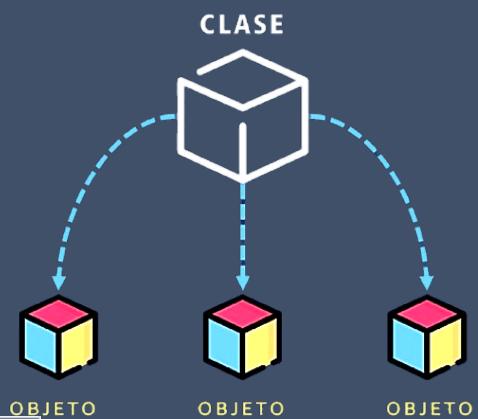


# POO

## Programación Orientada a Objetos



# Manual



**Docente Ingeniero**

Alexander Montoya

**Autores Estudiantes**

- Kevin Julián Guerrero Penagos ID 821270
- Carlos Jhoan Calderón Falla ID 931012
- Emanuel Rincón Sierra ID 938723

**Universidad Cooperativa de Colombia**



Universidad Cooperativa  
de Colombia

<b>Aprendiendo Programación Orientada a Objetos: teoría, lógica y videojuegos.....</b>	<b>6</b>
<b>1. Introducción a la POO.....</b>	<b>7</b>
1.1 Definición.....	7
1.2 Objetivo de la POO.....	7
1.3 Ejemplo aplicada en la Vida Real.....	8
1.4 Ejemplo aplicado a los videojuegos.....	9
1.5 Diferencia con la programación estructurada.....	10
<b>2. Instalación y Configuración del Entorno de Desarrollo.....</b>	<b>11</b>
2.1 ¿Qué es JDK?.....	11
2.2 ¿Cómo instalar el IDE?.....	11
2.3 ¿Ventajas de usar un IDE?.....	15
2.4 Creación de un Proyecto en Eclipse IDE.....	16
<b>3. Variables en la Programación Orientada a Objetos.....</b>	<b>22</b>
3.1 Concepto de variable.....	22
3.2 Tipos de variables en POO.....	23
Strings.....	23
Enteros (int).....	24
Flotantes (float).....	24
Booleanos (boolean).....	24
3.3 Variables en videojuegos: ejemplos prácticos.....	25
3.4 Variables en acción: ejemplo práctico en videojuegos.....	26
a. Incrementar vida.....	26
b. Modificar velocidad.....	26
c. Cambiar estado.....	26
d. Ejemplo práctico: enemigo.....	27
e. Mostrar información del enemigo.....	28
f. Simulación de daño al enemigo.....	28
g. Verificar si el enemigo sigue vivo.....	28
3.5 Buenas prácticas al usar variables.....	29
<b>4. Fundamentos de la Programación Orientada a Objetos.....</b>	<b>30</b>
4.1 Concepto de clase y objeto.....	30
4.2 Atributos y métodos.....	33
4.4 Encapsulamiento.....	35
4.5 Herencia.....	36
4.6 Polimorfismo.....	37
4.7 Abstracción.....	38
<b>5. Manejo de Strings en Java.....</b>	<b>40</b>
5.1 Definición.....	40
5.2 Características principales de los Strings.....	40
5.3 Métodos más usados en la clase String.....	41

5.4 Ejemplo básico: uso de métodos de String.....	42
5.5 Formateo de Strings.....	43
5.6 Concatenación de Strings.....	44
5.7 Ejemplo aplicado a los videojuegos.....	45
5.8 Buenas prácticas con Strings.....	46
<b>6. Sobre carga de métodos.....</b>	<b>47</b>
6.1 Concepto.....	47
6.2 Características principales.....	47
6.3 Ejemplo en la vida real.....	47
6.4 Ejemplo aplicado a los videojuegos.....	48
6.5 Código ejemplo.....	48
6.6 Salida en consola.....	50
6.7 Explicación del código.....	50
6.8 Ejemplo adicional: cálculo de daño sobre cargado.....	50
6.9 Buenas prácticas con la sobre carga.....	52
<b>7. Constructores en la Programación Orientada a Objetos.....</b>	<b>53</b>
7.1 Definición y propósito.....	53
7.2 Tipos de constructores en Java.....	53
7.2.1. Constructor por defecto.....	53
7.2.2. Constructor con parámetros.....	55
7.3 Ejemplo aplicado a los videojuegos.....	56
7.4 Salida de consola.....	58
7.5 Explicación del ejemplo.....	59
7.6 Buenas prácticas al usar constructores.....	59
7.7 Ejemplo adicional: enemigos creados con constructores.....	60
<b>8. Relaciones entre clases.....</b>	<b>62</b>
8.1 Introducción.....	62
8.2 Asociación.....	63
8.3 Agregación.....	65
8.4 Composición.....	69
8.5 Diferencias entre asociación, agregación y composición.....	72
8.6 En Conclusión.....	72
<b>9. Interfaces.....</b>	<b>73</b>
9.1 Concepto.....	73
9.2 Ejemplo aplicado a videojuegos.....	73
<b>10. Clases abstractas.....</b>	<b>76</b>
10.1 Concepto.....	76
10.2 Ejemplo aplicado a videojuegos.....	76
<b>11. ARREGLOS.....</b>	<b>79</b>
11.1 ¿Qué son los arreglos en Java?.....	79
11.2. ¿Para qué se utilizan los arreglos en un programa?.....	80

11.3. ¿Cómo se declara un arreglo en Java?.....	80
11.4. ¿Cómo se almacenan y acceden los datos dentro de un arreglo?.....	81
11.5. ¿Cuál es la diferencia entre un arreglo y una variable simple?.....	81
11.6. ¿Qué tipo de datos se pueden guardar dentro de un arreglo?.....	82
11.7. ¿Cómo se puede recorrer un arreglo utilizando un ciclo for o foreach?.....	82
11.9. Ejemplo de un programa en Java que use un arreglo.....	83
11.9.1 Para guardar 5 notas y mostrar su promedio.....	83
11.10. Explique de los arreglos razón que son útiles en la programación.....	84
<b>12. Arreglo Ejercicios Orientado A Videojuegos.....</b>	<b>85</b>
12.1. Ejercicio 1 EL DADO:.....	85
12.1.1 Descripción del código.....	85
12.1.2 Código.....	85
12.1.3 Paso a paso del código.....	86
12.2 Ejercicio 2 PIEDRA, PAPEL O TIJERAS:.....	88
12.2.1 Descripción del código.....	88
12.2.2 Código.....	89
12.2.3 Paso a paso del código.....	89
12.3. Ejercicio 3 EL BattleRoyale:.....	92
12.3.1 Descripción del código.....	92
12.3.2 Código.....	92
12.3.3 Paso a paso del código.....	93
12.4. Ejercicio 4 Combate Goblins Arreglo:.....	94
12.4.1 Descripción del código.....	94
12.4.2 Código.....	95
12.4.3 Paso a paso del código.....	96
12.5. Ejercicio La Vuelta Al Mundo (Carrera de Carros):.....	99
12.5.1 Descripción del código.....	99
12.5.2 Código.....	99
12.5.3 Paso a paso del código.....	100
<b>13. Matriz.....</b>	<b>102</b>
13.1 ¿Qué es una matriz?.....	102
13.2 ¿Para qué sirve una matriz en Java?.....	102
13.3 Para evaluar su funcionamiento haremos el siguiente programa básico en java.....	102
13.4 Salida en consola:.....	103
13.5 Miremos como funciona en relación a un videojuego:.....	104
13.6 Salida en consola:.....	105
<b>14. Introducción al modo grafico en Java.....</b>	<b>111</b>
14.1. ¿Qué es una interfaz gráfica de usuario?.....	111
14.2. Programación dirigida por eventos.....	111
14.3. Librerías gráficas en Java: AWT y Swing.....	112
14.3.1 AWT.....	112

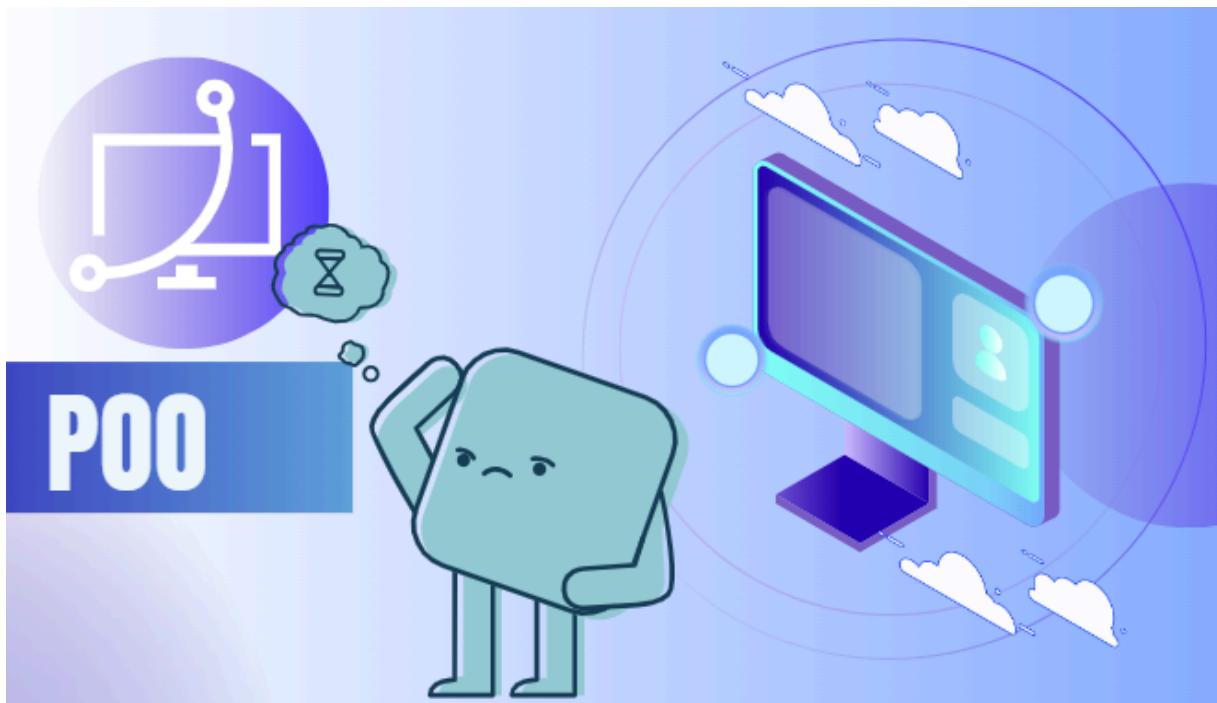
14.3.2 Swing (la principal que usaremos).....	112
14.4. ¿Qué es AWT?.....	112
14.5. Componentes principales de AWT.....	113
14.5. Swing.....	113
14.5.1. ¿Qué es Swing?.....	113
14.5.2. Ventajas de Swing frente a AWT.....	113
14.5.3 Componentes básicos de Swing.....	113
14.6. import javax.swing.*;.....	114
14.7. java.awt.*;.....	114
14.8. Dimensiones y ubicación de una ventana en Java Swing.....	117
4.8.1. Definición del tamaño de una ventana.....	117
14.9. Ubicación predeterminada de la ventana.....	117
14.10. Definición manual de la posición de la ventana.....	118
14.10.1 setLocation(x, y);.....	118
14.11 Centrando una ventana en la pantalla.....	118
14.12. Definir simultáneamente la posición y el tamaño de una ventana.....	119
14.13. Uso de setDefaultCloseOperation.....	120
14.13.1 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);.....	120
14.14. Eventos.....	120
14.14.1. ¿Qué es un evento?.....	120
14.14.2. ¿Qué es un listener o manejador de eventos?.....	120
14.15. Implementar ActionListener a la clase.....	124
14.15.1 Listener anónimo.....	124
14.16. Colores en Interfaces Gráficas con Java Swing.....	125
14.16.1 Colorear el fondo de un componente.....	125
14.16.2. Colorear el texto de un componente.....	126
14.16.3. Colores predefinidos en Java.....	126
14.16.4 Creación de colores personalizados (RGB).....	127
14.17 Cómo cambiar el tipo de letra, el estilo y el tamaño.....	129
14.18. Ejemplo gráfico.....	130
<b>15. Práctica aplicada: ejercicios 22 – 52.....</b>	<b>135</b>
15.1 Ejercicios del libro Lógica de Programación de Efraín Oviedo:.....	135
<b>16. Referencias.....</b>	<b>162</b>

## Aprendiendo Programación Orientada a Objetos: teoría, lógica y videojuegos

Este manual está orientado al aprendizaje y enseñanza de la Programación Orientada a Objetos (**POO**) desde una perspectiva técnica y práctica.

Su propósito es explicar de manera clara los conceptos fundamentales de la POO, como **clases, objetos, atributos, métodos, herencia, polimorfismo y encapsulamiento**, apoyándose en ejemplos matemáticos, lógicos y, especialmente, en contextos de **videojuegos**, donde este paradigma es clave para la creación de personajes, escenarios y mecánicas.

De esta forma queremos que pueda comprender la teoría y al mismo tiempo, visualizar su aplicación en situaciones reales, logrando un aprendizaje más dinámico y contextualizado.

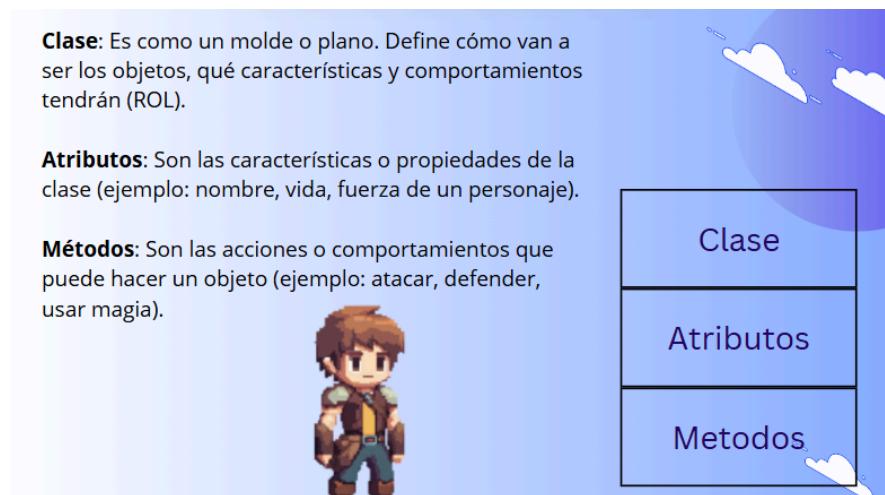


# 1. Introducción a la POO

## 1.1 Definición

La Programación Orientada a Objetos (POO) es un paradigma de desarrollo de software que organiza el código en estructuras llamadas **objetos**, los cuales son la unidad principal de construcción dentro de un programa, cada objeto combina en sí mismo tanto los **atributos** (que representan características o información del objeto) como los **métodos** (que definen las acciones o comportamientos que dicho objeto puede realizar).

De esta manera, la **POO** permite modelar el software de una forma muy similar a como los seres humanos perciben la realidad, ya que cada objeto refleja elementos tangibles o conceptuales, tales como una persona, un vehículo, un personaje de videojuego o incluso un sistema completo.



## 1.2 Objetivo de la POO.

El principal objetivo de la POO es proporcionar una forma estructurada y modular de programar, permitiendo la reutilización del código, la reducción de la complejidad y la facilidad de mantenimiento de los programas, nos facilita la creación de sistemas escalables y más cercanos a la realidad.

### 1.3 Ejemplo aplicada en la Vida Real.

Un **carro** puede representarse como una **clase**, la cual es como un molde o plantilla que describe las características y comportamientos comunes de todos los carros.

#### a. Atributos :

Son las características que describen al carro. Por ejemplo:

- **color** → puede ser rojo, azul, negro, etc.
- **marca** → Toyota, Ford, Chevrolet, etc.
- **modelo** → 2020, 2023, etc.

Estos atributos permiten diferenciar un carro de otro.

#### b. Métodos:

Son las funciones que puede realizar un carro (**NOTA** Los métodos siempre se encuentran cuando está escrito con “()”). Por ejemplo:

- **arrancar()** → enciende el motor.
- **frenar()** → detiene el carro.
- **acelerar()** → aumenta la velocidad.

Estos métodos son iguales para todos los carros, pero la forma en que se ejecutan puede variar según los atributos de cada carro.

#### c. Objetos (nace de la clase):

Cuando se crea un carro específico a partir de la clase, se obtiene un **objeto**. Por ejemplo:

- Carro1: color = rojo, marca = Toyota, modelo = 2020

- Carro2: color = azul, marca = Ford, modelo = 2023

## 1.4 Ejemplo aplicado a los videojuegos.

En el mundo de los **videojuegos**, en **(POO)** es esencial para organizar y estructurar el código, este paradigma permite representar de forma clara a los elementos del juego, facilitando su mantenimiento y evolución.

Un ejemplo sencillo sería un **juego de aventuras** donde existe una **clase Personaje**. Esta clase funciona como plantilla para crear distintos tipos de personajes y está definida por:

- **Atributos (características)**
  - **vida** → cantidad de energía o resistencia del personaje.
  - **fuerza** → determina el poder de ataque.
  - **nivel** → indica el progreso o experiencia alcanzada.
- **Métodos (acciones):**
  - **atacar()** → permite golpear o dañar a un enemigo.
  - **saltar()** → posibilita esquivar obstáculos o moverse en diferentes plataformas.
  - **recuperarVida()** → aumenta la energía del personaje al usar poción o descansando.

A partir de esta clase, se pueden crear **objetos (personajes específicos)**, como:

- **Guerrero:** vida alta, mucha fuerza, nivel inicial 1.
- **Magos:** vida moderada, fuerza mágica elevada, nivel inicial 1.
- **Arquero:** vida media, fuerza equilibrada, nivel inicial 1.

Cada uno tiene diferentes valores en sus atributos, pero todos comparten los mismos métodos para interactuar dentro del videojuego.

## 1.5 Diferencia con la programación estructurada

Mientras que la **programación estructurada** se enfoca principalmente en funciones y procedimientos, la Programación Orientada a Objetos (**POO**) organiza el software en objetos que integran tanto los datos (atributos) como las operaciones (métodos) que actúan sobre ellos.

Este enfoque permite que cada objeto sea una representación más fiel a los elementos del mundo real, facilitando la comprensión del código y la reutilización de componentes, mientras la **programación estructurada** divide el problema en bloques lógicos secuenciales, la **POO** promueve una visión más modular, escalable e intuitiva, alineada con la manera en que las personas perciben y representan la realidad.

## 2. Instalación y Configuración del Entorno de Desarrollo



### 2.1 ¿Qué es JDK?

El *Java Development Kit* (JDK) se define como un paquete de software diseñado para desarrolladores, el cual integra las herramientas esenciales para la creación y compilación de aplicaciones en Java, dentro de sus componentes principales se encuentran el compilador, el depurador, la *Java Virtual Machine* (JVM) para la ejecución del código, así como un conjunto de bibliotecas estándar que facilitan el desarrollo de programas robustos y eficientes.



Perales, I. (2021, 15 julio). *¿Qué es el JDK y el JRE en Java?* Israel Perales.

<https://www.israel-perales.com/que-es-el-jdk-y-el-jre-java/>

### 2.2 ¿Cómo instalar el IDE?

Para esto podemos utilizar tres páginas Eclipse, IntelliJ IDEA o NetBeans, pero para este curso utilizásemos eclipse.

- **¿Qué es un IDE?**

Un Entorno de Desarrollo Integrado, es una aplicación que reúne en un solo lugar todas las herramientas necesarias para escribir, probar y depurar código de manera eficiente, las herramientas que contiene un IDE pueden ser, editor de código, depurador, compilador o intérprete, automatización de tareas y una interfaz gráfica.

**Paso 1.** Ingresamos a la página de eclipse a través del siguiente enlace:

<https://www.eclipse.org/downloads/>



**Paso 2.** Al ingresar al sitio, se visualizará la página principal de descargas.

A screenshot of the Eclipse Foundation Downloads page. At the top, there is a navigation bar with links to Proyectos, Partidarios, Colaboraciones, Recursos, and La Fundación. Below the navigation bar, there is a large graphic of a computer monitor showing a terminal window with a download icon. To the right of the graphic, the text "Descargue la tecnología Eclipse adecuada para usted" is displayed. Below this section, there is a yellow banner with the text "Descubra la IA en la Fundación Eclipse" and a link to "Más información". On the right side of the banner, there are two buttons: "Más información" and "Suscribirse". At the bottom of the page, there are two sections: one for "Instale sus paquetes IDE de escritorio favoritos" featuring the Eclipse logo, and another for "El proyecto Eclipse Temurin® proporciona tiempos de ejecución más rápidos, una alta calidad y un mejor servicio TCO a los desarrolladores".

**Paso 3.** A continuación, se debe desplazar hacia abajo hasta ubicar el botón de descarga.

A screenshot of the Eclipse Temurin download page. At the top, there is a large Eclipse logo. Below the logo, the text "Instale sus paquetes IDE de escritorio favoritos" is displayed. There are two buttons: "Más información" and a prominent orange "Descargar" button. At the bottom of the page, there is a link "Descargar paquetes | ¿Necesitas ayuda?".

**Paso 4.** Una vez localizado, se hace clic en dicho botón para iniciar el proceso.

**Paso 5.** Despu s de presionar el bot n, aparecer  un mensaje con opciones de descarga.

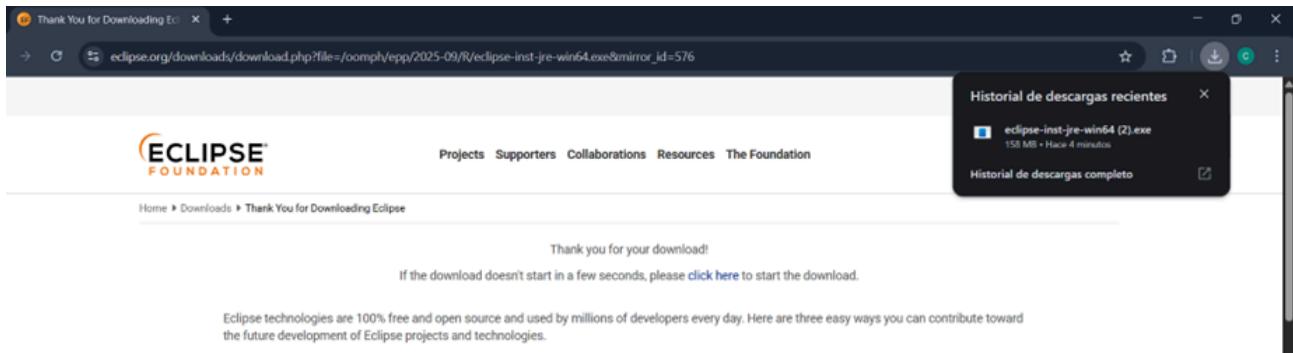


**Paso 6.** En este punto, se selecciona la opci n **Descargar x86\_64**, lo que redirige automáticamente a otra p gina.

**Paso 7.** En la nueva p gina, se mostrar  nuevamente el bot n de descarga. Al hacer clic en este, comenzar  la descarga del archivo de instalaci n.

