el valor de **promedio = 8 y Nombre="Sandra"**

```
Si (promedio>=6){
      Escribir(n.getNombre( ) +" es un alumno(a) aprobado(a)");
      //Sentencia if simple anidada
     Si (promedio>=9){
       Escribir(" tiene derecho a una Beca");
     }
sino{
       Escribir(n.getNombre( ) +" es un alumno(a) NO Aprobado(a)");
    }
```

    a. Sandra es un alumno(a) NO Aprobado(a)
    b. Sandra tiene derecho a una Beca.

c. Sandra es un alumno(a) aprobado(a)
d. Sandra es un alumno(a) aprobado(a) tiene derecho a una Beca.

8. Ordena de forma correcta la estructura del siguiente programa de acuerdo al orden de ejecución, siendo el **main** el primero en ejecutarse:

I.

```
import java.util.Scanner;
public class Operaciones{
    private double num1 = 0.0;
    private double num2 = 0.0;
    private double result= 0.0;
    private int opcion = 0;
```

II.

```
public static void main(String[ ] args) {

Operaciones oper = new Operaciones( );
 Scanner teclado=new
Scanner(System.in);
 System.out.print("Escribe el valor del
 primer número: ");
 oper.setNum1(teclado.nextDouble( ));
 System.out.print("Escribe el valor del
 segundo número: ");
 oper.setNum2(teclado.nextDouble( ));
 oper.Opcion( );
//muestra opciones
 oper.setOpcion(teclado.nextInt( ));
 //se ingresa alguna de las 2 opciones
 int opc= oper.getOpcion( );
```

III.

```
 //Se realizan los métodos
 para cada operación

 public double Suma( ){
        result = num1 + num2;
        return result;
 }

 public double Resta( ){
        result = num1 - num2;
        return result;
 }
```

IV.

```
//Se realiza la operación
seleccionada en opc

switch (opc) {

case 1: //opción 1 es la suma
    System.out.println("\n La suma
    es "+ oper.Suma( ));
         break;
case 2: //opción 2 es la resta
    System.out.println("\n La resta
    es "+ oper.Resta( ));
         break;
default: //ninguna de las 2
          opciones
System.out.println("\n Opción no
válida");
}
}
```

V.

```
//Se declaran los métodos Getter y Setter
para cada atributo

public double getNum1( ){
          return num1;
           }
public void setNum1(double num1){
          this.num1 = num1;
           }
public double getNum2( ){
          return num2;
           }
public void setNum2(double num2){
          this.num2 = num2;
           }
public int getOpcion( ){
          return opcion;
          }
public void setOpcion(int opcion){
          this.opcion = opcion;
          }
```

VI.

```
// Se declara el método
Opciones

public void Opcion( ){
System.out.println("\n
OPERACIONES");
System.out.println(" 1. Suma");
System.out.println(" 2. Resta");
System.out.print(" Elige el
número de la operación a
realizar: ");
}
```

a. II, I, III, VI, IV, V
b. II, V, IV, VI, I, III
c. II, V, III, VI, I, IV
d. II, I, V,VI, IV, III

9. Determina la salida del siguiente código.

```
public class CicloFor {
   public static void main(String args[ ]) {
      for(int i=1; i<=10; i--) {
         System.out.println(i);
      }
   }
}
```

   a. Imprime 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
   b. El ciclo no termina, es infinito.
   c. Imprime 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
   d. Error de sintaxis.