A screenshot of the Eclipse Foundation download page for the x86\_64 version. The page features the Eclipse Foundation logo at the top left. Navigation links include "Proyectos", "Partidarios", "Colaboraciones", "Recursos", and "La Fundaci n". A breadcrumb trail shows "Hogar &gt; Descargas &gt; Descargas de Eclipse - Seleccionar un espejo". A note states: "Todas las descargas se proporcionan seg n los t rminos y condiciones del Acuerdo de usuario del software de Eclipse Foundation, a menos que se especifique lo contrario." A large orange "Descargar" button is centered. Below it, the text "Descargar desde: Brasil - C3SL - Universidad Federal de Paran  (https)" and "Archivo: eclipse-inst-jre-win64.exe SHA-512" are displayed. A link "&gt;&gt; Seleccionar otro espejo" is also present.

**Paso 8.** Una vez descargado, se debe acceder al archivo desde el navegador (normalmente en la parte superior derecha) o desde la carpeta de descargas del sistema.



**Paso 9.** Se da doble clic sobre el archivo descargado para iniciar el proceso de instalación.

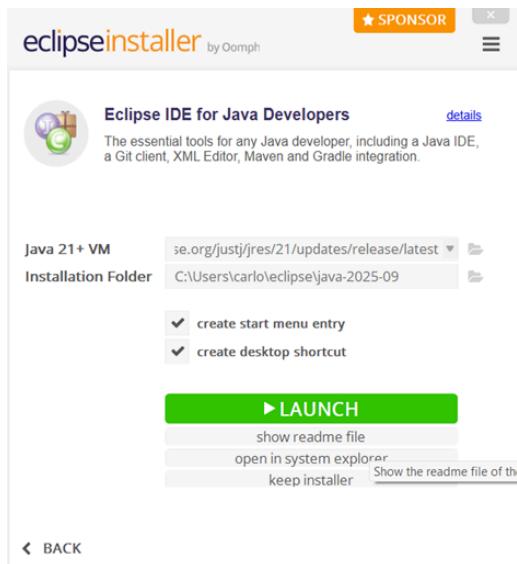
**Paso 10.** Aparecerá una ventana con diferentes opciones de instalación.



**Paso 11.** Se debe dar doble clic sobre la opción **Eclipse IDE for Java Developers**.

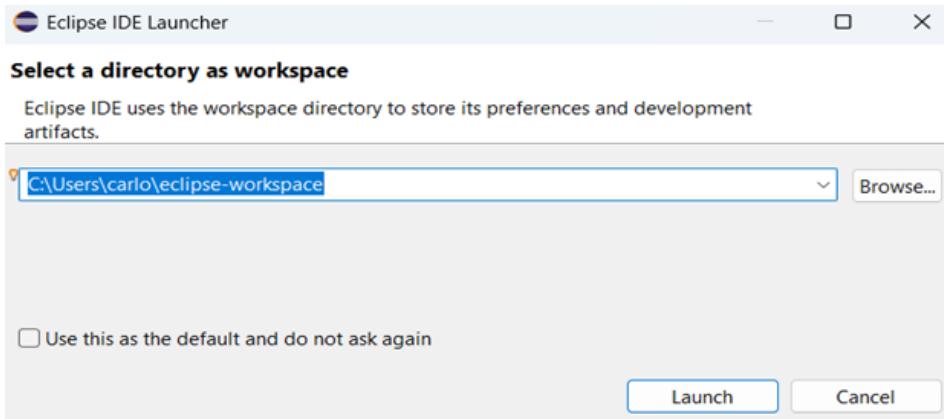
**Paso 12.** A continuación, se desplegará una nueva ventana en la que se hace clic en **Instalar** para iniciar la instalación del IDE.

**Paso 13.** Una vez finalizada la instalación, se mostrará la opción de ejecución.



**Paso 14.** Se selecciona el botón **Launch**.

**Paso 15.** Finalmente, al repetir la acción de **Launch**, se abrirá automáticamente el entorno de desarrollo Java (SUGERENCIA puedes vincularlos con alguna carpeta para organizar los proyectos).



## 2.3 ¿Ventajas de usar un IDE?

**Mejora la productividad:** Ya que, al venir integrado con un autocompletado de código, el cual te sugiere funciones, variables y estructuras mientras escribes.

**Menos errores y más claridad:** Ya que, al contar con un detector de errores en tiempo real, te avisará mientras escribes si hay fallos o inconsistencias.

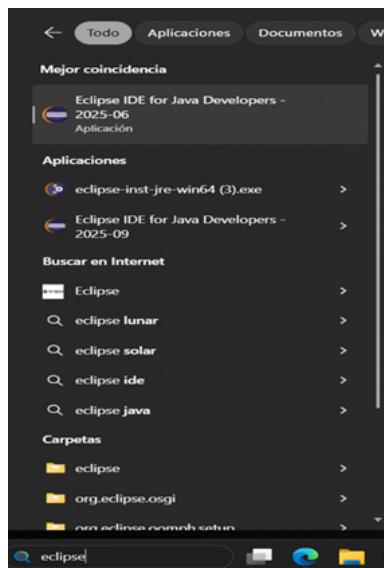
**Todo en un solo lugar:** Al incluir un editor, compilador, depurador y consola, no será necesario abrir varios programas, todo está incluido.

**Organización y estabilidad:** Al ser compatible con plugins puedes perdonar tu entorno según tus necesidades.

## 2.4 Creación de un Proyecto en Eclipse IDE

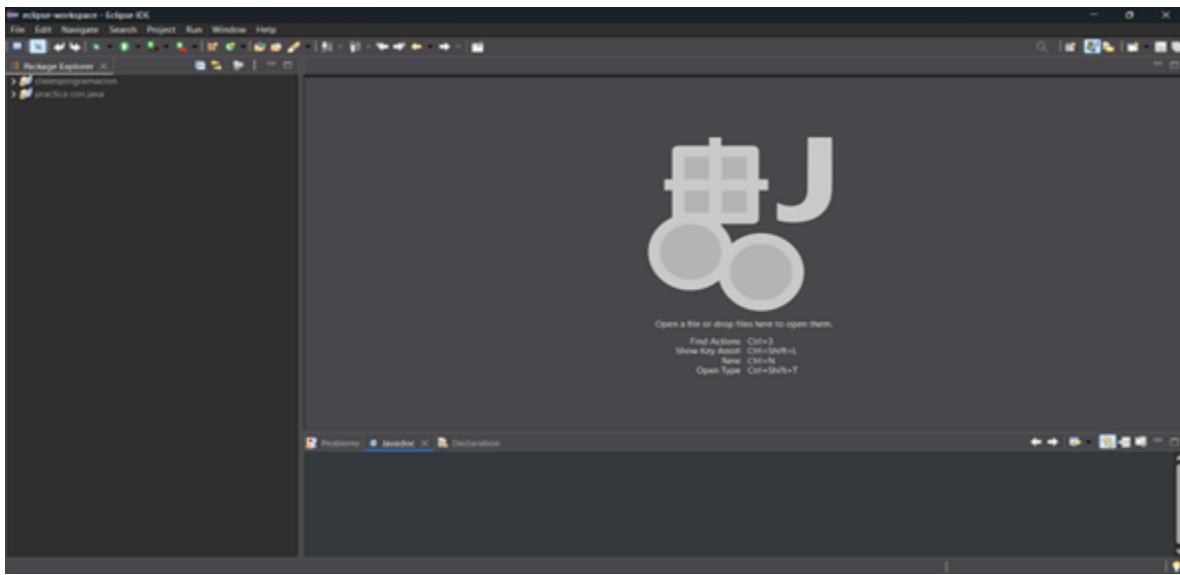
Antes de empezar a programar, el siguiente paso será crear nuestra primera carpeta (proyecto) en Java.

**Paso 1.** Abrir Eclipse: presionar **Windows + S** y escribir **Eclipse** en la barra de búsqueda.  
Dar doble clic sobre **Eclipse IDE** y automáticamente se abrirá Java.

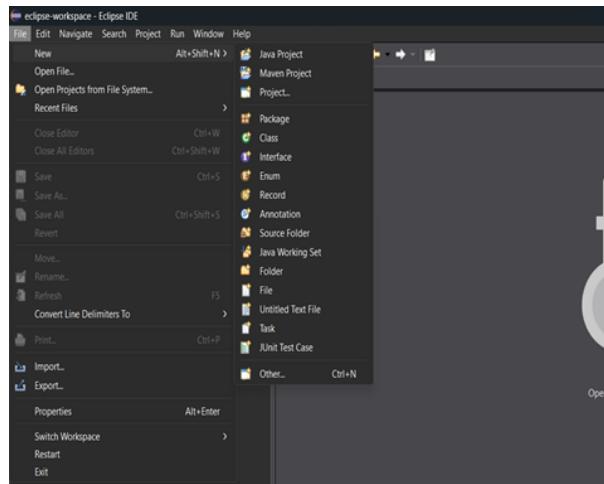


Daremos doble clic sobre Eclipse IDE y automáticamente se nos abrirá java.

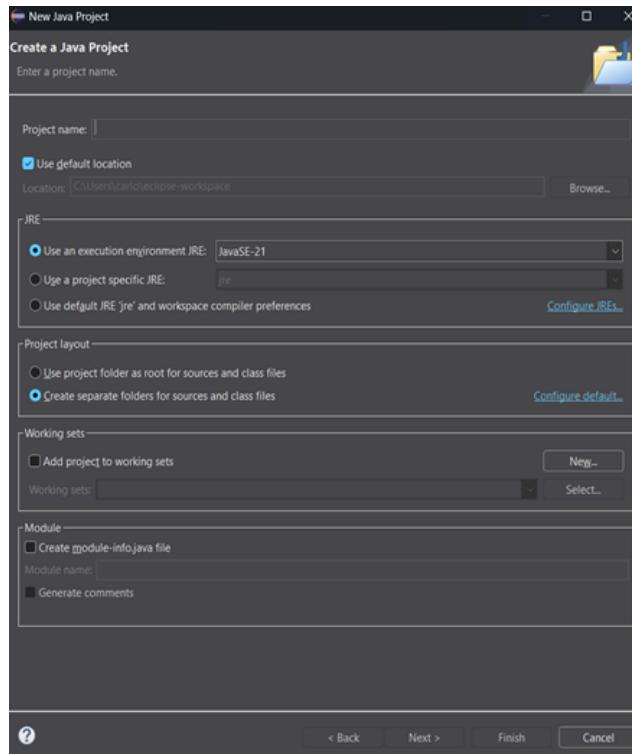
**Paso 2.** Una vez abierto, se visualizará la interfaz principal del IDE.



**Paso 3.** En la parte superior izquierda, dar clic en **File** → posicionar el cursor sobre **New** para ver más opciones.

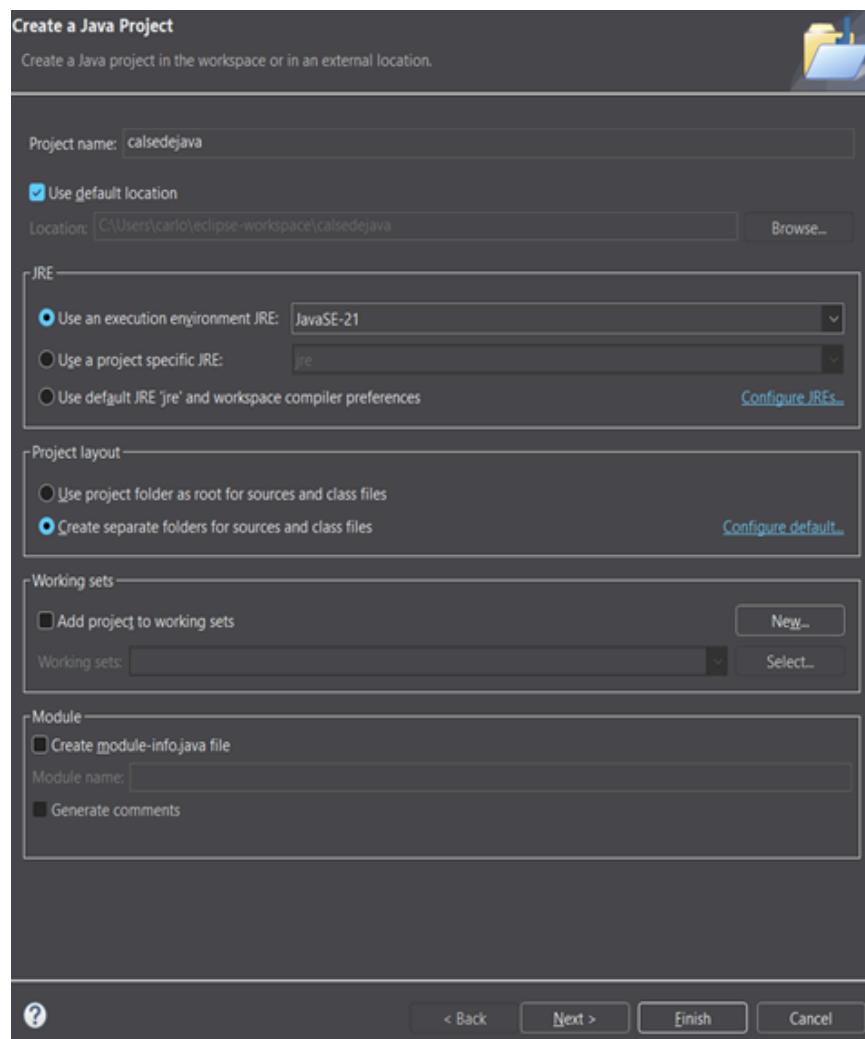


**Paso 4.** Seleccionar **Java Project**. Se abrirá una ventana emergente.

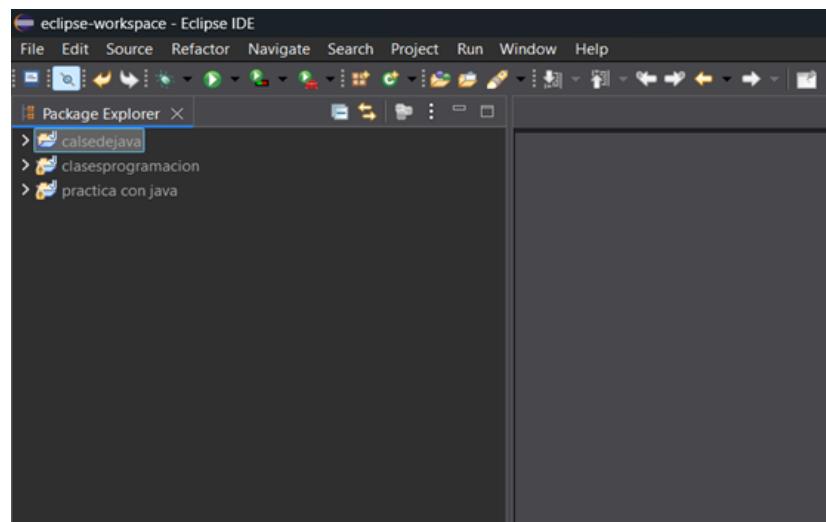


**Paso 5.** En la casilla **Project name**, asignar un nombre al proyecto (ejemplo: **clasedejava**), es importante que cuando asigne un nombre a una carpeta no dejes espacios. Después de asignado el nombre abajo hay un botón llamado **finish**, le daremos clic y automáticamente se nos creará la carpeta.

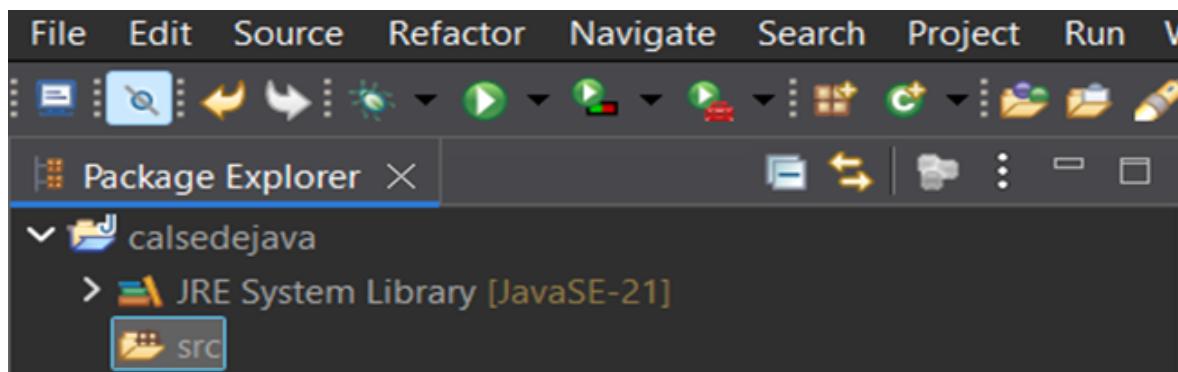
⚠ Importante: no se deben dejar espacios en los nombres.



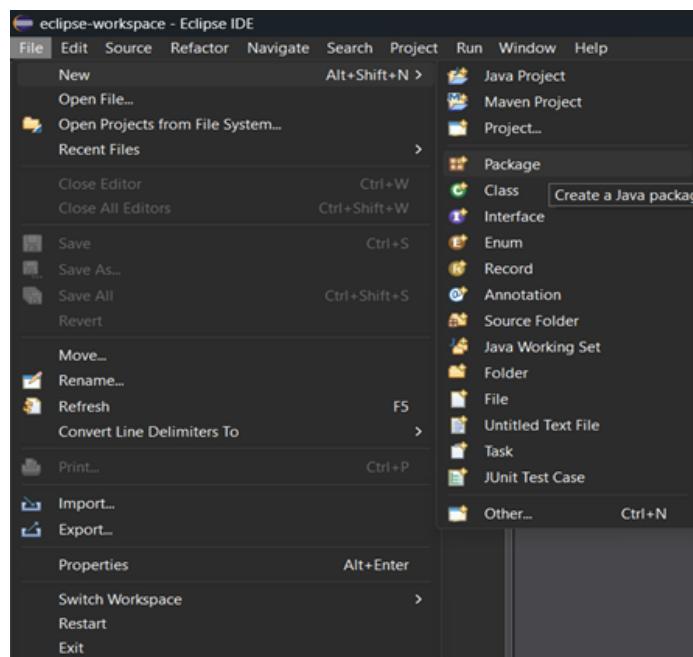
**Paso 6.** El proyecto creado aparecerá en la barra lateral izquierda.



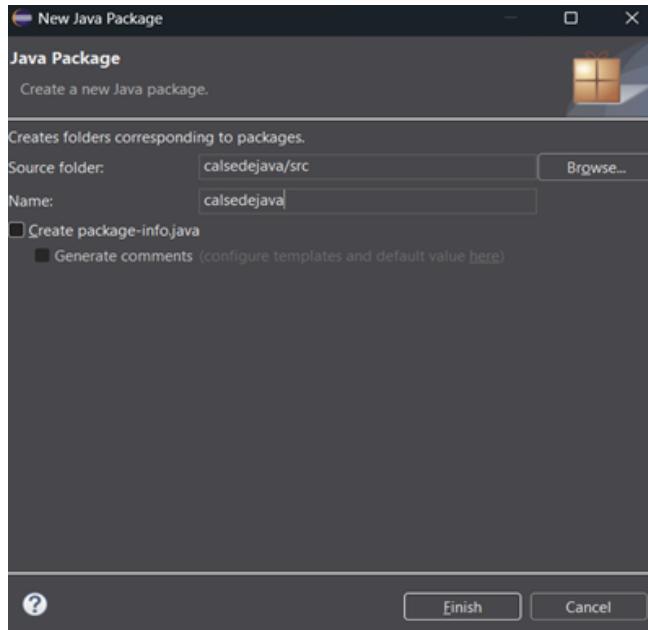
**Paso 7.** Dar doble clic sobre el proyecto creado. Allí se visualizarán dos carpetas. Seleccionar la carpeta **src**.



**Paso 8.** Volver a dar clic en **File → New → Package**.



**Paso 9.** En la ventana que aparece, dar clic en **Finish**.



Después de hacer lo anterior haremos un clásico hola mundo.

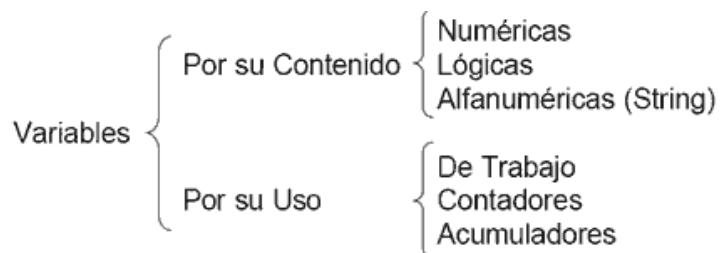
```
1 package paquete1;
2
3 public class Hello {
4
5     public static void main(String[] args) {
6         System.out.println("Hello Mundo de Java");
7     }
8
9 }
```

### 3. Variables en la Programación Orientada a Objetos

#### 3.1 Concepto de variable

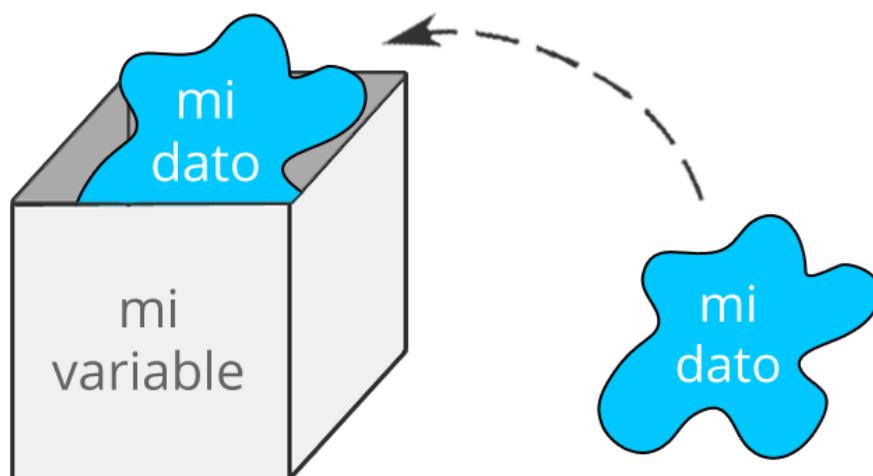
Una **variable** es un espacio de memoria identificado por un nombre único que se utiliza para **almacenar información temporal** dentro de un programa. Esta información puede ser de distintos tipos, como números, texto o valores lógicos, dependiendo del propósito.

Las variables son fundamentales en la programación, ya que permiten al programa **recordar y manipular datos** mientras se ejecuta. Por ejemplo, pueden almacenar el puntaje de un jugador, la vida de un personaje o la velocidad de un objeto en un videojuego.



Fuente: Desarrolloweb.com. Recuperado de  
<https://desarrolloweb.com/articulos/expresiones-instrucion-programacion.html>

Las variables facilitan la aplicación de **operaciones matemáticas, comparaciones lógicas y decisiones condicionales**, lo que convierte al programa en algo dinámico y adaptable según la interacción del usuario o el desarrollo de la lógica del sistema.



Fuente: TD Robótica. (s.f.). Categoría de Variables. Recuperado de  
<https://aprender.tdrobotica.co/courses/mblock-n1-introduccion/lecci%C3%B3n/categor%C3%ADa-de-variables/>

### 3.2 Tipos de variables en POO

En Java se utilizan principalmente los siguientes tipos de datos básicos:

#### Strings

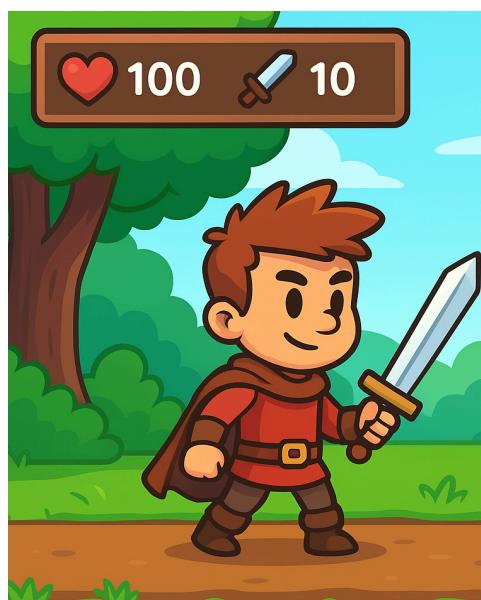
- Representan **cadenas de texto** o caracteres alfanuméricos.
- Se escriben entre comillas dobles "" y se declaran usando la palabra clave **String**.
- **Ejemplo:**

```
package EjemVideoJuego;

public class Variables {

    public static void main(String[] args) {

        // -----
        // Tipos de variables básicos
        // -----
        |
        // String: cadena de texto
        String nombrePersonaje = "Thor";
        System.out.println("Nombre del personaje: " + nombrePersonaje);
```



## Enteros (int)

- Almacenan **números enteros**, positivos o negativos, sin decimales.
- Se declaran con la palabra clave **int**.
- **Ejemplo:**

```
// int: número entero
int vida = 100;
System.out.println("Vida: " + vida);
```

## Flotantes (float)

- Almacenan **números decimales**.
- En Java, se debe agregar la letra **f** al final del número para indicar que es flotante.
- Se declaran con la palabra clave **float**.
- **Ejemplo:**

```
// float: número decimal
float velocidad = 5.5f;
System.out.println("Velocidad: " + velocidad);
```

## Booleanos (boolean)

- Representan **valores lógicos**: **true** (verdadero) o **false** (falso).
- Son útiles para condiciones y decisiones dentro del código.
- Se declaran con la palabra clave **boolean**.

- Ejemplo:

```
// boolean: valor lógico
boolean estaVivo = true;
System.out.println("¿Está vivo?: " + estaVivo);

System.out.println(); // Salto de línea para separar ejemplos
```

**Nota:** Conocer el tipo de dato de cada variable es importante, ya que determina cómo se puede manipular (operaciones matemáticas, lógicas, comparaciones, etc.).

### 3.3 Variables en videojuegos: ejemplos prácticos

En los videojuegos, las variables permiten representar propiedades y estados de los personajes, objetos o escenarios.

Ejemplos comunes:

```
// Nombre del personaje
String nombrePersonaje = "Thor";
```

```
// Vida del personaje
int vida = 100;
```

```
// Velocidad de movimiento
float velocidad = 5.5f;
```

```
// Estado de acción
boolean estaVivo = true;
```

```
// Imprimir información en consola
```

```
System.out.println("Personaje: " + nombrePersonaje);
System.out.println("Vida: " + vida);
System.out.println("Velocidad: " + velocidad);
System.out.println("¿Está vivo? " + estaVivo);
```

### 3.4 Variables en acción: ejemplo práctico en videojuegos

```
// -----
// Operaciones con variables
// -----
```

#### a. Incrementar vida

```
// Incrementar vida
vida += 20; // Suma 20 a la vida actual
System.out.println("Vida después de recibir vida extra: " + vida);
```

El fragmento muestra cómo modificar el valor de una variable utilizando el operador de asignación compuesto `+=`. En el contexto de un videojuego, se simula que el personaje recibe un bono de vida, actualizando su estado de manera dinámica.

#### b. Modificar velocidad

```
// Modificar velocidad
velocidad = velocidad + 2.0f; // Aumenta la velocidad
System.out.println("Velocidad aumentada: " + velocidad);
```

Aquí se ejemplifica cómo se pueden realizar operaciones matemáticas sobre variables tipo `float`. Se incrementa la velocidad de un personaje, demostrando cómo los atributos pueden cambiar durante la ejecución del juego.

#### c. Cambiar estado

```
// Cambiar estado
estaVivo = false; // El personaje muere
System.out.println("Estado actualizado: ¿Está vivo? " + (estaVivo ? "Sí" : "No"));
```

Se ilustra el uso de variables booleanas para representar estados lógicos. En este caso, el personaje muere, mostrando cómo se puede controlar la lógica de juego mediante valores `true` o `false`.

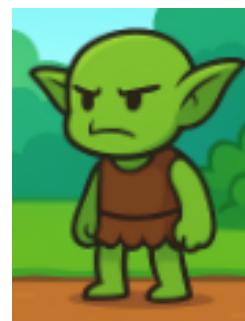
Se puede realizar un salto de línea de esta manera

```
System.out.println(); // Salto de linea
```

#### d. Ejemplo práctico: enemigo

```
// -----
// Ejemplo practico tipo videojuego
// -----
```

```
// Crear variables para un enemigo
String nombreEnemigo = "Goblin";
int vidaEnemigo = 50;
float velocidadEnemigo = 3.2f;
boolean enemigoVivo = true;
```



El bloque define un enemigo usando variables de distintos tipos, representando atributos típicos de un personaje en un videojuego: nombre, vida, velocidad y estado de vida.

#### e. Mostrar información del enemigo

```
// Mostrar informacion en consola
System.out.println("Enemigo: " + nombreEnemigo);
System.out.println("Vida del enemigo: " + vidaEnemigo);
System.out.println("Velocidad del enemigo: " + velocidadEnemigo);
System.out.println("Enemigo vivo?: " + (enemigoVivo ? "Si" : "No"));

System.out.println(); // Salto de linea
```

Se demuestra cómo imprimir variables en consola y concatenar diferentes tipos de datos para presentar información completa de un objeto dentro del juego.

#### f. Simulación de daño al enemigo

```
// Simulacion de daño al enemigo
int danio = 20;
vidaEnemigo -= danio; // Restar daño
System.out.println("Vida del enemigo despues del ataque: " + vidaEnemigo);
```

Se muestra cómo modificar la vida de un enemigo tras recibir daño, utilizando el operador `-=`. Esto refleja interacciones típicas de combate en videojuegos, donde los atributos cambian según las acciones.

#### g. Verificar si el enemigo sigue vivo

```
// Verificar si el enemigo sigue vivo
if (vidaEnemigo <= 0) {
    enemigoVivo = false;
}
System.out.println("Enemigo vivo despues del ataque?: " + (enemigoVivo ? "Si" : "No"));
```

El bloque combina variables booleanas y estructuras condicionales para controlar la lógica del juego. Se verifica si el enemigo ha perdido toda su vida y actualiza su estado en consecuencia.

### 3.5 Buenas prácticas al usar variables

- **Nombres descriptivos:** Usar nombres claros y representativos del dato que almacenan, por ejemplo: `vidaJugador` en lugar de `v`. **Evitar espacios:** Los nombres de las variables no deben contener espacios.
- **Inicializar variables:** Siempre asignar un valor inicial para evitar errores en tiempo de ejecución.
- **Tipos adecuados:** Elegir el tipo de dato correcto según la información que se va a almacenar.
- **Constantes cuando sea necesario:** Si un valor no debe cambiar, usar `final` para declararlo como constante.



Salida de Consola:

```
Nombre del personaje: Thor
Vida: 100
Velocidad: 5.5
Esta vivo?: Si

Vida despues de recibir vida extra: 120
Velocidad aumentada: 7.5
Estado actualizado: Esta vivo? No

Enemigo: Goblin
Vida del enemigo: 50
Velocidad del enemigo: 3.2
Enemigo vivo?: Si

Vida del enemigo despues del ataque: 30
Enemigo vivo despues del ataque?: Si
```

## 4. Fundamentos de la Programación Orientada a Objetos

La Programación Orientada a Objetos (POO) se fundamenta en la creación de programas estructurados en torno a objetos, estos **objetos** no solo contienen datos, sino también las operaciones que pueden realizar, se logra un enfoque más natural e intuitivo para representar situaciones reales y virtuales, como las que se ven en el desarrollo de videojuegos.

### 4.1 Concepto de clase y objeto

En la Programación Orientada a Objetos, una **clase** es la **plantilla o modelo** que define cómo serán los objetos que se crean a partir de ella. Dentro de una clase se especifican:

Por otro lado, un **objeto** es una **instancia concreta** en la que **nace** de una clase, mientras la clase es como un “**molde**” o “**receta**”, el **objeto** es el “**producto final**” construido a partir de dicho molde.

#### → Ejemplo en la vida real

- **Clase: Carro** (es la plantilla).
  - **Atributos:** color, marca, modelo, año.
  - **Métodos:** arrancar(), acelerar(), frenar().
- **Objeto:** Un carro específico, por ejemplo:
  - Toyota Corolla, color rojo, modelo 2022.
  - Este carro en particular es **una instancia de la clase Carro**.

Con esto se entiende que la clase define la estructura general de lo que es un carro, pero cada carro real tendrá valores concretos (azul, rojo, Toyota, Renault, etc.), y todos compartirán los mismos comportamientos básicos.

## CÓDIGO:

```
package EjemCarro;

public class Main {

    // Clase Carro: representa el molde o plantilla
    static class Carro {
        String marca;
        String modelo;

        // Métodos: acciones del carro
        void arrancar() {
            System.out.println(marca + " " + modelo + " esta arrancando...");
        }

        void frenar() {
            System.out.println(marca + " " + modelo + " esta frenando...");
        }
    }

    public static void main(String[] args) {
        // Crear un objeto (instancia de Carro)
        Carro miCarro = new Carro();

        // Asignar valores a los atributos
        miCarro.marca = "Toyota";
        miCarro.modelo = "Corolla";
    }
}
```

```

// Usar los métodos

miCarro.arrancar();

miCarro.frenar();

}

}

```

**Creamos la Clase Carro con los atributos y metodos().**

```

package EjemCarro;

public class Main {

    // Clase Carro: representa el molde o plantilla
    static class Carro {
        String marca;
        String modelo;

        // Métodos: acciones del carro
        void arrancar() {
            System.out.println(marca + " " + modelo + " está arrancando...");
        }

        void frenar() {
            System.out.println(marca + " " + modelo + " está frenando...");
        }
    }
}

```

Nace el objeto de la Clase Carro

```

public static void main(String[] args) {
    // Crear un objeto (instancia de Carro)
    Carro miCarro = new Carro();

    // Asignar valores a los atributos
    miCarro.marca = "Toyota";
    miCarro.modelo = "Corolla";

    // Usar los métodos
    miCarro.arrancar();
    miCarro.frenar();
}
}

```

## 4.2 Atributos y métodos

- **Atributos (propiedades o variables):** describen las características de un objeto.

Los **atributos** representan las propiedades o variables de un objeto y permiten describir sus características, estado o información interna. Por ejemplo, en un videojuego, un personaje puede tener atributos como `vida`, `fuerza` o `nivel`.

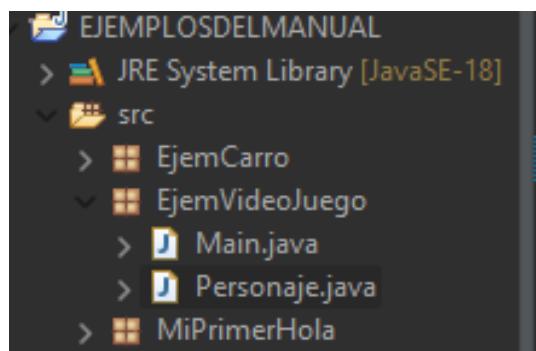
- **Métodos (funciones o acciones):** representan el comportamiento del objeto

Los **métodos**, por otro lado, definen las acciones o comportamientos que un objeto puede realizar. Siguiendo el ejemplo del personaje, los métodos podrían ser `atacar()`, `saltar()` o `recuperarVida()`.

→ Ejemplo en videojuegos:

- **Clase:** Personaje
- **Objeto:** Un personaje específico del juego (un guerrero o un mago).

Tenemos la siguiente Creacion de carpetas



/EJEMPLOSMANUAL/src/EjemVideoJuego/Personaje.java o Main.java

## Notas:

- Ambos archivos están en la carpeta EjemVideoJuego.
- Por eso **arriba de cada archivo** va la línea package EjemVideoJuego;.
- Desde Main puedes crear instancias de Personaje sin problema porque están en el mismo paquete.

## ARCHIVO CLASS Personaje.java

En esta clase se definen los atributos y métodos de un personaje dentro del videojuego. Se incluyen propiedades como el nombre y la vida, junto con acciones básicas como mostrar información, atacar y recibir daño

```
package EjemVideoJuego;

public class Personaje {
    // Atributos
    String nombre;
    int vida;

    // Método para mostrar información
    public void mostrarInfo() {
        System.out.println("El personaje es " + nombre + " con vida " + vida);
    }

    // Método para atacar
    public void atacar(String enemigo) {
        System.out.println(nombre + " ataca a " + enemigo + " con su espada.");
    }

    // Método para recibir daño
    public void recibirDanio(int puntos) {
        vida -= puntos;
        System.out.println(nombre + " ha recibido " + puntos + " de daño. Vida restante: " + vida);
    }
}
```

## ARCHIVO CLASS Main.java

En este archivo se crea un objeto de la clase `Personaje` y se utilizan sus métodos para simular interacciones, como atacar y recibir daño. De esta forma, se demuestra cómo instanciar una clase y aplicar sus comportamientos en un programa práctico.

```
package EjemVideoJuego;

public class Main {
    public static void main(String[] args) {
        // Crear un personaje
        Personaje guerrero = new Personaje();
        guerrero.nombre = "Thor";
        guerrero.vida = 100;

        // Usar los métodos
        guerrero.mostrarInfo();
        guerrero.atacar("Loki");
        guerrero.recibirDanio(20);
        guerrero.mostrarInfo();
    }
}
```

## 4.4 Encapsulamiento

El encapsulamiento es un principio de la Programación Orientada a Objetos que protege los atributos internos de un objeto para que no sean modificados directamente desde el exterior. En su lugar, se utilizan métodos públicos que controlan cómo se accede o modifica la información. Esto garantiza mayor seguridad, control y coherencia en el manejo de los datos.

### Ejemplo aplicado a los videojuegos:

La vida de un personaje no puede alterarse de forma directa, sino únicamente mediante métodos que representen acciones del juego, como recibir daño o curarse.

```

package EjemVideoJuego;

// Clase que representa a un personaje con vida encapsulada
class PersonajeEncapsulado {

    private int vida = 100;

    // Método público que permite modificar la vida de manera controlada
    // Aquí se resta la cantidad de vida cuando el personaje recibe daño
    public void recibirDaño(int cantidad) {
        vida -= cantidad;
    }

    // Método público para consultar la vida actual del personaje
    public int getVida() {
        return vida;
    }
}

// Clase principal para ejecutar el ejemplo de encapsulamiento
public class EjemploEncapsulamiento {
    public static void main(String[] args) {
        PersonajeEncapsulado p = new PersonajeEncapsulado();

        p.recibirDaño(30);

        System.out.println("Vida actual del personaje: " + p.getVida());
    }
}

```

### Salida de Consola:

```

Console X
<terminated> EjemploEncapsulamiento [Java Application]
Vida actual del personaje: 70

```

## 4.5 Herencia

La herencia permite que una clase (subclase) tome atributos y métodos de otra clase (superclase). Este mecanismo evita la duplicación de código y facilita la extensión de funcionalidades. Gracias a la herencia, se pueden crear diferentes tipos de personajes que comparten características básicas, pero que también agregan sus propias particularidades.

### Ejemplo aplicado a los videojuegos:

La clase **Personaje** es la base para crear otros personajes, como **Guerrero** o **Mago**, quienes heredan atributos comunes como el nombre y la vida, pero agregan sus propias habilidades.

```

package EjemVideoJuego;

// Clase base con atributos generales
class PersonajeBase {
    String nombre;
    int vida;
}

// Subclase que hereda de PersonajeBase
class Guerrero extends PersonajeBase {
    int fuerza; // Atributo adicional
}

public class EjemploHerencia {
    public static void main(String[] args) {
        // Crear un objeto Guerrero
        Guerrero g = new Guerrero();
        g.nombre = "Ares";
        g.vida = 120;
        g.fuerza = 80;

        // Mostrar atributos del guerrero
        System.out.println("Guerrero: " + g.nombre + " - Vida: " + g.vida +
                           " - Fuerza: " + g.fuerza);
    }
}

```

### Salida de Consola:

```

Console X
<terminated> EjemploHerencia [Java Application] C:\Pro...
Guerrero: Ares - Vida: 120 - Fuerza: 80

```

## 4.6 Polimorfismo

El polimorfismo permite que un mismo método tenga diferentes comportamientos según el objeto que lo utilice. Este principio hace que el código sea más flexible y extensible, ya que diferentes clases pueden redefinir métodos heredados para adaptarlos a su propia lógica.

### Ejemplo aplicado a los videojuegos:

El método **atacar()** se ejecuta de forma distinta en cada tipo de personaje: un guerrero realiza un ataque físico, mientras que un mago lanza un hechizo.

```

package EjemVideoJuego;

// Clase base con un método genérico
class PersonajePoli {
    void atacar() {
        System.out.println("El personaje ataca.");
    }
}

// Subclase Guerrero que redefine atacar()
class GuerreroPoli extends PersonajePoli {
    @Override
    void atacar() {
        System.out.println("El guerrero ataca con su espada.");
    }
}

// Subclase Mago que redefine atacar()
class MagoPoli extends PersonajePoli {
    @Override
    void atacar() {
        System.out.println("El mago lanza un hechizo.");
    }
}

public class EjemploPolimorfismo {
    public static void main(String[] args) {
        // Se crean personajes de distinto tipo
        PersonajePoli p1 = new GuerreroPoli();
        PersonajePoli p2 = new MagoPoli();

        // Cada objeto ejecuta atacar() de forma distinta
        p1.atacar(); // Guerrero
        p2.atacar(); // Mago
    }
}

```

Salida de Consola:

```

Console X
<terminated> EjemploPolimorfismo [Java Application]
El guerrero ataca con su espada.
El mago lanza un hechizo.

```

## 4.7 Abstracción

La abstracción consiste en enfocarse en lo esencial de un objeto y ocultar los detalles innecesarios. Este principio se implementa con **clases abstractas** e **interfaces**, que definen qué acciones deben realizar los objetos, sin especificar cómo se implementan. Esto facilita la organización del código y promueve la reutilización.

## Ejemplo aplicado a los videojuegos:

Todos los personajes pueden atacar, pero cada uno implementa su ataque de forma diferente (físico, mágico o a distancia).

```
package EjemVideoJuego;

// Clase abstracta que define un método atacar sin implementarlo
abstract class PersonajeAbs {
    abstract void atacar();
}

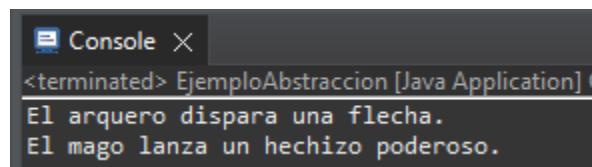
// Subclase Arquero que implementa el método atacar
class Arquero extends PersonajeAbs {
    @Override
    void atacar() {
        System.out.println("El arquero dispara una flecha.");
    }
}

// Subclase Mago que implementa el método atacar
class MagoAbs extends PersonajeAbs {
    @Override
    void atacar() {
        System.out.println("El mago lanza un hechizo poderoso.");
    }
}

public class EjemploAbstraccion {
    public static void main(String[] args) {
        // Crear objetos de tipo PersonajeAbs usando las subclases
        PersonajeAbs a = new Arquero();
        PersonajeAbs m = new MagoAbs();

        // Llamar al método atacar de cada objeto
        a.atacar();
        m.atacar();
    }
}
```

Salida de Consola:



The screenshot shows a terminal window titled "Console X" with the following text:  
<terminated> EjemploAbstraccion [Java Application] 0  
El arquero dispara una flecha.  
El mago lanza un hechizo poderoso.

## 5. Manejo de Strings en Java

### 5.1 Definición

En Java, un **String** es un tipo de dato que representa una **cadena de caracteres** (texto).