10. De los siguientes segmentos de código, selecciona el que imprime de forma correcta la tabla de multiplicar del 5.

a.

```
int  numero = 5;
 for(i = 1; i=10; i++){
      tabla = numero * i;
      System.out.println(numero + "*"+ i +"="+ tabla);
     }
```

b.

```
int numero = 5;
 for(i = 1; i>=10; i++){
     tabla = numero * i;
     System.out.println(numero + "*"+ i +"="+ tabla);
     }
```

c.

```
int numero = 5;
for(i = 1; i<=10; i++){
     tabla = numero * i;
     System.out.println(numero + "*"+ i +"="+ tabla);
     }
```

d.

```
int  numero = 5;
for(i = 1; i<10; i++){
      tabla = numero * i;
      System.out.println(numero + "*"+ i +"="+ tabla);
 }
```

11. Identifica la sintaxis correcta para instanciar un objeto de la clase Scanner, para permitir al usuario el ingreso de datos mediante teclado.

   a. Scanner teclado Scanner(System.in);
   b. Scanner teclado=new Scanner(System.in);
   c. Scanner new=teclado Scanner(System.in);
   d. Scanner teclado=new Scanner();

12. Inspecciona el siguiente código y selecciona el inciso que determine las afirmaciones de forma correcta con respecto al programa.

```
public class Contador {
    // Atributo de la clase
    private int valor;
    // Constructor para inicializar el contador
    public Contador(int valorInicial) {
        this.valor = valorInicial;
    }
    // Método para incrementar el contador
    public void incrementar() {
        valor++;
    }
    // Método para obtener el valor actual del
       contador
    public int getValor() {
        return valor;
    }
    // Método para mostrar el valor actual del
       contador
    public void mostrarValor() {
        System.out.println("Valor actual del contador: "
        + valor);
    }
```

```
// Método principal (main) para ejecutar el programa

public static void main(String[] args) {

    // Crear un objeto de la clase Contador con valor
    // inicial de 0

    Contador contador = new Contador(0);

    // Bucle while que se ejecuta mientras el valor del
    //contador sea menor que el límite

        while (contador.getValor() < 10) {
            // Muestra el valor actual
            contador.mostrarValor();
            // Incrementa el valor del contador
            contador.incrementar();
        }
    }
}
```

a. Afirmaciones:
   i. La clase se llama Main.
   ii. El tipo de dato "valor ", solo es para números enteros.
   iii. El "Valor actual del contador:", inicia en 1.
   iv. El método get es de tipo entero.
   v. La salida del programa imprime del 1 al 10.

b. Afirmaciones:
   i. La clase se llama Contador.
   ii. El tipo de dato "valor ", solo es para números enteros.
   iii. El "Valor actual del contador:", inicia en 1.
   iv. El método set es de tipo entero.
   v. La salida del programa imprime del 1 al 10.

c. Afirmaciones:
   i. La clase se llama Contador.
   ii. El tipo de dato "valor ", solo es para números enteros.
   iii. El "Valor actual del contador:", inicia en 0.
   iv. El método get es de tipo entero.
   v. La salida del programa imprime del 0 al 9.

d. Afirmaciones:
   i. La clase principal se llama Contador.
   ii. El tipo de dato "valor ", solo es para números racionales.
   iii. El "Valor actual del contador:", inicia en 0.
   iv. El método get es de tipo racional.
   v. La salida del programa imprime del 0 al 9.

13. Es la estructura repetitiva, donde se ejecuta al menos una vez antes de que se evalúe la expresión lógica, ya que la condición se verifica al final del ciclo, más no, al inicio.
    a. Estructura repetitiva for.
    b. Estructura repetitiva while.
    c. Estructura repetitiva do - while.
    d. Estructura repetitiva switch.

14. Es la sintaxis correcta para almacenar 5 elementos de tipo entero en un arreglo llamado números.
    a. char[] numeros = {1, 2, 3, 4, 5}
    b. int[] numeros = [10, 20, 30, 40, 50}];
    c. int[] numeros = {"a", "b", "c", "d", "e"};
    d. int[] numeros = {10, 20, 30, 40, 50};

Unidad 3. Polimorfismo, constructores, colaboración y herencia de clases.