Aunque parece un tipo de dato simple, en realidad **String** es **una clase** dentro del paquete **java.lang**, lo que significa que tiene **propiedades y métodos** que permiten manipular textos de forma avanzada.

Por ejemplo, se puede crear un String de las siguientes dos maneras:

```
String mensaje = "Hola, soy un String";  
String mensaje2 = new String("Hola desde Java");
```

Ambas formas son válidas, pero la primera es la más común y recomendada por su simplicidad.

### 5.2 Características principales de los Strings

- Los Strings **no se pueden modificar directamente** (son **inmutables**). Cada vez que se cambia o concatenar un String, en realidad se crea uno nuevo.
- Se pueden **concatenar, comparar, buscar texto o formatear** fácilmente usando los métodos de la clase **String**.
- Se utilizan ampliamente para mostrar mensajes, guardar nombres, direcciones, descripciones y más.

### 5.3 Métodos más usados en la clase String

Método	Descripción	Ejemplo
<code>length()</code>	Devuelve la cantidad de caracteres del texto.	<code>mensaje.length()</code>
<code>contains("text o")</code>	Verifica si la cadena contiene una palabra específica.	<code>mensaje.contains("Hola")</code>
<code>endsWith("palabra")</code>	Verifica si la cadena termina con cierta palabra.	<code>mensaje.endsWith("Java")</code>
<code>concat("texto")</code>	Une dos cadenas.	<code>"Hola".concat("Mundo")</code>
<code>toUpperCase()</code>	Convierte todo el texto a mayúsculas.	<code>mensaje.toUpperCase()</code>
<code>toLowerCase()</code>	Convierte todo el texto a minúsculas.	<code>mensaje.toLowerCase()</code>
<code>replace("a", "e")</code>	Reemplaza caracteres o palabras dentro del texto.	<code>mensaje.replace("a", "e")</code>

#### 5.4 Ejemplo básico: uso de métodos de String

```
package Main;

public class Main {

    public static void main(String[] args) {

        // Crear cadenas

        String mensaje = "Hola, soy un String ñaña";

        String mensaje2 = new String("texto2222");

        System.out.println("Texto 1: " + mensaje);

        System.out.println("Texto 2: " + mensaje2);

        // Aplicar métodos de String

        String mensaje3 = "Hola, soy un String2";

        int cantidad = mensaje3.length();

        boolean contiene = mensaje3.contains("hola");

        boolean terminaCon = mensaje3.endsWith("String2");

        System.out.println("Cantidad de caracteres: " +
cantidad);

        System.out.println("¿Contiene 'hola'? " + contiene);
    }
}
```

```

        System.out.println("¿Termina con 'String2'? " +
terminaCon);

        String nuevoString = mensaje3.concat(" estamos en el
curso de Java");

        System.out.println("Concatenado: " + nuevoString);

    }

}

```

## 5.5 Formateo de Strings

El método `String.format()` permite **combinar texto con variables** numéricas o de texto dentro de una misma cadena.

Es muy útil cuando se necesita mostrar mensajes personalizados o con valores calculados.

```

package Main;

public class Main2 {

    public static void main(String[] args) {

        float valor = 77.13431f;

        float precio = 5000.50f;

        int cantidadComprada = 5;

        float total = precio * cantidadComprada;

        String mensaje = String.format("Bienvenidos %d al curso
%s. Valor decimal: %.4f",

```

```

        99, "Java", valor);

    String mensaje2 = String.format("El total de %d artículos
%s es de %.3f",
                                    cantidadComprada,
"Leche", total);

    System.out.println(mensaje);

    System.out.println(mensaje2);

}

}

```

## 5.6 Concatenación de Strings

Existen varias formas de unir cadenas de texto en Java:

### Versión 1: usando **concat()**

```

String nombre = "Codi";

String curso = "Curso de Java 9";

String mensaje = "Hola, bienvenido ".concat(nombre).concat(" al
").concat(curso).concat(".");

System.out.println(mensaje);

```

### Versión 2: usando el operador **+**

```
String nombre2 = "Julianito";
```

```
String salon = "Salón de Sistemas 4";  
  
String mensaje2 = "Hola, bienvenido " + nombre2 + " al " + salon  
+ ".";  
  
System.out.println(mensaje2);
```

## 5.7 Ejemplo aplicado a los videojuegos

En un videojuego, las cadenas (**String**) pueden utilizarse para mostrar mensajes en pantalla, dialogar con el jugador, mostrar el nombre del personaje, o imprimir resultados de acciones.

```
package VideoJuego;  
  
public class Personaje {  
  
    String nombre;  
  
    int vida;  
  
    Personaje(String nombre, int vida) {  
  
        this.nombre = nombre;  
  
        this.vida = vida;  
    }  
  
    void mostrarEstado() {  
  
        String mensaje = String.format("El personaje %s tiene %d  
puntos de vida.", nombre, vida);  
    }  
}
```

```

        System.out.println(mensaje);

    }

}

public class Main {

    public static void main(String[] args) {

        Personaje heroe = new Personaje("Link", 100);

        heroe.mostrarEstado();

        String mensajeBatalla = "¡" + heroe.nombre + " ha
derrotado al enemigo!";

        System.out.println(mensajeBatalla.toUpperCase());

    }

}

```

## 5.8 Buenas prácticas con Strings

- Evitar usar `+` en bucles (mejor usar `StringBuilder` para rendimiento).
- Usar nombres descriptivos en las variables (`nombreJugador`, `mensajeVictoria`).
- Aprovechar los métodos integrados de la clase `String` para búsquedas, validaciones y reemplazos.
- Evitar concatenar excesivamente cadenas grandes (usar `StringBuffer` o `StringBuilder` en esos casos).

## 6. Sobrecarga de métodos

### 6.1 Concepto

La **sobrecarga de métodos** es un principio fundamental de la **Programación Orientada a Objetos (POO)** que permite **definir varios métodos con el mismo nombre**, pero **con diferentes parámetros** (es decir, diferente cantidad o tipo de datos que reciben).

Esto significa que un método puede ejecutar **acciones similares**, pero adaptadas a distintas situaciones, según los **argumentos** que se le envíen al llamarlo.

En otras palabras, la sobrecarga de métodos **no crea métodos nuevos**, sino **diferentes versiones del mismo método**, cada una capaz de recibir distintos tipos de datos.

### 6.2 Características principales

- Todos los métodos **tienen el mismo nombre**, pero **difieren en su lista de parámetros** (número, tipo o ambos).
- La **firma del método** (nombre + parámetros) debe ser diferente para que Java los distinga.
- La **sobrecarga se resuelve en tiempo de compilación** (es decir, el compilador decide cuál método usar antes de ejecutar el programa).
- No depende del tipo de retorno (no se puede sobrecargar solo cambiando el tipo de dato que devuelve).

### 6.3 Ejemplo en la vida real

Imagina un **control remoto universal**. Tiene un mismo botón de “Encender”, pero puede funcionar para varios dispositivos:

- Si se presiona “Encender” frente al televisor → enciende el televisor.
- Si se presiona “Encender” frente al aire acondicionado → enciende el aire.

- Si se presiona “Encender” frente al decodificador → enciende el decodificador.

## 6.4 Ejemplo aplicado a los videojuegos

En un videojuego, un personaje puede atacar de diferentes formas:

- Si no tiene arma, ataca con las manos.
- Si tiene un arma, ataca con ella.
- Si además tiene poder o nivel, puede realizar un ataque especial.

La sobrecarga de métodos permite representar todas esas variaciones usando **el mismo método atacar()**, pero con parámetros distintos.

## 6.5 Código ejemplo

```
package VideoJuego;

public class Personaje {

    // Método atacar sin parámetros
    void atacar() {
        System.out.println("El personaje ataca con sus manos.");
    }

    // Método atacar con un parámetro
    void atacar(String arma) {
```

```

        System.out.println("El personaje ataca con " + arma +
" .");
    }

// Método atacar con dos parámetros

void atacar(String arma, int fuerza) {

    System.out.println("El personaje ataca con " + arma + " "
con una fuerza de " + fuerza + ".");
}

}

public class Main {

    public static void main(String[] args) {

        Personaje heroe = new Personaje();

        heroe.atacar();                                // Llama al primer
método (sin parámetros)

        heroe.atacar("espada");                      // Llama al segundo
método

        heroe.atacar("arco", 80);                     // Llama al tercero
    }
}

```

## 6.6 Salida en consola

El personaje ataca con sus manos.

El personaje ataca con espada.

El personaje ataca con arco con una fuerza de 80.

## 6.7 Explicación del código

- El método `atacar()` fue declarado **tres veces**, con **diferentes parámetros**.
- Según la llamada que haga el programa, Java selecciona el método correcto:
  - ◆ `atacar()` → sin parámetros.
  - ◆ `atacar(String arma)` → con una cadena de texto.
  - ◆ `atacar(String arma, int fuerza)` → con texto y número.
- Esto hace el código **más flexible, ordenado y reutilizable**, ya que no es necesario crear nombres distintos como `atacarSinArma()`, `atacarConArma()` o `atacarFuerte()`.

## 6.8 Ejemplo adicional: cálculo de daño sobrecargado

También se puede aplicar sobrecarga para **cálculos o acciones repetitivas** dentro del videojuego:

```
public class Combate {  
  
    // Daño base  
  
    int calcularDaño(int fuerza) {  
  
        return fuerza * 2;
```

```
}

// Daño con arma

int calcularDaño(int fuerza, int bonificacion) {

    return (fuerza + bonificacion) * 2;

}

// Daño con arma y multiplicador de nivel

    int calcularDaño(int fuerza, int bonificacion, float
multiplicador) {

        return (int) ((fuerza + bonificacion) * multiplicador);

    }

}

public class MainCombate {

    public static void main(String[] args) {

        Combate combate = new Combate();

            System.out.println("Daño básico: " +
combate.calcularDaño(10));

            System.out.println("Daño con arma: " +
combate.calcularDaño(10, 5));

            System.out.println("Daño con multiplicador: " +
combate.calcularDaño(10, 5, 1.5f));

    }

}
```

## 6.9 Buenas prácticas con la sobrecarga

- Usar sobrecarga **solo cuando las acciones sean similares** y compartan propósito.
- Mantener los nombres de parámetros **claros y coherentes**.
- No sobrecargar un método solo para hacer cosas completamente distintas.
- Evitar más de 3 o 4 versiones del mismo método si la lógica se vuelve confusa.
- Aprovecharla para mejorar la **legibilidad y reutilización del código**.

## 7. Constructores en la Programación Orientada a Objetos

### 7.1 Definición y propósito

En la Programación Orientada a Objetos, un **constructor** es un tipo especial de método que se **utiliza para crear e inicializar objetos**.

Su función principal es **asignar valores iniciales** a los atributos de una clase cuando nace un nuevo objeto. A diferencia de los métodos comunes, el constructor:

- **Tiene el mismo nombre que la clase.**
- **No devuelve ningún tipo de dato**, ni siquiera `void`.
- **Se ejecuta automáticamente** al momento de crear el objeto con la palabra clave `new`.

El constructor es como la **fábrica o el molde que da forma al objeto cuando “nace”**.

Por ejemplo, si se crea un personaje en un videojuego, el constructor se encarga de darle su nombre, puntos de vida y velocidad inicial, justo cuando aparece en el juego.

### 7.2 Tipos de constructores en Java

#### 7.2.1. Constructor por defecto

Es el constructor que **Java crea automáticamente** cuando no se define uno en la clase.

Este tipo de constructor **no recibe parámetros** y deja los atributos con sus valores por defecto:

- `0` para números enteros (`int`),
- `0.0` para decimales (`float` o `double`),
- `false` para valores booleanos,
- y `null` para textos (`String`).

Sin embargo, el programador también puede **declararlo manualmente** si desea inicializar los atributos con valores predeterminados.

**Ejemplo:**

```
public class Jugador {  
  
    String nombre;  
  
    int vida;  
  
    // Constructor por defecto  
    public Jugador() {  
  
        nombre = "Sin nombre";  
  
        vida = 100;  
  
    }  
  
    void mostrar() {  
  
        System.out.println("Jugador: " + nombre);  
  
        System.out.println("Vida: " + vida);  
  
    }  
}  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        Jugador j1 = new Jugador(); // Se usa el constructor por  
        // defecto  
  
        j1.mostrar();  
  
    }  
}
```

Podemos ver que en el código se:

Presenta que el jugador es creado, no se le pasa ningún dato.

El constructor le da un nombre por defecto (“Sin nombre”) y una vida inicial de 100 puntos.

### 7.2.2. Constructor con parámetros

Este tipo de constructor **permite personalizar los valores** de los atributos al crear el objeto. Recibe datos entre paréntesis y los utiliza para inicializar las variables internas de la clase.

**Ejemplo:**

```
public class Jugador {  
  
    String nombre;  
  
    int vida;  
  
    // Constructor con parámetros  
  
    public Jugador(String nombre, int vida) {  
  
        this.nombre = nombre;  
  
        this.vida = vida;  
    }  
  
    void mostrar() {  
  
        System.out.println("Jugador: " + nombre);  
  
        System.out.println("Vida: " + vida);  
    }  
}
```

```

public class Main {

    public static void main(String[] args) {

        Jugador heroe = new Jugador("Link", 150);

        Jugador mago = new Jugador("Zelda", 120);

        heroe.mostrar();

        mago.mostrar();

    }

}

```

**En este caso:**

El constructor **recibe datos (nombre y vida)** y los asigna directamente al personaje. Esto permite crear distintos jugadores con características únicas, sin tener que escribir líneas adicionales de código para configurar cada uno.

### 7.3 Ejemplo aplicado a los videojuegos

En los videojuegos, los constructores son muy útiles para **crear personajes, enemigos, armas o niveles**, asignando a cada uno sus propiedades básicas desde el inicio del juego.

Por ejemplo, al iniciar una partida, cada personaje puede nacer con un nombre, una cantidad de vida y una velocidad distinta.

El constructor hace ese proceso automático, evitando que el programador tenga que escribir código repetido cada vez que crea un nuevo personaje.

**Código ejemplo:**

```
package VideoJuego;
```

```
public class Personaje {  
    String nombre;  
    int vida;  
    float velocidad;  
  
    // Constructor con parámetros  
    public Personaje(String nombre, int vida, float velocidad) {  
        this.nombre = nombre;  
        this.vida = vida;  
        this.velocidad = velocidad;  
    }  
  
    void mostrarInfo() {  
        System.out.println("Personaje: " + nombre);  
        System.out.println("Vida: " + vida);  
        System.out.println("Velocidad: " + velocidad);  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {
```

```
Personaje guerrero = new Personaje("Guerrero", 120,  
5.5f);  
  
Personaje arquero = new Personaje("Arquero", 90, 7.0f);  
  
Personaje mago = new Personaje("Mago", 80, 6.2f);  
  
  
guerrero.mostrarInfo();  
  
arquero.mostrarInfo();  
  
mago.mostrarInfo();  
  
}  
  
}
```

#### 7.4 Salida de consola

Personaje: Guerrero

Vida: 120

Velocidad: 5.5

Personaje: Arquero

Vida: 90

Velocidad: 7.0

Personaje: Mago

Vida: 80

Velocidad: 6.2

## 7.5 Explicación del ejemplo

- Cada vez que se usa `new Personaje(...)`, el **constructor se ejecuta automáticamente**.
- Se asignan los valores iniciales de nombre, vida y velocidad a cada personaje.

Esto evita tener que escribir líneas como:

```
guerrero.nombre = "Guerrero";  
guerrero.vida = 120;  
guerrero.velocidad = 5.5f;
```

- porque todo se hace dentro del constructor.

## 7.6 Buenas prácticas al usar constructores

- Usar constructores para **asegurar que cada objeto empiece con valores válidos**.
- Si una clase tiene muchos atributos, crear **constructores sobrecargados** con diferentes combinaciones de parámetros.
- Usar `this` para diferenciar los nombres de los atributos de los parámetros.
- Si la clase requiere siempre ciertos datos al crearse (por ejemplo, el nombre del personaje), no usar un constructor vacío.
- Agregar comentarios en el código para aclarar qué inicializa cada constructor.

## 7.7 Ejemplo adicional: enemigos creados con constructores

```
package VideoJuego;

public class Enemigo {

    String tipo;
    int nivel;
    int vida;

    // Constructor

    public Enemigo(String tipo, int nivel) {
        this.tipo = tipo;
        this.nivel = nivel;
        this.vida = nivel * 50; // la vida depende del nivel
    }

    void mostrar() {
        System.out.println("Tipo de enemigo: " + tipo);
        System.out.println("Nivel: " + nivel);
        System.out.println("Vida total: " + vida);
    }
}
```

```
public class MainEnemigo {  
    public static void main(String[] args) {  
        Enemigo goblin = new Enemigo("Goblin", 2);  
        Enemigo dragón = new Enemigo("Dragón", 5);  
  
        goblin.mostrar();  
        dragón.mostrar();  
    }  
}
```

**Podemos ver que:** Aquí, los enemigos son creados con un tipo y nivel, el constructor calcula automáticamente la vida total según el nivel, mostrando cómo este tipo de método puede **automatizar tareas dentro de la creación de objetos**.

## 8. Relaciones entre clases

### 8.1 Introducción

En la Programación Orientada a Objetos (POO), los objetos no trabajan de forma aislada, sino que colaboran entre sí.

Cada clase representa una entidad con su propia información y comportamientos, pero en la mayoría de los programas, esas clases necesitan relacionarse para lograr un objetivo común.

Por ejemplo, en un videojuego, un personaje puede tener un arma, pertenecer a un equipo, o moverse dentro de un mapa.

Cada uno de esos elementos es una clase diferente, pero todas están conectadas mediante relaciones.

**Las tres relaciones más comunes en POO son:**

- **Asociación**
- **Agregación**
- **Composición**

**A continuación se explica cada una con ejemplos sencillos y prácticos en videojuegos.**

## 8.2 Asociación

La asociación se da cuando una clase utiliza o se comunica con otra, pero ambas pueden existir de forma independiente.

Ninguna depende totalmente de la otra para existir.

Es una relación temporal o de colaboración.

### Ejemplo de la vida real:

Un profesor enseña a un estudiante, pero ambos pueden existir por separado.

Si el profesor deja de enseñar, el estudiante sigue existiendo.

### Ejemplo en videojuegos:

Un Personaje puede tener un Arma para atacar, pero si el arma se rompe o se cambia, el personaje sigue existiendo y el arma también puede ser usada por otro personaje.

### Código ejemplo:

```
package VideoJuego;

public class Arma {

    String nombre;
    int daño;

    public Arma(String nombre, int daño) {
        this.nombre = nombre;
        this.daño = daño;
    }

    void mostrarInfo() {
        System.out.println("Arma: " + nombre + " | Daño: " +
daño);
    }
}
```

```
    }

}

public class Personaje {

    String nombre;

    Arma arma; // Asociación con la clase Arma


    public Personaje(String nombre, Arma arma) {

        this.nombre = nombre;

        this.arma = arma;

    }

    void atacar() {

        System.out.println(nombre + " ataca con " + arma.nombre +
" causando " + arma.daño + " de daño.");
    }

}

public class Main {

    public static void main(String[] args) {

        Arma espada = new Arma("Espada de Acero", 50);

        Personaje heroe = new Personaje("Aiden", espada);
```

```

        heroe.atacar();

        espada.mostrarInfo();

    }

}

```

**Lo que vemos en el ejemplo es:**

- El personaje usa un arma, pero no depende de ella para existir.
- Si se elimina el objeto **espada**, el personaje **Aiden** aún puede existir con otra arma o sin ella.
- Esta relación representa una asociación simple, ya que ambos objetos pueden vivir por separado.

### 8.3 Agregación

La agregación es un tipo especial de asociación en la que una clase contiene a otras como parte de su estructura, pero sin ser dueña total de ellas, las clases contenidas pueden existir independientemente del contenedor.

#### Ejemplo de la vida real:

Un equipo de fútbol tiene jugadores, pero si el equipo desaparece, los jugadores pueden pasar a otro equipo.

#### Ejemplo en videojuegos:

Un Escuadrón contiene varios Personajes, pero si el escuadrón se disuelve, los personajes pueden unirse a otro grupo o seguir existiendo solos.

#### Código ejemplo:

```

package VideoJuego;

import java.util.ArrayList;

```

```
public class Personaje {  
    String nombre;  
  
    public Personaje(String nombre) {  
        this.nombre = nombre;  
    }  
  
    void mostrar() {  
        System.out.println("Personaje: " + nombre);  
    }  
}  
  
public class Escuadron {  
    String nombreEscuadron;  
    ArrayList<Personaje> miembros; // Agregación: contiene  
    personajes  
  
    public Escuadron(String nombreEscuadron) {  
        this.nombreEscuadron = nombreEscuadron;  
        this.miembros = new ArrayList<>();  
    }  
}
```

```

void agregarMiembro(Personaje p) {
    miembros.add(p);
}

void mostrarMiembros() {
    System.out.println("Escuadrón: " + nombreEscuadron);
    for (Personaje p : miembros) {
        p.mostrar();
    }
}

public class MainAgregacion {
    public static void main(String[] args) {
        Personaje p1 = new Personaje("Guerrero");
        Personaje p2 = new Personaje("Arquero");
        Personaje p3 = new Personaje("Mago");

        Escuadron equipo = new Escuadron("Fuerza Elite");
        equipo.agregarMiembro(p1);
        equipo.agregarMiembro(p2);
    }
}

```

```
equipo.agregarMiembro(p3);  
  
equipo.mostrarMiembros();  
}  
}
```

### Explicación:

- El escuadrón contiene varios personajes, pero no los controla totalmente.
- Si se elimina el escuadrón, los personajes pueden seguir existiendo fuera de él.
- Esta relación muestra cómo una clase puede agrupar o manejar a otras sin ser su dueña.

## 8.4 Composición

La composición es la relación más fuerte entre clases, en este caso, una clase depende completamente de otra, lo que significa que si la clase principal se destruye, las que contiene también desaparecen.

### Ejemplo de la vida real:

Un corazón pertenece a un cuerpo.

Si el cuerpo deja de existir, el corazón también.

### Ejemplo en videojuegos:

Un Mapa tiene varios Escenarios o niveles, pero esos escenarios solo existen dentro de ese mapa.

Si el mapa se elimina, los escenarios desaparecen también.

### Código ejemplo:

```
package VideoJuego;

import java.util.ArrayList;

public class Escenario {

    String nombre;

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

    void mostrar() {
        System.out.println("Escenario: " + nombre);
    }
}
```

```
}

public class Mapa {

    String nombreMapa;

    ArrayList<Escenario> escenarios;

    // Composición: los escenarios son creados dentro del mapa

    public Mapa(String nombreMapa) {

        this.nombreMapa = nombreMapa;

        this.escenarios = new ArrayList<>();

        crearEscenarios();

    }

    private void crearEscenarios() {

        escenarios.add(new Escenario("Bosque Oscuro"));

        escenarios.add(new Escenario("Castillo del Rey"));

        escenarios.add(new Escenario("Cueva de Lava"));

    }

    void mostrarMapa() {

        System.out.println("Mapa: " + nombreMapa);

        for (Escenario e : escenarios) {


```

```

        e.mostrar();

    }

}

public class MainComposicion {

    public static void main(String[] args) {

        Mapa mapa1 = new Mapa("Aventuras del Reino");

        mapa1.mostrarMapa();

    }

}

```

**Tenemos en cuenta que:**

- Los escenarios no se crean fuera del mapa; nacen dentro de él.
- Si se destruye el mapa, automáticamente se eliminan todos los escenarios que contiene.
- Esta relación refleja una dependencia total, donde una clase controla completamente la vida de la otra.

## 8.5 Diferencias entre asociación, agregación y composición

Tipo de relación	Dependencia	Ejemplo en videojuegos	Nivel de unión
Asociación	Las clases pueden existir por separado	Personaje usa un Arma	Débil
Agregación	Una clase agrupa otras, pero no las posee	Escuadrón con varios Personajes	Media
Composición	Una clase depende totalmente de la otra	Mapa contiene Escenarios	Fuerte

## 8.6 En Conclusión

En la Programación Orientada a Objetos, las relaciones entre clases permiten que un programa sea más organizado, realista y modular.

En el contexto de los videojuegos, estos tres tipos de relaciones hacen posible que los distintos elementos del juego —como personajes, armas, escenarios o equipos— interactúen de forma natural, como en la vida real.

La asociación, agregación y composición reflejan los distintos grados de conexión entre los objetos, desde simples colaboraciones hasta dependencias totales.

## 9. Interfaces

### 9.1 Concepto

En la Programación Orientada a Objetos, una interfaz define un comportamiento común que varias clases pueden compartir, sin importar cómo lo implementen.

Se puede decir que una interfaz establece un contrato, donde las clases que la utilizan se comprometen a cumplir con los métodos que esta define.

A diferencia de una clase normal, una interfaz no tiene código dentro de sus métodos, solo define los nombres de los comportamientos que las clases deben tener.

**Ejemplo en videojuegos:** En un juego, varios personajes o enemigos pueden recibir daño o atacar, pero cada uno lo hace de forma diferente.

Una interfaz permite asegurar que todos tengan el mismo método, aunque su comportamiento varíe según el tipo de personaje.

### 9.2 Ejemplo aplicado a videojuegos

```
// Interfaz que define un comportamiento común

interface Atacable {

    void recibirDaño(int cantidad);

}

// Clase que implementa la interfaz

class Enemigo implements Atacable {

    int vida = 100;

    public void recibirDaño(int cantidad) {
```

```

        vida -= cantidad;

        System.out.println("El enemigo recibió " + cantidad + " de daño. Vida restante: " + vida);

    }

}

// Otra clase que también implementa la interfaz

class JefeFinal implements Atacable {

    int vida = 500;

    public void recibirDaño(int cantidad) {

        vida -= cantidad / 2; // Tiene más resistencia

        System.out.println("El jefe final recibió " + (cantidad / 2) + " de daño. Vida restante: " + vida);

    }

}

public class Main {

    public static void main(String[] args) {

        Enemigo enemigo = new Enemigo();

        JefeFinal boss = new JefeFinal();

        enemigo.recibirDaño(40);
    }
}

```

```
    boss.recibirDaño(40);  
}  
}
```

**El código permite ver que:**

- Tanto **Enemigo** como **JefeFinal** implementan la interfaz **Atacable**.
- Cada uno usa el mismo método **recibirDaño()**, pero lo ejecuta de forma distinta.
- Esto permite tener comportamientos comunes con resultados personalizados, ideal para videojuegos con diferentes tipos de enemigos.

## 10. Clases abstractas

### 10.1 Concepto

Una clase abstracta es una clase que no se puede usar directamente para crear objetos, sino que sirve como modelo base para otras clases permitiendo definir atributos y métodos comunes, dejando algunos métodos sin definir, para que las clases hijas los completen según su necesidad.

#### Ejemplo en videojuegos:

Un videojuego puede tener distintos tipos de personajes: guerreros, magos, arqueros, entre otros, todos comparten características como vida o nivel, pero cada uno ataca de manera distinta.

Una clase abstracta puede definir esa base común y dejar que cada personaje implemente su propio ataque.

### 10.2 Ejemplo aplicado a videojuegos

```
// Clase abstracta base

abstract class Personaje {

    String nombre;

    int vida;

    Personaje(String nombre, int vida) {

        this.nombre = nombre;

        this.vida = vida;

    }

    // Método común

    void mostrarInfo() {
```

```
        System.out.println("Personaje: " + nombre + " | Vida: " +
vida);

    }

    // Método abstracto (debe ser implementado por las clases
hijas)

    abstract void atacar();

}

// Clases que heredan de Personaje

class Guerrero extends Personaje {

    Guerrero(String nombre, int vida) {

        super(nombre, vida);

    }

    void atacar() {

        System.out.println(nombre + " ataca con su espada.");

    }

}

class Mago extends Personaje {

    Mago(String nombre, int vida) {

        super(nombre, vida);

    }

}
```

```

    }

void atacar() {
    System.out.println(nombre + " lanza un hechizo mágico.");
}

}

public class Main {
    public static void main(String[] args) {
        Guerrero g = new Guerrero("Arthas", 150);
        Mago m = new Mago("Merlín", 100);
        g.mostrarInfo();
        g.atacar();

        m.mostrarInfo();
        m.atacar();
    }
}

```

- La clase **Personaje** no puede crearse directamente, porque es abstracta.
- Las clases **Guerrero** y **Mago** heredan de ella y definen su propia forma de atacar.
- Permite mantener un diseño organizado y claro, donde los personajes comparten lo esencial, pero se diferencian en sus acciones.

## 11. ARREGLOS

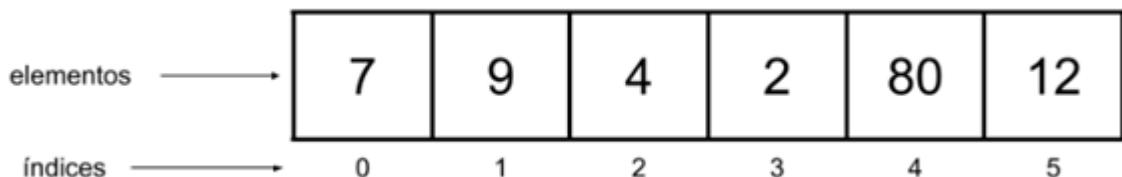
### 11.1 ¿Qué son los arreglos en Java?

Los arreglos en Java son estructuras de datos fundamentales que permiten almacenar y manejar una colección de elementos del mismo tipo dentro de una sola variable. Esto significa que, en lugar de declarar múltiples variables individuales para cada dato, un arreglo ofrece una manera ordenada y compacta de agruparlos bajo un mismo nombre.

Desde un punto de vista técnico, los arreglos en Java son **objetos** que se crean en memoria y cuentan con una **longitud fija**, determinada en el momento de su inicialización. Cada elemento dentro del arreglo ocupa una posición específica, conocida como **índice**, que comienza desde el número cero. Por ejemplo, el primer elemento de un arreglo se encuentra en la posición 0, el segundo en la posición 1, y así sucesivamente.

Además, los arreglos permiten un acceso directo a sus elementos, lo que significa que el programa puede recuperar o modificar cualquier valor de manera inmediata utilizando su índice. Esta característica hace que sean estructuras muy eficientes para el manejo de datos, especialmente en operaciones que requieren recorrer o procesar grandes volúmenes de información.

En el ámbito de la programación, los arreglos son ampliamente utilizados para representar listas de valores como números, nombres, notas o cualquier tipo de datos repetitivos. También sirven como base para estructuras más complejas, como **matrices**, **vectores** o **colecciones dinámicas**.



## **11.2. ¿Para qué se utilizan los arreglos en un programa?**

Se utilizan para guardar conjuntos de datos relacionados, como una lista de números, nombres o calificaciones. Esto ayuda a simplificar el código y permite procesar grandes cantidades de información de manera eficiente mediante ciclos.

## **11.3. ¿Cómo se declara un arreglo en Java?**

En Java, un arreglo se declara indicando primero el tipo de dato que almacenará, seguido de corchetes ([] ) y luego el nombre del arreglo. Esta estructura le permite al programa identificar que se trata de una colección de elementos del mismo tipo.

Por ejemplo, para declarar un arreglo de números enteros se puede escribir:

```
int[] numeros;
```

En esta línea, int representa el tipo de dato de los elementos (números enteros), los corchetes [] indican que se trata de un arreglo, y numeros es el nombre de la variable que lo identifica.

Sin embargo, en este punto el arreglo solo está declarado, no inicializado, lo que significa que aún no tiene espacio reservado en memoria. Para poder usarlo, se debe crear una instancia del arreglo, especificando su tamaño mediante la palabra clave new:

```
numeros = new int[5];
```

Aquí se está creando un arreglo con espacio para 5 elementos enteros, cuyas posiciones van del índice 0 al 4.

También es posible declarar e inicializar un arreglo en una sola línea:

```
int[] numeros = new int[5];
```

O incluso asignarle valores directamente al momento de la declaración:

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

En este caso, Java creará automáticamente un arreglo de tamaño 5 con los valores indicados.

## 11.4. ¿Cómo se almacenan y acceden los datos dentro de un arreglo?

En Java, los datos dentro de un arreglo se almacenan de forma **secuencial** en posiciones llamadas **índices**, que representan la ubicación de cada elemento dentro del arreglo. Estos índices comienzan desde el número **cero (0)** y se incrementan de uno en uno hasta llegar al último elemento, cuyo índice siempre es **tamaño del arreglo – 1**.

Cada elemento ocupa una posición específica en la memoria, lo que permite que Java acceda rápidamente a cualquier valor mediante su índice. Esta característica convierte a los arreglos en una estructura de datos **muy eficiente** para la lectura o modificación de elementos.

Para acceder o modificar un dato, se utiliza el **nombre del arreglo** seguido del **índice** entre corchetes. Por ejemplo:

```
int[] edades = {18, 20, 22, 24, 26};
```

```
// Acceder al primer elemento  
System.out.println(edades[0]); // Imprime 18
```

```
// Modificar el tercer elemento  
edades[2] = 23;
```

```
// Mostrar el nuevo valor  
System.out.println(edades[2]); // Imprime 23
```

## 11.5. ¿Cuál es la diferencia entre un arreglo y una variable simple?

La principal diferencia entre un arreglo y una variable simple radica en la cantidad de valores que pueden almacenar.

Una variable simple solo puede contener un valor a la vez, mientras que un arreglo puede guardar múltiples valores del mismo tipo bajo un mismo nombre.

**Por ejemplo:**

```
int edad = 20; // Variable simple: solo un valor  
int[] edades = {18, 20, 22, 24, 26}; // Arreglo: varios valores
```

En el caso anterior, edad almacena únicamente un número entero, mientras que edades puede contener una lista completa de edades.

Otra diferencia importante es que los arreglos permiten **recorrer sus elementos mediante ciclos**, lo cual facilita procesar datos en masa, como calcular promedios o buscar valores específicos. Además, los arreglos se almacenan **de forma contigua en memoria**, lo que optimiza el acceso a la información.

### 11.6. ¿Qué tipo de datos se pueden guardar dentro de un arreglo?

Se pueden almacenar datos primitivos como int, double, char, boolean, o también objetos de clases, como String o incluso arreglos dentro de arreglos.

### 11.7. ¿Cómo se puede recorrer un arreglo utilizando un ciclo for o foreach?

Con un ciclo for tradicional se puede recorrer usando un índice, por ejemplo:

```
for(int i = 0; i < arreglo.length; i++) .
```

El ciclo foreach simplifica el recorrido:

```
for(int num : arreglo) .
```

### 11.8. ¿Qué sucede si se intenta acceder a una posición fuera del tamaño del arreglo?

Cuando en Java se intenta acceder a una posición que no existe dentro del tamaño establecido del arreglo, el programa genera un error denominado

`ArrayIndexOutOfBoundsException.`

Este error ocurre porque los arreglos en Java poseen una longitud fija y cada posición válida está delimitada por índices que comienzan desde 0 hasta  $n-1$ , donde  $n$  es la cantidad total de elementos que contiene el arreglo.

Podemos ver por ejemplo si un arreglo tiene 5 elementos, sus índices válidos serán del 0 al 4. Intentar acceder al índice 5 provocará que el programa se detenga abruptamente y muestre dicha excepción.

Este tipo de error es común en principiantes, por lo que se recomienda siempre verificar el tamaño del arreglo utilizando la propiedad `.length` antes de acceder a sus elementos.

## 11.9. Ejemplo de un programa en Java que use un arreglo

### 11.9.1 Para guardar 5 notas y mostrar su promedio

Un ejemplo sencillo de cómo se puede utilizar un arreglo en Java para almacenar cinco notas y calcular su promedio:

```
public class PromedioNotas {
    public static void main(String[] args) {
        double[] notas = {4.0, 3.5, 5.0, 4.2, 3.8};
        double suma = 0;

        // Recorre el arreglo sumando cada nota
        for (double nota : notas) {
            suma += nota;
        }

        // Calcula el promedio
        double promedio = suma / notas.length;

        // Muestra el resultado en consola
        System.out.println("El promedio es: " + promedio);
    }
}
```

Luego, mediante un bucle `foreach`, se suman todas las notas y se divide la suma total entre el tamaño del arreglo (`notas.length`) para obtener el promedio.

Finalmente, se imprime el resultado en pantalla. Este tipo de ejercicio demuestra cómo los arreglos facilitan el manejo de conjuntos de datos sin necesidad de declarar múltiples variables.

### **11.10. Explique de los arreglos razón que son útiles en la programación**

Los arreglos son una de las estructuras más importantes en la programación, ya que permiten almacenar y manipular conjuntos de datos relacionados de manera organizada.

Gracias a ellos, un programador puede trabajar con grandes volúmenes de información —como listas de números, nombres o calificaciones— sin necesidad de declarar variables por separado.

Además, los arreglos ofrecen eficiencia y estructura al permitir recorrer y procesar sus elementos mediante ciclos, lo que simplifica el desarrollo de algoritmos y reduce errores en el código.

Desde un punto de vista técnico, los arreglos también son la base para estructuras de datos más avanzadas, como listas, matrices, vectores y colecciones dinámicas.

Por ello, comprender su funcionamiento resulta esencial para construir programas más sólidos, organizados y fáciles de mantener.

## 12. Arreglo Ejercicios Orientado A Videojuegos

### 12.1. Ejercicio 1 EL DADO:

#### 12.1.1 Descripción del código

El primer juego de dados en Java simula una partida sencilla donde el jugador lanza dos dados virtuales y gana si la suma es 7 o 11; de lo contrario, se le invita a intentarlo de nuevo. Aunque está escrito de forma estructurada, puede adaptarse fácilmente al enfoque de programación orientada a objetos creando clases como `Dado`, `Jugador` y `Juego`, lo que permite organizar mejor el código, reutilizar componentes y aplicar principios como encapsulamiento y responsabilidad única.

#### 12.1.2 Código

```
package ejercicios;
// Importamos las librerías Scanner y Random
import java.util.Scanner;
import java.util.Random;
public class Dados {
    public static void main(String[] args) {
        System.out.println("Bienvenido al juego de los DADOS \n");
        Scanner teclado = new Scanner(System.in);
        Random rr = new Random();
        // Generamos dos números aleatorios entre 1 y 6 (como dos dados reales)
        int dado1 = rr.nextInt(6) + 1;
        int dado2 = rr.nextInt(6) + 1;
        // Sumamos los valores de los dos dados
        int suma = dado1 + dado2;
        // Mostramos el resultado al jugador
        System.out.println("El resultado de los dados es: " + dado1 + " y " + dado2 +
                           " → Total: " + suma + "\n");
        // Condición de victoria: si la suma es 7 o 11, el jugador gana
        if (suma == 7 || suma == 11) {
            System.out.println("¡Ganaste! Eres un maestro de los dados !");
        } else {
            System.out.println("Perdiste Muy malo manito, vuelve a intentarlo.");
        }
        teclado.close(); // Cerramos el scanner para evitar fugas de recursos
    }
}
```

### **12.1.3 Paso a paso del código**

#### **I. Paquete y librerías:**

```
package ejercicios;  
  
import java.util.Scanner;  
  
import java.util.Random;
```

- Se define el paquete ejercicios.

#### **I. Definición de la clase y método principal:**

```
public class Dados {  
  
    public static void main(String[] args) {
```

- Se define la clase Dados.
- El método main es donde se ejecuta todo el programa.

#### **II. Mensaje de bienvenida:**

```
System.out.println("🎲 Bienvenido al juego de los DADOS 🎲\n");
```

- Muestra en pantalla un mensaje dando la bienvenida al jugador.

#### **III. Creación de objetos Scanner y Random:**

```
Scanner teclado = new Scanner(System.in);  
  
Random rr = new Random();
```

- Scanner teclado se crea para poder leer datos (aunque no se usa aquí).
- Random rr se usa para generar números aleatorios, simulando el lanzamiento de los dados.

#### **IV. Generar los valores de los dados:**

```
int dado1 = rr.nextInt(6) + 1;  
  
int dado2 = rr.nextInt(6) + 1;
```

- rr.nextInt(6) genera un número entre 0 y 5, por eso se suma 1 para obtener valores entre 1 y 6, como un dado real.
- dado1 y dado2 guardan los valores de los dos dados.

#### **V. Calcular la suma de los dados:**

```
int suma = dado1 + dado2;
```

- Se suman los valores de los dos dados y se guarda el resultado en suma.

#### **VI. Mostrar el resultado al jugador:**

```
System.out.println("El resultado de los dados es: " + dado1 + " y  
" + dado2 +  
" → Total: " + suma + "\n");
```

- Se imprime en pantalla el valor de cada dado y el total.

#### **VII. Verificar si el jugador gana o pierde:**

```
if (suma == 7 || suma == 11) {  
  
    System.out.println("¡Ganaste! 🎉 Eres un maestro de los dados  
😎");  
  
} else {  
  
    System.out.println("Perdiste 😞 Muy malo manito, vuelve a  
intentarlo.");  
  
}
```

- Si la suma es 7 u 11, el jugador gana.
- Si no, el jugador pierde.

### VIII. Cerrar el Scanner:

```
teclado.close();
```

- Se cierra el objeto Scanner para liberar recursos y evitar fugas de memoria.

## 12.2 Ejercicio 2 PIEDRA, PAPEL O TIJERAS:

### 12.2.1 Descripción del código

El juego de piedra, papel o tijeras en Java permite al jugador enfrentarse a la máquina eligiendo una de las tres opciones: piedra, papel o tijeras la máquina selecciona aleatoriamente su jugada, y el programa compara ambas elecciones para determinar al ganador según las reglas clásicas, el ejercicio es ideal para principiantes, ya que utiliza estructuras básicas como condicionales, entrada por teclado, generación aleatoria y manejo de cadenas, facilitando el aprendizaje de lógica y control de flujo en programación orientada a objetos.

## 12.2.2 Código

```
package ejercicios;
import java.util.Scanner;
import java.util.Random;
public class PiedraPapelTijera {
    public static void main(String[] args) {
        System.out.println("Bienvenido al juego de Piedra, Papel o Tijeras \n");
        Scanner teclado = new Scanner(System.in);
        Random rr = new Random();
        // Pedir al jugador que ingrese su opción
        System.out.println("Escribe una opción: piedra, papel o tijeras");
        String jugador = teclado.nextLine().toLowerCase(); // Convertir a minúscula para evitar errores
        String maquina = ""; // variable para la opción de la máquina
        // Generar opción aleatoria para la máquina (0 = piedra, 1 = papel, 2 = tijeras)
        int opmaquina = rr.nextInt(3);
        if(opmaquina == 0) {
            maquina = "piedra";
        } else if(opmaquina == 1) {
            maquina = "papel";
        } else {
            maquina = "tijeras";
        }
        System.out.println("La máquina eligió: " + maquina + "\n");
        // Comparar opciones y determinar el ganador
        if(jugador.equals(maquina)) {
            System.out.println("¡Empate! \n");
        } else if (
            (jugador.equals("piedra") && maquina.equals("tijeras")) ||
            (jugador.equals("tijeras") && maquina.equals("papel")) ||
            (jugador.equals("papel") && maquina.equals("piedra"))
        ) {
            System.out.println("¡El jugador es el ganador! ");
        } else {
            System.out.println("El jugador ha perdido ");
        }
        teclado.close(); // Cerrar el scanner
    }
}
```

## 12.2.3 Paso a paso del código

- I. El programa da la bienvenida al jugador y explica el juego.

```
System.out.println("🎮 Bienvenido al juego de Piedra, Papel o  
Tijeras 🎮\n");
```

- II. Se preparan dos cosas: una para leer lo que el jugador escribe y otra para que la máquina elija al azar su jugada.

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

```
Random rr = new Random();
```

**III. El jugador escribe su opción: piedra, papel o tijeras, y el programa la convierte a minúsculas para no tener problemas.**

```
System.out.println("Escribe una opción: piedra, papel o  
tijeras");
```

```
String jugador = teclado.nextLine().toLowerCase();
```

**IV. La máquina elige su jugada al azar.**

```
String maquina = "";  
  
int opmaquina = rr.nextInt(3); // 0,1,2  
  
if(opmaquina == 0) {  
  
maquina = "piedra";  
  
} else if(opmaquina == 1) {  
  
maquina = "papel";  
  
} else {  
  
maquina = "tijeras";  
  
}
```

**V. Se muestra qué eligió la máquina.**

```
System.out.println("La máquina eligió: " + maquina + "\n");
```

**VI. El programa compara las opciones y decide quién gana:**

```
if(jugador.equals(maquina)) {  
    System.out.println("¡Empate! \n");  
}  
else if (  
    (jugador.equals("piedra") && maquina.equals("tijeras")) ||  
    (jugador.equals("tijeras") && maquina.equals("papel")) ||  
    (jugador.equals("papel") && maquina.equals("piedra"))  
) {  
    System.out.println("El jugador es el ganador! ");  
}  
else {  
    System.out.println("El jugador ha perdido ");  
}
```

- Si son iguales → empate.
- Si el jugador gana según las reglas → mensaje de victoria.
- Si no → mensaje de derrota.