15. De los siguientes segmentos de programa identifica aquel donde se esté utilizando la colaboración entre clases entre la Clase Dueño y la clase Mascota.

a.
```
// Clase Mascota
public class Mascota {
    private String nombre;
//Método Constructor
public Mascota(String nombre) {
    this.nombre = nombre;    }
public String getNombre() {
    return nombre;   }
}
// Clase Dueño
public class Dueño {
    private String nombre;
    private String nombreMascota;
 //Método Constructor
public Dueño(String nombre, String
    nombreMascota) {
    this.nombre = nombre;
    this.nombreMascota =  nombreMascota;    }
public void presentarMascota() {
    System.out.println("Soy " + nombre + " y mi
    mascota es " + nombreMascota);
    }
}
```

b.
```
// Clase Mascota
public class Mascota {
    private String nombre;
//Método Constructor
public Mascota(String nombre) {
    this.nombre = nombre;    }
public String getNombre() {
    return nombre;   }
}
// Clase Dueño
public class Dueño {
    private String nombre;
 //Método Constructor
public Dueño(String nombre) {
    this.nombre = nombre;    }
public void presentarMascota() {
    System.out.println("Soy " + nombre + " y mi
    mascota es " + "Firulais");
    }
}
```

c.
```
// Clase Mascota
public class Mascota {
```

d.
```
// Clase Mascota
public class Mascota {
```

```
        private String nombre;
    //Método Constructor
    public Mascota(String nombre) {
        this.nombre = nombre;     }
    public String getNombre() {
        return nombre;    }
    }
    // Clase Dueño
    public class Dueño {
        private String nombre;
        private Mascota mascota;
     //Método Constructor
    public Dueño(String nombre, Mascota mascota) {
        this.nombre = nombre;
        this.mascota = mascota;
        }
    public void presentarMascota() {
        System.out.println("Soy " + nombre + " y mi
        mascota es " + mascota.getNombre());
        }
    }
```

```
        private String nombre;
    //Método Constructor
    public Mascota(String nombre) {
        this.nombre = nombre;     }
    public String getNombre() {
        return nombre;    }
    }
    // Clase Dueño
    public class Dueño {
        private String nombre;
        private int edad;
     //Método Constructor
    public Dueño(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
        }
    public void presentarMascota() {
        System.out.println("Soy " + nombre + " y mi
        edad es " + edad);
        }
    }
```

16. Asocia la definición con su correspondiente respuesta correcta.

	Definición	**Respuesta**
I.	Nombre como se le conoce a una clase hija en programación orientada a objetos.	a. protected
II.	Palabra reservada para indicar herencia.	b. Reutilización de código
III.	Menciona una de las ventajas del uso de herencia en la programación.	c. subclase
		d. extends
IV.	Nombre como se le conoce a una clase padre en programación orientada a objetos.	e. superclase

a. I:e - II:a - III:b - IV:c
b. I:c - II:e - III:b - IV:a
c. I:d - II:e - III:b - IV:c
d. I:c - II:d - III:b - IV:e

Unidad 4. Interfaz gráfica de usuario.

17. ¿Qué es una Interfaz Gráfica de Usuario (GUI)?

    a. Un conjunto de instrucciones en Java utilizadas exclusivamente para la manipulación de bases de datos.
    b. Un sistema operativo basado en Java para ejecutar aplicaciones sin necesidad de hardware adicional.
    c. Una colección de componentes visuales como botones, ventanas y cuadros de texto que permiten la interacción del usuario con la aplicación.
    d. Un tipo de archivo que almacena exclusivamente código fuente sin elementos visuales.

18. En Java, existen dos paquetes principales para la creación de interfaces gráficas: **javax.swing** y **java.awt.** Relacione correctamente las siguientes características con el paquete al que pertenecen.