**VII. Se cierra la lectura y termina el juego.**

```
teclado.close();
```

## 12.3. Ejercicio 3 EL BattleRoyale:

### 12.3.1 Descripción del código

El programa BattleRoyale.java simula un juego donde 10 jugadores se atacan entre sí hasta que solo queda uno; cada jugador tiene 100 de vida, los ataques son aleatorios y el código usa arreglos para guardar la vida y los nombres, mostrando en cada turno quién ataca a quién y finalmente anunciando al ganador.

### 12.3.2 Código

```
package ejercicios;
import java.util.Random;
public class BattleRoyale {
    public static void main(String[] args) {
        int n = 10; // número de jugadores
        int[] vida = new int[n]; // arreglo que almacena la vida de cada jugador
        String[] jugadores = new String[n]; // nombres
        Random rand = new Random(); //números aleatorios
        // Inicializar vida y nombres de los jugadores
        for (int i = 0; i < n; i++) {
            vida[i] = 100; //100 de vida
            jugadores[i] = "Jugador " + (i + 1); //nombre
        }
        int turno = 1; //contador de turnos
        // Bucle principal: se repite mientras queden más de un jugador vivo
        while (contarVivos(vida) > 1) {
            System.out.println("Turno " + turno + ":");

            // Cada jugador vivo ataca a otro jugador aleatorio vivo
            for (int i = 0; i < n; i++) {
                if (vida[i] > 0) { // solo jugadores vivos atacan
                    int objetivo;
                    // Seleccionar un objetivo aleatorio que no sea el mismo y que esté vivo
                    do {
                        objetivo = rand.nextInt(n);
                    } while (objetivo == i || vida[objetivo] <= 0);

                    // daño aleatorio entre 10 y 30
                    int daño = rand.nextInt(21) + 10;
                    vida[objetivo] -= daño;
                    if (vida[objetivo] < 0) vida[objetivo] = 0; // evitar vida negativa

                    // Mostrar acción del turno
                    System.out.println(jugadores[i] + " ataca a " + jugadores[objetivo] + " con " + daño + " de daño. Vida restante: " + vida[objetivo]);
                }
                System.out.println();
                turno++;
            }
            // Mostrar el ganador
            for (int i = 0; i < n; i++)
                if (vida[i] > 0)
                    System.out.println("El ganador es " + jugadores[i] + " con " + vida[i] + " de vida!");
        }
        // Función que cuenta cuántos jugadores están vivos (vida > 0)
        public static int contarVivos(int[] vida) {
            int c = 0;
            for (int v : vida) if (v > 0) c++;
            return c;
        }
    }
}
```

### **12.3.3 Paso a paso del código**

#### **I. Inicio del juego:**

Se define que habrá 10 jugadores y se preparan dos arreglos: uno para guardar la vida de cada jugador y otro para sus nombres.

#### **II. Asignar vida y nombres:**

Cada jugador empieza con 100 de vida y se le asigna un nombre como "Jugador 1", "Jugador 2", etc.

#### **III. Preparar herramientas aleatorias:**

Se crea un objeto Random para generar números aleatorios que decidirán los ataques y a quién atacan.

#### **IV. Iniciar los turnos:**

Se usa un bucle que continúa mientras queden más de un jugador vivo. Se lleva un contador de turnos para mostrar el progreso del juego.

#### **V. Ataques de los jugadores:**

- Cada jugador que esté vivo selecciona al azar a otro jugador vivo como objetivo.
- Se genera un daño aleatorio entre 10 y 30 y se resta de la vida del objetivo.
- Si la vida del jugador atacado baja de 0, se ajusta a 0.
- Se muestra en pantalla quién atacó a quién, cuánto daño hizo y la vida que le queda al objetivo.

#### **VI. Comprobar jugadores vivos:**

Cada turno se verifica cuántos jugadores siguen vivos para decidir si el juego continúa

## VII. **Final del juego:**

Cuando solo queda un jugador vivo, se anuncia como ganador mostrando su nombre y la vida que le queda.

### **12.4. Ejercicio 4 Combate Goblins Arreglo:**

#### **12.4.1 Descripción del código**

El programa CombateGoblinsARREGLO.java simula un combate donde tres personajes (Caballero, Mago y Arquero) atacan a un grupo de goblins; cada goblin tiene cierta vida, los ataques les quitan puntos de vida y al final el programa muestra cuántos goblins sobreviven o si todos han sido derrotados, usando arreglos para llevar el control de la vida de cada goblin.

## 12.4.2 Código

```
package ejercicios;

public class CombateGoblinsARREGLO {

    public static void main(String[] args) {
        // Arreglo que representa la vida de cada goblin
        int[] goblins = {50, 60, 40, 30}; // goblins[0] = 50 de vida, goblins[1] = 60, etc.
        // Daño que infilige cada personaje
        int dañoCaballero = 20;
        int dañoMago = 25;
        int dañoArquero = 15;
        System.out.println("¡Combate contra los goblins!");
        System.out.println("Goblins iniciales:");
        mostrarGoblins(goblins); // Mostrar la vida inicial de los goblins
        // Caballero ataca a todos los goblins
        System.out.println("\nEl Caballero ataca...");
        atacarGoblins(goblins, dañoCaballero);
        mostrarGoblins(goblins); // Mostrar vida después del ataque
        // Mago ataca a todos los goblins
        System.out.println("\nEl Mago ataca...");
        atacarGoblins(goblins, dañoMago);
        mostrarGoblins(goblins);
        // Arquero ataca a todos los goblins
        System.out.println("\nEl Arquero ataca...");
        atacarGoblins(goblins, dañoArquero);
        mostrarGoblins(goblins);

        // Contar cuántos goblins siguen vivos
        int goblinsVivos = contarGoblinsVivos(goblins);
        System.out.println("\nGoblins sobrevivientes: " + goblinsVivos);
        if (goblinsVivos == 0) {
            System.out.println("Todos los goblins han sido derrotados!");
        } else {
            System.out.println("Algunos goblins siguen en pie!");
        }
    }

    // Función para mostrar la vida de cada goblin
    public static void mostrarGoblins(int[] goblins) {
        for (int i = 0; i < goblins.length; i++) {
            System.out.println("Goblin " + (i + 1) + ": " + goblins[i] + " de vida");
        }
    }

    // Función que aplica daño a todos los goblins vivos
    public static void atacarGoblins(int[] goblins, int daño) {
        for (int i = 0; i < goblins.length; i++) {
            if (goblins[i] > 0) { // solo atacar goblins que tengan vida
                goblins[i] -= daño; // Restar el daño
                if (goblins[i] < 0) { // evitar valores negativos
                    goblins[i] = 0;
                }
            }
        }
    }

    // Función que cuenta cuántos goblins tienen vida mayor a 0
    public static int contarGoblinsVivos(int[] goblins) {
        int vivos = 0;
        for (int vida : goblins) {
            if (vida > 0) {
                vivos++;
            }
        }
        return vivos;
    }
}
```

### 12.4.3 Paso a paso del código

#### I. Preparar el juego:

Se define que habrá 10 jugadores y se crean dos arreglos: uno para la **vida** de cada jugador y otro para los **nombres**.

```
int n = 10;  
  
int[] vida = new int[n];  
  
String[] jugadores = new String[n];
```

#### II. Inicializar jugadores:

Cada jugador empieza con 100 de vida y se le asigna un nombre como "Jugador 1",

```
for (int i = 0; i < n; i++) {  
  
    vida[i] = 100;  
  
    jugadores[i] = "Jugador " + (i + 1);  
  
}
```

#### III. Preparar el generador de números aleatorios:

Se crea un objeto Random para decidir a quién atacará cada jugador y cuánto daño hace.

```
Random rand = new Random();
```

#### IV. Iniciar los turnos de ataque:

Mientras queden más de un jugador vivo, se repite un bucle que representa cada turno.

```
while (contarVivos(vida) > 1) {  
  
    System.out.println("Turno " + turno + ":");  
  
    // ataques de cada jugador  
  
}
```

## V. Cada jugador ataca a otro:

- Solo los jugadores vivos atacan.
- Se elige un objetivo al azar que también esté vivo y que no sea él mismo.
- Se genera un daño aleatorio entre 10 y 30 y se resta de la vida del objetivo.
- Se imprime en pantalla quién atacó a quién y la vida restante del objetivo.

```
for (int i = 0; i < n; i++) {  
  
    if (vida[i] > 0) {  
  
        int objetivo;  
  
        do {  
  
            objetivo = rand.nextInt(n);  
  
        } while (objetivo == i || vida[objetivo] <= 0);  
  
        int daño = rand.nextInt(21) + 10;  
  
        vida[objetivo] -= daño;  
  
        if (vida[objetivo] < 0) vida[objetivo] = 0;  
  
        System.out.println(jugadores[i] + " ataca a " +  
            jugadores[objetivo] + " con " + daño + " de daño. Vida restante:  
            " + vida[objetivo]);  
  
    }  
  
}
```

## VI. Incrementar el turno:

Se aumenta el contador de turnos después de cada ronda de ataques.

```
turno++;
```

## VII. Mostrar el ganador:

Cuando solo queda un jugador vivo, se imprime su nombre y la vida que le queda.

```
for (int i = 0; i < n; i++)  
  
if (vida[i] > 0)  
  
System.out.println("¡El ganador es " + jugadores[i] + " con "  
+ vida[i] + " de vida!");
```

### VIII. Función auxiliar contarVivos:

Esta función cuenta cuántos jugadores aún tienen vida mayor a 0, para controlar el bucle principal.

```
public static int contarVivos(int[] vida) {  
  
int c = 0;  
  
for (int v : vida) if (v > 0) c++;  
  
return c;  
}
```

## 12.5. Ejercicio La Vuelta Al Mundo (Carrera de Carros):

### 12.5.1 Descripción del código

El programa comienza definiendo que habrá 10 autos y les asigna nombres y posiciones iniciales en cero. Cada turno, todos los autos avanzan una cantidad de posiciones al azar entre 1 y 10. Se muestra en pantalla cuánto avanzó cada auto y su posición total. La carrera continúa hasta que algún auto llega a la meta. Cuando esto ocurre, se detiene la carrera y se anuncia en pantalla cuál auto fue el primero en cruzar la meta, declarándose ganador.

### 12.5.2 Código

```
package ejercicios;

import java.util.Random;

public class LaVueltaAlMundo {

    public static void main(String[] args) {
        // Número de autos
        int n = 10;

        // Arreglo que representa la posición actual de los autos en la carrera
        int[] posiciones = new int[n]; // todas inician en 0

        // Arreglo que almacena los nombres de los autos
        String[] autos = {"Carro Rojo", "Carro Azul", "Carro Amarillo", "Carro Violeta", "Carro Café", "Carro Negro", "Carro Gris",
                          "Carro Lila", "Carro Naranja", "Carro Verde"};

        int meta = 50; // distancia a la meta
        Random rand = new Random(); // objeto Random para generar números aleatorios
        int turno = 1; // contador de turnos

        System.out.println("¡Comienza la carrera entre todos los autos!\n");

        // Bucle principal de la carrera: continúa hasta que algún auto cruce la meta
        boolean carreraActiva = true;
        while (carreraActiva) {
            System.out.println("Turno " + turno + ":");

            for (int i = 0; i < n; i++) {
                int avance = rand.nextInt(10) + 1; // genera un avance aleatorio de 1 a 10
                posiciones[i] += avance;

                if (posiciones[i] > meta) {
                    posiciones[i] = meta; // no sobrepasar la meta
                }

                // Mostrar avance de cada auto
                System.out.println(autos[i] + " avanza " + avance + " posiciones. Total: " + posiciones[i]);

                // Si algún auto llegó a la meta, terminar carrera
                if (posiciones[i] >= meta) {
                    carreraActiva = false;
                }
            }

            System.out.println(); // salto de línea entre turnos
            turno++; // incrementar el turno
        }

        // Determinar ganador (el primero que llegó a la meta)
        for (int i = 0; i < n; i++) {
            if (posiciones[i] >= meta) {
                System.out.println("El ganador es " + autos[i] + "!");
                break;
            }
        }
    }
}
```

### **12.5.3 Paso a paso del código**

#### **I. Preparar la carrera:**

Se define que habrá 10 autos, se les asignan nombres y se crean arreglos para guardar sus posiciones, que al inicio están en cero.

#### **II. Definir la meta y herramientas:**

Se establece la distancia de la meta (50 posiciones) y se crea un generador de números aleatorios para simular los avances de los autos.

#### **III. Iniciar la carrera:**

Se empieza un bucle que representa cada turno de la carrera y que se repite mientras ningún auto haya llegado a la meta.

#### **IV. Avance de los autos:**

- Cada auto avanza un número aleatorio de posiciones entre 1 y 10.
- Si un auto supera la meta, su posición se ajusta a la meta.
- Se imprime cuánto avanzó cada auto y su posición total.

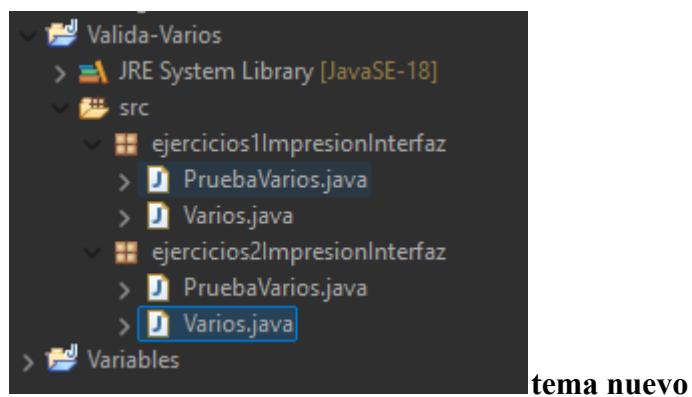
#### **V. Comprobar si hay ganador:**

- Si algún auto llega a la meta, la carrera termina.
- Se busca cuál auto fue el primero en cruzar la meta.

#### **VI. Mostrar ganador:**

Se imprime en pantalla el nombre del auto que ganó la carrera.

Cada vez que se reinicie ganara un carro distinto



tema nuevo

## 13. Matriz

### 13.1 ¿Qué es una matriz?

En general, una matriz es una forma de organizar datos en una estructura rectangular compuesta por filas y columnas. Cada posición dentro de esa tabla se llama elemento y se identifica mediante dos coordenadas; la fila en la que está y la columna que ocupa. Es como tener una cuadrícula o rejilla donde cada casilla guarda un valor, y la característica principal es que todos los elementos de la matriz pertenecen al mismo tipo de dato.

### 13.2 ¿Para qué sirve una matriz en Java?

En Java, una matriz sirve para almacenar y organizar múltiples valores del mismo tipo de dato en una estructura ordenada. Esto permite manejar grandes cantidades de información sin necesidad de declarar muchas variables. Facilita el acceso a los datos mediante índices, lo que hace posible recorrerlos, modificarlos y aplicar operaciones de forma sistemática, es muy útil cuando necesitas trabajar con colecciones de datos que tienen una relación estructural, como tablas, listas o cuadrículas.

- Por ejemplo, imagina que quieras almacenar las notas de 3 estudiantes de cuatro materias distintas, donde las columnas representan a las materias, las filas a los estudiantes y cada casilla son las notas obtenidas por los estudiantes.

### 13.3 Para evaluar su funcionamiento haremos el siguiente programa básico en java.

```
package ejemplomanual;

public class ejemplobasicomatriz {
    public static void main(String[] args) {
        /* aqui crearemos un arreglo bidimensional en lugar de un arreglo simple,
        lo que diferencia a ambos es que el arreglo simple solo puede almacenar datos
        de forma lineal osea en fila, el bidimensional almacena filas y columnas
        lo que nos permite crear una matriz. */
        double[][] notas = {
            { 2.0, 5.0, 3.4, 4.1 }, // Sysooooooooo
            { 3.2, 2.9, 4.4, 3.7 }, // Kevin Guerred
            { 3.0, 4.2, 3.3, 4.8 }, // Emanuel Sierra
        };
        // aca le daremos visualizacion a la matriz en la consola.

        System.out.println("Tabla de calificaciones");
        for(int i = 0; i < notas.length; i++) {

            System.out.println("Estudiante " + (i+1) + " :");
            for(int j = 0; j < notas[i].length; j++) {
                System.out.println(notas[i][j] + " ");
            }
        }
    }
}
```

### 13.4 Salida en consola:

```
Tabla de calificaciones
Estudiante 1 :
2.0
5.0
3.4
4.1
Estudiante 2 :
3.2
2.9
4.4
3.7
Estudiante 3 :
3.0
4.2
3.3
4.0
8.0
```

Pero obviamente esto no solo aplica en temas matemáticos o académicos, también se puede aplicar como si fuese un mapa en los videojuegos, por ejemplo, imagina que tu personaje se mueve por las casillas de la matriz, donde el 0 es el camino libre, el 1 es un enemigo y el 2 es una recompensa. El personaje empieza en una posición inicial y explora el tablero moviéndose por las casillas, enfrentándose a enemigos y recogiendo recompensas mientras evita obstáculos.

### 13.5 Miremos como funciona en relación a un videojuego:

```
package ejemplomanual;

import java.util.Scanner;
public class minijuego {

    public static void main(String[]args) {
        Scanner teclado = new Scanner(System.in);
        // aca crearemos nuestra matriz usando el arreglo bidimensional nombrado anteriormente
        // 0 = camino libre manito; 1 = enemigo en la juega; 2 = recompensa
        int[][] mapa = {
            {0 , 1 , 0},
            {0 , 2 , 1},
            {2 , 1 , 1}
        };
        // aca le damos imagen a la matriz
        System.out.println(" el mapa del juego tiene un tamaño de 3x3: \n");
        for(int i = 0; i < mapa.length; i++) {
            for(int j = 0; j < mapa.length; j++) {
                System.out.print(mapa[i][j] );
            }
            System.out.println();
        }
        // aca elige su posicion
        System.out.println("ingresa la fila (0-2): \n");
        int fila = teclado.nextInt();
        System.out.println("ingrese la columna (0-2): \n");
        int columna = teclado.nextInt();

        // posicion actual mano
        System.out.println("llego a:" + fila + columna + " y encontraste: " + posicion(mapa[fila][columna]));
    }

    // aca miramos en que posicion queda y si debe enfrentarse a un enemigo,
    // si hay un premio o no hay nada
    public static String posicion(int valor) {
        if(valor == 0)return "no hay nada \n";
        else if(valor == 1)return "en la juega que hay un enemigo \n";
        else if(valor == 2)return "mas debuenas encontro algo \n";
        return "que escribio no ve que es de 0 a 2 gueva \n";
    }
}
```

### 13.6 Salida en consola:

```
el mapa del juego tiene un tamaño de 3x3:  
010  
021  
211  
ingresa la fila (0-2):  
0  
ingrese la columna (0-2):  
1  
|llego a:01 y encontraste: en la juega que hay un enemigo
```

Ahora con lo aprendido anteriormente haremos un problema más complejo usando las matrices y otros temas vistos durante el curso haremos un programa en java el cual es un menú interactivo que permite practicar con matrices bidimensionales. El usuario puede crear una matriz ingresando datos manualmente o generarlos de forma aleatoria, mostrarla en pantalla, buscar elementos dentro de ella, invertirla como un espejo, eliminarla, ordenar sus valores de menor a mayor y realizar operaciones matemáticas básicas como suma, resta y multiplicación con otra matriz.

En pocas palabras, es un ejercicio que reúne las operaciones más comunes con matrices y sirve para que un principiante entienda cómo capturar datos, recorrerlos con ciclos, manipularlos y aplicar lógica paso a paso en programación.

pero antes de continuar con la creación de este programa tendremos que crear una clase auxiliar llamada VARIOS esta clase nos ayudará a interacción con el usuario cuando se utilizan cuadros de diálogo Swing (JOptionPane). El método leer entero nos permitirá pedir un número entero mediante un mensaje, validando si la entrada es correcta y el usuario no coloca un valor que no sea numérico. El otro método llamado Mensaje solo muestra un cuadro con el texto que se le pase. Por lo tanto, esta clase centraliza la captura y visualización de datos, facilitando que el programa principal trabaje con matrices sin preocuparse por la validación de entradas o la forma de mostrar mensajes.

```
package tareadePOO;

import javax.swing.JOptionPane;

public class Varios {

    public int leerEntero(String mensaje) {
        int num = 0;
        boolean valido = false;
        while (!valido) {
            try {
                String entrada = JOptionPane.showInputDialog(null, mensaje);
                if (entrada == null) {
                    JOptionPane.showMessageDialog(null, "Entrada cancelada. Se usará 0 por defecto.");
                    return 0;
                }
                num = Integer.parseInt(entrada);
                valido = true;
            } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(null, "Por favor, ingrese un número entero válido.");
            }
        }
        return num;
    }

    public void Mensaje(String mensaje) {
        JOptionPane.showMessageDialog(null, mensaje);
    }

}
```

Una vez creada la clase auxiliar Varios es importante que el programa que creemos y utilice este clase estén en el mismo package, después de tener todo listo ahora si haremos nuestro menú de matrices en java.

```

package tareadeP00;

public class tarea13denoviembre {

    static Varios v = new Varios();
    static int TAM = 5;
    static int a[][] = new int[TAM][TAM];
    static int fil = 0;
    static int col = 0;

    public static void main(String[] args) {
        int op;
        do {
            String menu = "Menu de matrices \n" +
                "1. Cargar \n" +
                "2. Listar \n" +
                "3. Buscar \n" +
                "4. Invertir \n" +
                "5. Eliminar \n" +
                "6. Aleatorio \n" +
                "7. Ordenar \n" +
                "8. Operaciones con matrices \n" +
                "0. Salir \n";
            op = v.leerEntero(menu);
            switch (op) {
                case 1: Cargar(); break;
                case 2: Listar(); break;
                case 3: Buscar(); break;
                case 4: Invertir(); break;
                case 5: Eliminar(); break;
                case 6: Aleatorio(); break;
                case 7: Ordenar(); break;
                case 8: OperacionesMatrices(); break;
            }
        } while (op != 0);
    }

    public static void Cargar() {
        Captura();
        for (int i = 0; i < fil; i++) {
            for (int j = 0; j < col; j++) {
                a[i][j] = v.leerEntero("Ingrese [" + i + "][" + j + "]");
            }
        }
    }

    public static void Captura() {
        fil = v.leerEntero("Ingrese cantidad de filas");
        col = v.leerEntero("Ingrese cantidad de columnas");
        a = new int[fil][col];
    }

    public static void Listar() {
        for (int i = 0; i < fil; i++) {
            for (int j = 0; j < col; j++) {
                System.out.print(a[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static void Buscar() {
        int buscado = v.leerEntero("Ingrese el valor a buscar");
        boolean encontrado = false;
        for (int i = 0; i < fil; i++) {
            for (int j = 0; j < col; j++) {
                if (a[i][j] == buscado) {
                    System.out.println("El valor " + buscado + " se encuentra en la posicion [" + i + "][" + j + "]");
                    encontrado = true;
                }
            }
        }
        if (!encontrado) {
            System.out.println("El valor " + buscado + " no se encuentra en la matriz");
        }
    }

    public static void Invertir() {
        for (int i = 0; i < fil; i++) {
            for (int j = 0; j < col; j++) {
                a[i][j] = a[j][i];
            }
        }
    }

    public static void Eliminar() {
        int fila = v.leerEntero("Ingrese la fila a eliminar");
        int columna = v.leerEntero("Ingrese la columna a eliminar");
        if (fila < 0 || fila > fil - 1 || columna < 0 || columna > col - 1) {
            System.out.println("Fila o columna invalida");
        } else {
            for (int i = fila; i < fil - 1; i++) {
                for (int j = columna; j < col; j++) {
                    a[i][j] = a[i + 1][j];
                }
            }
            fil--;
            col--;
        }
    }

    public static void Aleatorio() {
        for (int i = 0; i < fil; i++) {
            for (int j = 0; j < col; j++) {
                a[i][j] = (int) (Math.random() * 100);
            }
        }
    }

    public static void Ordenar() {
        for (int i = 0; i < fil; i++) {
            for (int j = 0; j < col; j++) {
                a[i][j] = a[i][j] * 2;
            }
        }
    }

    public static void OperacionesMatrices() {
        System.out.println("Operaciones con matrices");
        System.out.println("1. Sumar");
        System.out.println("2. Restar");
        System.out.println("3. Multiplicar");
        System.out.println("4. Dividir");
        int op = v.leerEntero("Seleccione una operacion");
        if (op == 1) {
            Matrices.sumar();
        } else if (op == 2) {
            Matrices.restar();
        } else if (op == 3) {
            Matrices.multiplicar();
        } else if (op == 4) {
            Matrices.dividir();
        }
    }

    public static void sumar() {
        Matrices m1 = new Matrices(fil, col);
        Matrices m2 = new Matrices(fil, col);
        Matrices resultado = new Matrices(fil, col);
        for (int i = 0; i < fil; i++) {
            for (int j = 0; j < col; j++) {
                m1.a[i][j] = v.leerEntero("Ingrese el valor para la posicion [" + i + "][" + j + "]");
                m2.a[i][j] = v.leerEntero("Ingrese el valor para la posicion [" + i + "][" + j + "]");
            }
        }
        resultado = m1.sumar(m2);
        resultado.listar();
    }

    public static void restar() {
        Matrices m1 = new Matrices(fil, col);
        Matrices m2 = new Matrices(fil, col);
        Matrices resultado = new Matrices(fil, col);
        for (int i = 0; i < fil; i++) {
            for (int j = 0; j < col; j++) {
                m1.a[i][j] = v.leerEntero("Ingrese el valor para la posicion [" + i + "][" + j + "]");
                m2.a[i][j] = v.leerEntero("Ingrese el valor para la posicion [" + i + "][" + j + "]");
            }
        }
        resultado = m1.restar(m2);
        resultado.listar();
    }

    public static void multiplicar() {
        Matrices m1 = new Matrices(fil, col);
        Matrices m2 = new Matrices(fil, col);
        Matrices resultado = new Matrices(fil, col);
        for (int i = 0; i < fil; i++) {
            for (int j = 0; j < col; j++) {
                m1.a[i][j] = v.leerEntero("Ingrese el valor para la posicion [" + i + "][" + j + "]");
                m2.a[i][j] = v.leerEntero("Ingrese el valor para la posicion [" + i + "][" + j + "]");
            }
        }
        resultado = m1.multiplicar(m2);
        resultado.listar();
    }

    public static void dividir() {
        Matrices m1 = new Matrices(fil, col);
        Matrices m2 = new Matrices(fil, col);
        Matrices resultado = new Matrices(fil, col);
        for (int i = 0; i < fil; i++) {
            for (int j = 0; j < col; j++) {
                m1.a[i][j] = v.leerEntero("Ingrese el valor para la posicion [" + i + "][" + j + "]");
                m2.a[i][j] = v.leerEntero("Ingrese el valor para la posicion [" + i + "][" + j + "]");
            }
        }
        resultado = m1.dividir(m2);
        resultado.listar();
    }
}

```

```

        }
    }

    public static void Captura() {
        while (true) {
            fil = v.leerEntero("Cuantas Filas ");
            if (fil >= 1 && fil <= TAM) break;
            else v.Mensaje("Debe estar entre 1 y " + TAM);
        }
        while (true) {
            col = v.leerEntero("Cuantas Columnas ");
            if (col >= 1 && col <= TAM) break;
            else v.Mensaje("Debe estar entre 1 y " + TAM);
        }
    }

    public static void Listar() {
        if (fil == 0 || col == 0)
            v.Mensaje("No hay elementos");
        else {
            for (int i = 0; i < fil; i++) {
                for (int j = 0; j < col; j++) {
                    System.out.printf("%4d", a[i][j]);
                }
                System.out.println();
            }
        }
    }

    public static void Aleatorio() {
        Captura();
        for (int i = 0; i < fil; i++)
            for (int j = 0; j < col; j++)
                a[i][j] = (int) (Math.random() * 50) + 1;
        v.Mensaje("Matriz generada aleatoriamente");
    }

    public static void Buscar() {
        if (fil == 0 || col == 0)
            v.Mensaje("No hay elementos");
        else {
            ...
        }
    }
}

```

```

        int n = v.leerEntero("Elemento a buscar ");
        boolean encontrado = false;
        int i = 0, j = 0;
        for (i = 0; i < fil; i++) {
            for (j = 0; j < col; j++) {
                if (a[i][j] == n) {
                    encontrado = true;
                    break;
                }
            }
            if (encontrado) break;
        }
        if (encontrado)
            v.Mensaje("Elemento encontrado en [" + i + "][" + j + "]");
        else
            v.Mensaje("No se encontró el elemento");
    }

    public static void Invertir() {
        if (fil == 0 || col == 0) {
            v.Mensaje("No hay elementos");
            return;
        }
        for (int i = 0; i < fil; i++) {
            for (int j = 0; j < col / 2; j++) {
                int temp = a[i][j];
                a[i][j] = a[i][col - 1 - j];
                a[i][col - 1 - j] = temp;
            }
        }
        v.Mensaje("Matriz invertida horizontalmente");
    }

    public static void Eliminar() {
        fil = 0;
        col = 0;
        v.Mensaje("Matriz eliminada");
    }

    public static void Ordenar() {
        if (fil == 0 || col == 0) {

```

```

        v.Mensaje("No hay elementos");
        return;
    }
    int[] temp = new int[fil * col];
    int k = 0;
    for (int i = 0; i < fil; i++)
        for (int j = 0; j < col; j++)
            temp[k++] = a[i][j];

    // Ordenamiento burbuja
    for (int i = 0; i < temp.length - 1; i++) {
        for (int j = 0; j < temp.length - 1 - i; j++) {
            if (temp[j] > temp[j + 1]) {
                int aux = temp[j];
                temp[j] = temp[j + 1];
                temp[j + 1] = aux;
            }
        }
    }

    k = 0;
    for (int i = 0; i < fil; i++)
        for (int j = 0; j < col; j++)
            a[i][j] = temp[k++];
    v.Mensaje("Matriz ordenada");
}

public static void OperacionesMatrices() {
    if (fil == 0 || col == 0) {
        v.Mensaje("Primero debe cargar la matriz principal.");
        return;
    }
    int[][] b = new int[TAM][TAM];
    int[][] resultado = new int[TAM][TAM];

    v.Mensaje("Captura de la segunda matriz:");
    int fil2 = v.leerEntero("Filas de la segunda matriz:");
    int col2 = v.leerEntero("Columnas de la segunda matriz:");

    if (fil2 != fil || col2 != col) {
        v.Mensaje("Las matrices deben tener el mismo tamaño para suma y resta.");
        return;
    }
}

```

## 14. Introducción al modo grafico en Java

### 14.1. ¿Qué es una interfaz gráfica de usuario?

Una interfaz gráfica de usuario es la parte visual de un programa con la que el usuario interactúa mediante elementos como ventanas, botones, cuadros de texto, menús, etc.

En lugar de escribir comandos en una consola, el usuario hace clic, escribe en campos de texto, selecciona opciones, arrastra elementos, etc.

#### Características:

- Son más intuitivas para el usuario que una interfaz de texto.
- Están compuestas por componentes gráficos (ventanas, botones, textos, listas...).
- Funcionan mediante un modelo de programación dirigida por eventos: el programa “reacciona” a lo que hace el usuario (clics, teclas, movimientos del ratón).

### 14.2. Programación dirigida por eventos

En una aplicación gráfica, el flujo del programa no lo controla un main secuencial solamente, sino que:

- El programa crea la ventana y los componentes.
- Se inicia un bucle interno de eventos que gestiona el sistema gráfico.
- Cada vez que ocurre algo (clic, tecla, cerrar ventana), se genera un evento.
- Si el programador registró un manejador de eventos (listener), ese código se ejecuta en respuesta.

#### Es decir, ya no hacemos:

##### Primer línea de código à segunda línea de código à tercera línea de código

- Sino que ahora será así:
- Si el usuario hace click aquí à hacer esto
- Si pasa el cursor aquí à activar este evento

- Si el usuario escribe aquí à hacer esto otro
- Si cierra la venta à terminar el programa

### **14.3. Librerías gráficas en Java: AWT y Swing**

Para trabajar con interfaces graficas, Java nos proporciona diferentes librerías para trabajar, pero las mas comunes e importantes son:

#### **14.3.1 AWT**

- Es la primera librería gráfica de Java.
- Usa componentes "pesados" (heavyweight): dependen directamente del sistema operativo.
- Los botones, ventanas, etc., son “envolturas” de controles nativos.

#### **14.3.2 Swing (la principal que usaremos)**

- Es una librería más moderna construida encima de AWT.
- Usa componentes "ligeros" (lightweight): se dibujan en Java, no dependen tanto del sistema operativo.
- Es más flexible, personalizable y completa.
- Tiene componentes más avanzados: tablas (JTable), listas (JList), paneles con pestañas, etc.

### **14.4. ¿Qué es AWT?**

**AWT es un conjunto de clases que permiten:**

- Crear ventanas y diálogos.
- Dibujar formas (líneas, rectángulos, etc.).
- Gestionar componentes gráficos básicos: botones, etiquetas, campos de texto.
- Manejar eventos (clics de ratón, teclado, etc.)
- Se encuentra en el paquete java.awt y java.awt.event.

## 14.5. Componentes principales de AWT

Algunos componentes típicos de AWT son:

- Frame: ventana principal con barra de título, botones de cerrar, minimizar, etc.
- Label: etiqueta de texto no editable.
- TextField: campo de texto de una sola línea.
- TextArea: área de texto de varias líneas.
- Button: botón sobre el que el usuario puede hacer clic.
- Checkbox, CheckboxGroup: casillas de selección.
- Choice: desplegable de opciones.

## 14.5. Swing

### 14.5.1. ¿Qué es Swing?

Swing es una biblioteca gráfica de Java que amplía y mejora a AWT. Se encuentra en el paquete javax.swing y proporciona:

- Componentes con la letra J delante: JFrame, JButton, JLabel, JTextField, etc.
- Mayor independencia del sistema operativo (se dibujan en Java).
- Soporte para temas visuales (look and feel).
- **Componentes avanzados:** tablas (JTable), listas (JList), paneles con pestañas (JTabbedPane), etc.

### 14.5.2. Ventajas de Swing frente a AWT

- **Más componentes:** Swing tiene muchos más controles listos para usar.
- **Personalización:** puedes cambiar colores, fuentes, bordes, íconos, etc. con facilidad.
- **Portabilidad visual:** la apariencia es más consistente entre distintos sistemas operativos.
- Modelos de datos: algunos componentes (como tablas y listas) separan los datos de la vista.

### 14.5.3 Componentes básicos de Swing

- Los más usados, y que puedes explicar en tu manual, son:
- **JFrame:** ventana principal de la aplicación.
- **JPanel:** panel contenedor, sirve para agrupar otros componentes.

- **JLabel**: etiqueta de texto no editable.
- **JTextField**: campo de texto de una sola línea.
- **JTextArea**: campo de texto de varias líneas.
- **JButton**: botón.
- **JCheckBox**: casilla de verificación.
- **JRadioButton**: botón de opción (normalmente agrupados con ButtonGroup).
- JMenuBar, JMenu, JMenuItem: barra de menús, menús y opciones.

Ahora bien, la librería Swing tiene muchas clases y en la mayoría de programas se usan mas de una de estas, por lo que para no poner una por una de estas clases podemos importar lo siguiente:

#### **14.6. import javax.swing.\*;**

Este paquete contiene los elementos fundamentales para construir interfaces gráficas en Java, tales como ventanas (JFrame), botones ( JButton), etiquetas ( JLabel), cuadros de diálogo ( JOptionPane), paneles ( JPanel) y otros muchos componentes visuales.

No obstante, hay que tener en cuenta que este paquete no contiene clases de otros paquetes como java.awt, no crea objetos por sí misma y no carga librerías externas, solo clases del propio JDK.

Para la biblioteca java.awt también podemos importar de forma resumida la mayoría de clases con:

#### **14.7. java.awt.\*;**

**A continuación un ejemplo sencillo usando la biblioteca Swing:**

```
// Importamos la clase JButton desde la librería Swing.

// JButton permite crear botones gráficos que el usuario puede pulsar.

import javax.swing.JButton;

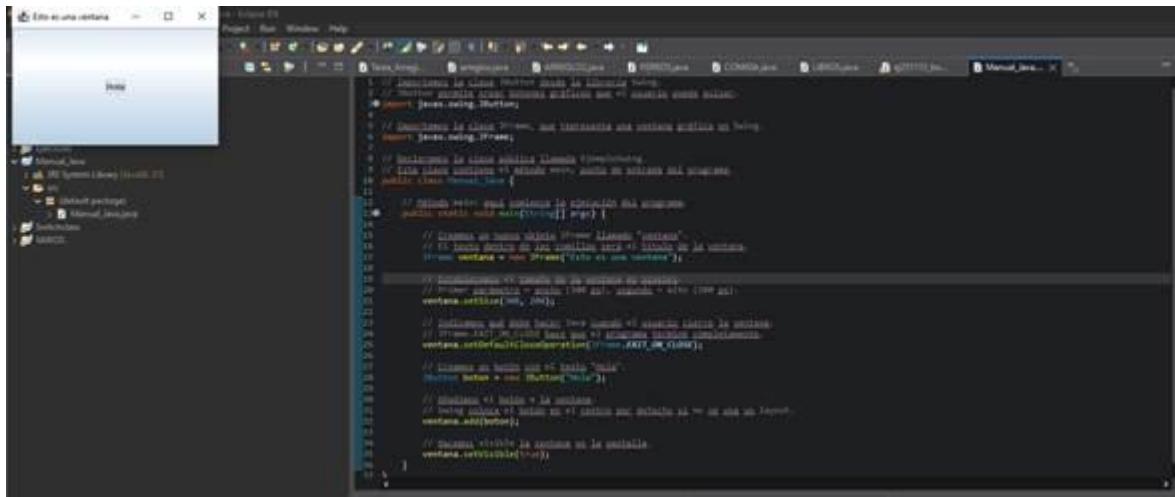
// Importamos la clase JFrame, que representa una ventana gráfica en Swing.

import javax.swing.JFrame;
```

```
// Declaramos la clase pública llamada EjemploSwing.  
  
// Esta clase contiene el método main, punto de entrada del programa.  
  
public class Manual_Java {  
  
    // Método main: aquí comienza la ejecución del programa.  
  
    public static void main(String[] args) {  
  
        // Creamos un nuevo objeto JFrame llamado "ventana".  
  
        // El texto dentro de las comillas será el título de la ventana.  
  
        JFrame ventana = new JFrame("Esto es una ventana");  
  
        // Establecemos el tamaño de la ventana en píxeles.  
  
        // Primer parámetro = ancho (300 px), segundo = alto (200 px).  
  
        ventana.setSize(300, 200);  
  
        // Indicamos qué debe hacer Java cuando el usuario cierre la ventana.  
  
        // JFrame.EXIT_ON_CLOSE hace que el programa termine completamente.  
  
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        // Creamos un botón con el texto "Hola".  
  
        JButton boton = new JButton("Hola");  
  
        // Añadimos el botón a la ventana.  
  
        // Swing coloca el botón en el centro por defecto si no se usa un layout.  
  
        ventana.add(boton);
```

```
// Hacemos visible la ventana en la pantalla.  
  
ventana.setVisible(true);  
  
}  
  
}
```

Nos debería de soltar una ventana como la siguiente:



En este código hay 2 elementos que hay que tener muy en cuenta para la visualización y el correcto funcionamiento de la ventana respectivamente:

## **14.8. Dimensiones y ubicación de una ventana en Java Swing**

En Java, las interfaces gráficas se crean utilizando la clase JFrame, que representa una ventana dentro de la aplicación. Para controlar el tamaño y la posición de esta ventana en la pantalla, se emplean métodos específicos que permiten al programador definir su apariencia y ubicación de forma precisa.

### **4.8.1. Definición del tamaño de una ventana**

El tamaño de una ventana se expresa en píxeles, la unidad mínima visible en una pantalla. En Swing, el tamaño se establece mediante el método:

**setSize(ancho, alto);**

- **ancho**: cantidad de píxeles en sentido horizontal.
- **alto**: cantidad de píxeles en sentido vertical.

**Por ejemplo:**

**ventana.setSize(300, 200);**

crea una ventana de 300 píxeles de ancho y 200 píxeles de alto.

El uso de píxeles permite definir de manera precisa la dimensión de los elementos gráficos, independientemente de la resolución del monitor utilizado.

## **14.9. Ubicación predeterminada de la ventana**

Cuando se define únicamente el tamaño de la ventana, sin especificar su posición, esta aparece en una ubicación automática, determinada por el sistema operativo o por el propio gestor de ventanas de Swing. Generalmente, esta posición se encuentra en la parte superior de la pantalla, ligeramente desplazada desde la esquina izquierda.

Es importante tener en cuenta que, si no se especifica la localización, la ventana no aparecerá centrada, sino en un punto por defecto.

#### **14.10. Definición manual de la posición de la ventana**

Si se desea controlar exactamente dónde aparecerá la ventana, se puede utilizar el método:

##### **14.10.1 setLocation(x, y);**

**donde:**

- **x** es la distancia en píxeles desde el borde izquierdo de la pantalla hasta la ventana.
- **y** es la distancia desde el borde superior hasta la ventana.

**Ejemplo:**

```
ventana.setLocation(200, 100);
```

**Con estos valores, la esquina superior izquierda de la ventana se situará 200 píxeles hacia la derecha y 100 píxeles hacia abajo respecto al punto (0,0) de la pantalla.**

Es importante tener en cuenta las dimensiones en píxeles de la pantalla que se mostrará la ventana, por lo general la mayoría de ordenadores poseen un resolución de 1920x1080 píxeles.

#### **14.11 Centrando una ventana en la pantalla**

Para que la ventana aparezca directamente en el centro de la pantalla, Java proporciona un método especializado:

```
setLocationRelativeTo(null);
```

Cuando se usa este método, el sistema calcula automáticamente la posición central del monitor y coloca la ventana en ese punto, sin necesidad de indicar coordenadas manuales. Esta técnica es especialmente recomendada en aplicaciones que buscan una presentación más profesional.

#### **14.12. Definir simultáneamente la posición y el tamaño de una ventana.**

Además de los métodos `setSize()` y  `setLocation()`, utilizados para definir el tamaño y la posición de una ventana de forma separada, Java proporciona un método que permite establecer ambos valores en una sola línea de código. Este método es:

- `setBounds(x, y, ancho, alto);`
- Los parámetros representan:
- **x**: distancia en píxeles desde el borde izquierdo de la pantalla.
- **y**: distancia en píxeles desde el borde superior.
- **ancho**: anchura de la ventana.
- **alto**: altura de la ventana.
- Por ejemplo:
- `ventana.setBounds(300, 100, 500, 400);`

##### **Con esta instrucción:**

La ventana se sitúa a 300 píxeles a la derecha desde el borde izquierdo de la pantalla, a 100 píxeles hacia abajo desde el borde superior.

Tendrá un tamaño de 500 píxeles de ancho y 400 píxeles de alto.

El método **setBounds()** resulta especialmente útil cuando se desea configurar la ventana de manera directa y compacta, evitando el uso separado de `setSize()` y `setLocation()`.

### 14.13. Uso de setDefaultCloseOperation

Para un mejor funcionamiento del programa, en este caso la ventana, siempre debemos indicar qué ocurre al cerrar la ventana:

#### 14.13.1 setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

- `EXIT_ON_CLOSE` hace que el programa termine cuando se cierra la ventana.
- Otras opciones (`HIDE_ON_CLOSE`, `DISPOSE_ON_CLOSE`) se usan según las necesidades.