Característica	Paquete
I. Proporciona componentes ligeros que no dependen directamente del sistema operativo.	a. javax.swing
II. Incluye elementos como JButton, JLabel y JFrame.	b. java.awt
III. Depende del sistema operativo para la representación visual de los componentes.	
IV. Ofrece mayor flexibilidad en la personalización de los componentes gráficos.	
V. Utiliza los componentes nativos del sistema operativo para la interfaz gráfica.	

    a. I:b - II:b - III:a - IV:b - V:a
    b. I:a - II:a - III:b - IV:a - V:b
    c. I:a - II:b - III:a - IV:b - V:a
    d. I:b - II:a - III:b - IV:a - V:b

19. Completa las palabras del párrafo con el orden de las opciones mostradas.

    Una vez creado un JFrame podemos modificar algunas propiedades como: _____ para cambiar el tamaño de la ventana, _____ para indicar el título de la ventana, si utilizamos un JLabel podremos mediante _____ indicar el texto que deseamos se muestre y al utilizar un JButton _____ para cambiar su color de fondo.

    a. preferredSize, title, getText, defaultColor
    b. resizable, title, setText, setBackground
    c. resizable, setText, title, setBackground
    d. preferredSize, title, setText, defaultColor

20. Ordena las sentencias para calcular el área de un cuadrado empleando una interfaz gráfica de usuario.

    I) area = lado * lado;
    II. double lado, area;
    III) jLabelArea.setText(Double.toString(area));
    IV) lado = Double.parseDouble(jTextlado.getText());

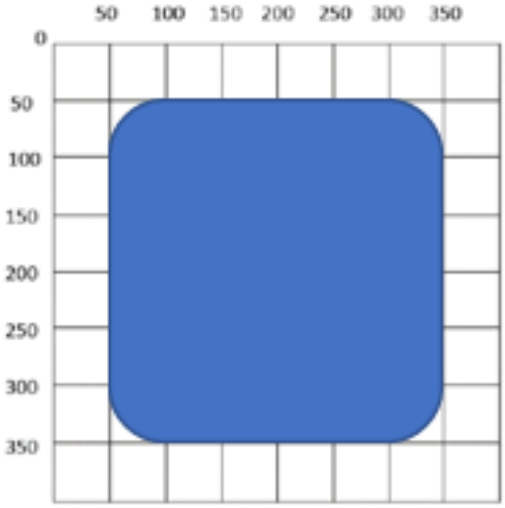
    a. III, I, IV, II
    b. II, I, III, IV
    c. III, IV, II, I
    d. II, IV, I, III

21. Ordena las sentencias para mostrar un Menú con dos opciones empleando una interfaz gráfica de usuario.

    I) JMenuItem itemOp1 = new JMenuItem("Opción 1");
    II) JMenuBar menuBar = new JMenuBar();
    III) JMenuItem itemOp2 = new JMenuItem("Opción 2");
    IV) JMenu menuOpciones = new JMenu("Opciones");

    a. IV, II, I, III
    b. II, IV, I, III
    c. I, III, II, IV
    d. IV, I, III, IV

22. Selecciona las sentencias que dibujan la siguiente figura.



a. dibujo.setColor(Color.blue);

```
dibujo.fillRoundRect(50,50,350,350,50,50)
;
```

b. dibujo.setColor(Color.blue);

```
dibujo.fillRoundRect(50,50,50,50,350,350)
;
```

c. dibujo.setColor(Color.blue);

```
dibujo.fillRoundRect(50,50,300,300,50,50)
;
```

d. dibujo.setColor(Color.blue);

```
dibujo.fillRoundRect(50,350,300,300,350,5
0);
```

ÍNDICE

Hoja de respuestas.

Pregunta	Respuesta
1	C
2	C
3	B
4	A
5	D
6	B
7	C
8	D
9	B
10	C
11	B
12	C
13	C
14	D
15	C
16	D
17	C
18	B
19	B
20	D
21	B
22	C
Aciertos	**Calificación**
0 a 12	5
13 a 14	6
15 a 16	7
17 a 18	8
19 a 21	9
22	10

Esta obra se terminó, y publicó, en la primavera de 2025.