## 14.14. Eventos

### 14.14.1. ¿Qué es un evento?

Un evento es una acción que ocurre en la interfaz gráfica, por ejemplo:

- El usuario hace clic en un botón.
- Cierra la ventana.
- Escribe texto en un campo.
- Cambia la selección de una lista, etc.

En Java, cada tipo de evento se representa con una clase, como `ActionEvent`, `MouseEvent`, `KeyEvent`, etc.

### 14.14.2. ¿Qué es un listener o manejador de eventos?

Un listener (escuchador o manejador de eventos) es un objeto que “escucha” un tipo de evento y ejecuta un código cuando ese evento ocurre.

En Swing/AWT:

- Se define una interfaz como `ActionListener`, `MouseListener`, `KeyListener`, etc.
- El programador crea una clase que implementa esa interfaz y define qué hacer en los métodos obligatorios.
- Después se registra ese listener sobre un componente mediante métodos como `addActionListener(...)`, `addMouseListener(...)`, etc.

A continuación un ejemplo de cómo usar los listeners:

```
// Importamos las clases necesarias de Swing

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

// Importamos ActionListener para manejar eventos

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

// La clase VentanaEjemplo es una ventana (JFrame)
// y además implementa ActionListener para manejar eventos

public class VentanaEjemplo extends JFrame implements ActionListener {

    // Declaramos dos botones

    JButton boton1;
    JButton boton2;

    // Método main: donde inicia el programa

    public static void main(String[] args) {
        // Creamos una ventana de esta misma clase
        VentanaEjemplo ventana = new VentanaEjemplo();

        // Hacemos visible la ventana
        ventana.setVisible(true);
    }
}
```

```
// Constructor: aquí configuramos y montamos la ventana

public VentanaEjemplo0 {

    // Título de la ventana
    setTitle("Ejemplo de Listeners");

    // Tamaño de la ventana
    setSize(300, 200);

    // Hace que el programa se cierre al cerrar la ventana
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // No usamos layouts automáticos para simplificar
    setLayout(null);

    // BOTÓN 1: usa THIS como listener

    // Creamos el botón 1
    boton1 = new JButton("Botón 1");

    // Posición dentro de la ventana
    boton1.setBounds(50, 40, 100, 30);

    // Lo añadimos a la ventana
    add(boton1);

    // Registraremos THIS como listener (es decir, la clase actual)
```

```

boton1.addActionListener(this);

// BOTÓN 2: usa un listener anónimo

// Creamos el botón 2

boton2 = new JButton("Botón 2");

boton2.setBounds(160, 40, 100, 30);

add(boton2);

// Le agregamos un listener directamente, sin crear métodos aparte

boton2.addActionListener(
    new ActionListener() { // Esto crea un "listener anónimo"

        public void actionPerformed(ActionEvent e) {

            // Código que se ejecuta cuando se pulsa el botón 2

            JOptionPane.showMessageDialog(null, "Pulsaste el botón 2");
        }
    }
);

}

// Este método se ejecuta SOLO para los botones que usan "this"

@Override

public void actionPerformed(ActionEvent e) {

    // Preguntamos qué botón generó el evento

    if (e.getSource() == boton1) {
}

```

```
JOptionPane.showMessageDialog(null, "Pulsaste el botón 1");  
}  
}  
}
```

#### 14.15. Implementar ActionListener a la clase

La clase declara:

- public class VentanaEjemplo extends JFrame implements ActionListener
- Esto significa que la propia ventana actuará como “escuchador” (listener).

Cuando registramos el botón:

- boton1.addActionListener(this);
- La ventana responderá usando su método:
- public void actionPerformed(ActionEvent e)
- y allí distinguimos qué botón fue pulsado.

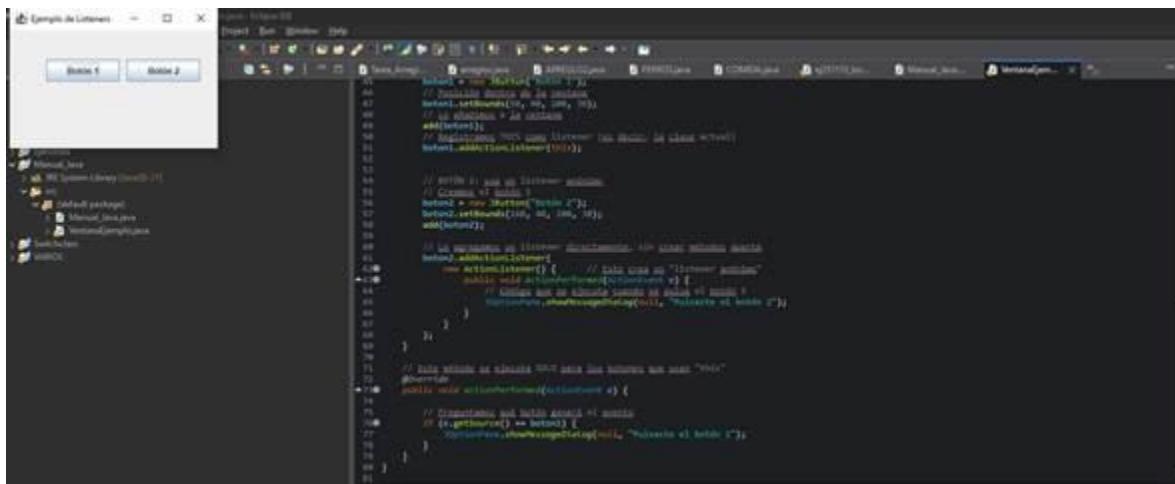
##### 14.15.1 Listener anónimo

Aquí no usamos this, sino:

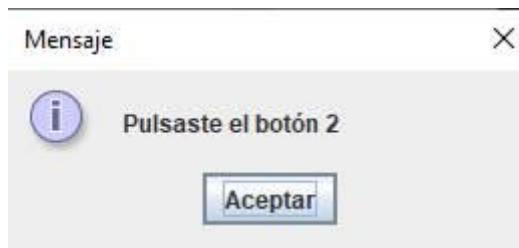
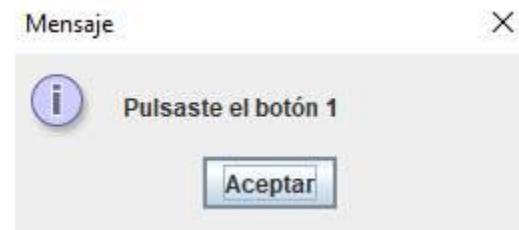
```
boton2.addActionListener(new ActionListener() {  
  
    public void actionPerformed(ActionEvent e) {  
  
        // Código específico del botón 2  
  
    }  
  
});
```

Esta forma crea un “listener en línea”, útil cuando el código del evento es corto y no necesitamos un método general.

El ejemplo nos debería de soltar una ventana con 2 botones, como la siguiente:



Al darle a uno de los 2 botones nos dirá que estamos pulsando ese botón en específico:



## 14.16. Colores en Interfaces Gráficas con Java Swing

En Java Swing es posible personalizar la apariencia de los componentes gráficos utilizando colores. Esto permite crear interfaces más atractivas y diferenciadas visualmente. Los colores se gestionan mediante la clase `java.awt.Color`, la cual proporciona una gran variedad de tonos predefinidos, así como la posibilidad de crear colores personalizados mediante valores RGB.

### 14.16.1 Colorear el fondo de un componente

Para establecer el color de fondo de un componente se utiliza el método:

- setBackground(Color.colorElegido);
- Ejemplo:
- panel.setBackground(Color.BLUE);
- Este código asigna el color azul al fondo de un JPanel.

#### 14.16.2. Colorear el texto de un componente

Para definir el color del texto (por ejemplo, el texto de un botón o una etiqueta) se utiliza el método:

- setForeground(Color.colorElegido);
- Ejemplo:
- etiqueta.setForeground(Color.RED);
- En este caso, el texto de la etiqueta aparecerá en color rojo.

#### 14.16.3. Colores predefinidos en Java

La clase **Color** incluye numerosos colores listos para usar. Algunos de los más comunes son:

- **Color.BLACK**
- **Color.WHITE**
- **Color.RED**
- **Color.BLUE**
- **Color.GREEN**
- **Color.YELLOW**
- **Color.ORANGE**
- **Color.GRAY**
- **Color.LIGHT\_GRAY**
- **Color.DARK\_GRAY**
- **Color.CYAN**

- **Color.MAGENTA**
- **Color.PINK**

Estos valores son útiles para asignar colores rápidamente sin necesidad de cálculos adicionales.

#### 14.16.4 Creación de colores personalizados (RGB)

Además de los colores predefinidos, Java permite crear cualquier color utilizando valores RGB (Red, Green, Blue). Cada canal se expresa con un valor entre 0 y 255:

```
Color miColor = new Color(120, 45, 200);
```

En este ejemplo:

- **120 → cantidad de rojo**
- **45 → cantidad de verde**
- **200 → cantidad de azul**

A continuación un ejemplo del uso de colores en interfaces:

```
import javax.swing.*;  
  
import java.awt.Color;  
  
  
public class VentanaColores extends JFrame {  
  
  
    public VentanaColores() {  
  
        // Título y tamaño  
  
        setTitle("Ejemplo de Colores en Swing");  
  
        setSize(300, 200);  
  
  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
}
```

```

setLayout(null);

// Crear un panel y darle un color de fondo

JPanel panel = new JPanel();

panel.setBounds(0, 0, 300, 200);

panel.setBackground(Color.GREEN); // Fondo Verde

add(panel);

// Crear una etiqueta con texto de color

JLabel etiqueta = new JLabel("Texto en rojo");

etiqueta.setForeground(Color.RED); // Color del texto

panel.add(etiqueta);

// Crear un botón con colores personalizados

 JButton boton = new JButton("Botón azul");

boton.setBackground(new Color(0, 100, 255)); // Azul personalizado

boton.setForeground(Color.WHITE); // Texto blanco

panel.add(boton);

}

public static void main(String[] args) {

new VentanaColores().setVisible(true);

}

```

El anterior código nos suelta una ventana como la siguiente:



## 14.17 Cómo cambiar el tipo de letra, el estilo y el tamaño

En Java Swing, la apariencia del texto (fuente, tamaño, estilo) se controla mediante la clase:

**java.awt.Font**

Esta clase permite especificar tres cosas:

1. El nombre de la fuente (Arial, Times New Roman, Verdana, etc.)
2. El estilo de la fuente
  - **Font.PLAIN** à normal
  - **Font.BOLD** à negrita
  - **Font.ITALIC** à cursiva
3. El tamaño de la letra (en puntos)

La formula general es:

```
miComponente.setFont(new Font("NombreFuente", estilo, tamaño));
```

**Ejemplo:**

```
etiqueta.setFont(new Font("Arial", Font.BOLD, 24));
```

**Los componentes que muestran texto y permiten modificar la fuente son los siguientes:**

- **JLabel**
- **JButton**
- **JTextField**
- **JTextArea**
- **JCheckBox**
- **JRadioButton**
- **JMenu y JMenuItem**

#### **14.18. Ejemplo gráfico**

Ahora que comprendemos los conceptos de como usar la interfaz gráfica en Java, podemos realizar un ejemplo más complejo que abarque la mayoría de temas tratados, para ello buscaremos crear un código que se asemeje al menú de un videojuego. A continuación el código:

```
import javax.swing.*;      // Importamos Swing  
import java.awt.*;        // Importamos Color y Font  
import java.awt.event.*;   // Importamos ActionListener  
  
  
public class JuegoSimple extends JFrame implements ActionListener {  
  
  
    JButton botonJugar; // Listener con "this"  
    JButton botonSalir; // Listener anónimo  
    JLabel titulo;
```

```
public static void main(String[] args) {  
  
    // Creamos la ventana  
  
    JuegoSimple ventana = new JuegoSimple();  
  
    // Definimos tamaño y posición en una sola línea  
  
    ventana.setBounds(300, 100, 400, 250);  
  
    // Centramos la ventana en la pantalla  
  
    ventana.setLocationRelativeTo(null);  
  
    // Hacemos visible la ventana  
  
    ventana.setVisible(true);  
}  
  
public JuegoSimple() {  
  
    // Título de la ventana  
  
    setTitle("Mini Videojuego");  
  
    // Cerrar la aplicación al cerrar la ventana  
  
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
    // Elegimos layout manual  
  
    setLayout(null);
```

```

// Fondo negro (aspecto de videojuego)

getContentPane().setBackground(Color.BLACK);

// Etiqueta de título

titulo = new JLabel("MENU DEL JUEGO");
titulo.setForeground(Color.GREEN);
titulo.setFont(new Font("Arial", Font.BOLD, 22));
titulo.setBounds(100, 20, 250, 30);
add(titulo);

// Botón Jugar (usa THIS como listener)

botonJugar = new JButton("Jugar");
botonJugar.setBounds(50, 100, 120, 40);
botonJugar.addActionListener(this);
add(botonJugar);

// Botón Salir (usa listener anónimo)

botonSalir = new JButton("Salir");
botonSalir.setBounds(220, 100, 120, 40);
botonSalir.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null, "Cerrando el juego...");
        System.exit(0);
    }
});

```

```

});  

add(botonSalir);  

}  

// Listener principal (para el botón Jugar)  

@Override  

public void actionPerformed(ActionEvent e) {  

    JOptionPane.showMessageDialog(this, "¡Comienza el juego!");  

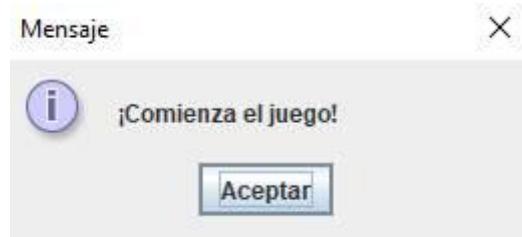
}
}

```

Este código nos debe de soltar el siguiente menú:



Al darle al botón “jugar” soltará el siguiente mensaje:



**Y al darle al botón “salir” nos soltará el siguiente mensaje para proceder a cerrarse la ventana:**



Esto es solo una muy pequeña muestra de lo que podemos lograr con la interfaz grafica en Java, presenta muchas más posibilidades, el único limite es tu capacidad como programador para sacarle el máximo jugo a esta función.

## 15. Práctica aplicada: ejercicios 22 – 52

### 15.1 Ejercicios del libro Lógica de Programación de Efraín Oviedo:

#### 22. Cálculo del salario mensual de un empleado según horas trabajadas.

Elaborar un algoritmo que entre el nombre de un empleado, su salario básico por hora y el número de horas trabajadas en el mes; escriba su nombre y salario mensual si éste es mayor de \$450.000, de lo contrario escriba sólo el nombre.

```
1 package taller1;
2 import java.util.Scanner;
3 public class ejercicio22 {
4     public static void main(String[]args) {
5
6         System.out.println("por favor ingrese su nombre");
7         Scanner teclado = new Scanner(System.in);
8         String nombre = teclado.nextLine();
9         System.out.println("tu nombre es: " + nombre);
10
11        System.out.println("por favor ingrese su salario por hora");
12        int salario;
13        salario = teclado.nextInt();
14        System.out.println("tu salario por hora es de:" + salario);
15
16        System.out.println("por favor ingrese el numero de horas trabajadas en el mes");
17        int horas;
18        horas = teclado.nextInt();
19        System.out.println("las horas trabajadas son:" + horas);
20
21        int salariomensual;
22        salariomensual = salario * horas;
23
24        if(salariomensual > 450000) {
25            System.out.println("tu nombre es:" + nombre);
26            System.out.println("tu salario mensual es de:" + salariomensual);
27
28        }else {
29            System.out.println("tu nombre es:" + nombre);
30        }
31    }
32 }
33 }
```

```
por favor ingrese su nombre
carlos
tu nombre es: carlos
por favor ingrese su salario por hora
40000
tu salario por hora es de:40000
por favor ingrese el numero de horas trabajadas en el mes
120
las horas trabajadas son:120
tu nombre es:carlos
tu salario mensual es de:4800000
```

### 23. Resolución de una ecuación de segundo grado.

Dados los valores A, B y C que son los parámetros de una ecuación de segundo grado, elaborar un algoritmo para hallar las posibles soluciones de dicha ecuación.

```
1 package taller1;
2 import java.util.Scanner;
3 public class ejercicio23 {
4     public static void main(String[]args) {
5
6         Scanner teclado = new Scanner(System.in);
7         int a, b, c;
8         System.out.println("ingrese el valor de a");
9         a = teclado.nextInt();
10        System.out.println("ingrese el valor de b");
11        b = teclado.nextInt();
12        System.out.println("ingrese el valor de c");
13        c = teclado.nextInt();
14
15        double respuesta;
16        respuesta = b * b - 4 * a * c;
17
18        if(respuesta < 0) {
19            System.out.println("no hay soluciones reales");
20        }else {
21
22            double r1 = (-b + Math.sqrt(respuesta))/(2*a);
23            double r2 = (-b - Math.sqrt(respuesta))/(2*a);
24            System.out.println("raices:" + r1 + "," + r2);
25        }
26
27
28    }
29
30 }
```

```
ingrese el valor de a  
1  
ingrese el valor de b  
-5  
ingrese el valor de c  
6  
raices:3.0,2.0
```

#### 24. Determinar la esfera de mayor peso entre tres objetos.

Se tienen tres esferas (A,B,C) de diferente peso, elaborar un algoritmo que determine cuál es la esfera de mayor peso.

```
Ingrese el peso de la primera esfera  
4  
ingrese el valor de la segunda esfera  
3  
ingrese el valor de la tercera esfera  
2  
la esfera de mayor peso es la esfera1 con una masa de:4.0
```

```
1 package taller1;  
2 import java.util.Scanner;  
3 public class ejercicio24 {  
4     public static void main(String[]args) {  
5         Scanner teclado = new Scanner(System.in);  
6         System.out.println("Ingrese el peso de la primera esfera");  
7         float esfera1 = teclado.nextFloat();  
8         System.out.println("ingrese el valor de la segunda esfera");  
9         float esfera2 = teclado.nextFloat();  
10        System.out.println("ingrese el valor de la tercera esfera");  
11        float esfera3 = teclado.nextFloat();  
12        if(esfera1 > esfera2 && esfera1 > esfera3) {  
13            System.out.println("la esfera de mayor peso es la esfera1 con una masa de:" + esfera1);  
14        } else if(esfera2 > esfera1 && esfera2 > esfera3) {  
15            System.out.println("la esfera de mayor peso es la esfera2 con una masa de:" + esfera2);  
16        } else {  
17            System.out.println("la esfera de mayor peso es la esfera3 con una masa de:" + esfera3);  
18        }  
19    }  
20}
```

```
Ingrese el peso de la primera esfera
4
ingrese el valor de la segunda esfera
3
ingrese el valor de la tercera esfera
2
la esfera de mayor peso es la esfera1 con una masa de:4.0
```

## 25. Identificar el número mayor entre cuatro valores.

Hacer un algoritmo que determine cuál es el mayor en un grupo de cuatro datos diferentes.

```
1 package taller1;
2 import java.util.Scanner;
3 public class ejercicio25 {
4     public static void main(String[]args) {
5         Scanner teclado = new Scanner(System.in);
6
7         System.out.println("ingrese 4 numeros enteros");
8         int a = teclado.nextInt();
9         int b = teclado.nextInt();
10        int c = teclado.nextInt();
11        int d = teclado.nextInt();
12
13        if(a > b && a > c && a > d) {
14            System.out.println("el dato mayor es:" + a);
15
16        }else if(b > a && b > c && b > d) {
17            System.out.println("el dato mayor es:" + b);
18
19        }else if(c > a && c > b && c > d) {
20            System.out.println("el dato mayor es:" + c);
21
22        }else {
23            System.out.println("el dato mayor es:" + d);
24
25        }
26
27
28    }
29
30 }
```

```
ingrese 4 numeros enteros
1
2
3
4
el dato mayor es:4
```

## 26. Sumar el valor menor y mayor entre cuatro datos.

Hacer un algoritmo que determine la suma del valor menor y mayor en un grupo de 4 datos.

```
1 package taller1;
2 import java.util.Scanner;
3 public class ejercicio26 {
4     public static void main(String[]args) {
5
6         Scanner teclado = new Scanner(System.in);
7
8         int[] datos = new int[4];
9
10        for (int i = 0; i < 4; i++) {
11            System.out.print("Ingresa el número " + (i + 1) + ": ");
12            datos[i] = teclado.nextInt();
13        }
14
15        int menor = datos[0];
16
17        int mayor = datos[0];
18        for (int i = 1; i < 4; i++) {
19            if (datos[i] < menor) {
20                menor = datos[i];
21            }
22            if (datos[i] > mayor) {
23                mayor = datos[i];
24            }
25        }
26
27
28        int suma = menor + mayor;
29
30
31        System.out.println("Menor: " + menor);
32        System.out.println("Mayor: " + mayor);
33        System.out.println("Suma del menor y mayor: " + suma);
34    }
35
36
37 }
```

```
Ingresá el número 1: 3
Ingresá el número 2: 4
Ingresá el número 3: 5
Ingresá el número 4: 5
Menor: 3
Mayor: 5
Suma del menor y mayor: 8
```

## 27. Conversión de un número octal de cinco dígitos a decimal.

Elaborar un algoritmo que determine el valor equivalente en el sistema numérico decimal, de un número de cinco dígitos octales.

```
1 package taller1;
2 import java.util.Scanner;
3
4 public class ejercicio27 {
5     public static void main(String[] args) {
6         Scanner scanner = new Scanner(System.in);
7
8         System.out.print("Ingresá un número octal de 5 dígitos: ");
9         int numeroOctal = scanner.nextInt();
10
11         int decimal = 0;
12         int potencia = 0;
13
14         while (numeroOctal > 0) {
15             int digito = numeroOctal % 10;
16             decimal += digito * Math.pow(8, potencia);
17             potencia++;
18             numeroOctal /= 10;
19         }
20
21         System.out.println("El equivalente decimal es: " + decimal);
22     }
23
24
25 }
```

```
Ingresá el número 1: 2
Ingresá el número 2: 3
Ingresá el número 3: 4
Ingresá el número 4: 2
Menor: 2
Mayor: 4
Suma del menor y mayor: 6
```

## 28. Clasificación y cálculo del área de un triángulo según sus lados.

Dados tres valores positivos determinar si éstos no forman triángulo o si forman triángulo, decir si éste es: equilátero, isósceles o escaleno y obtener el área del triángulo.

```

1 package taller1;
2 import java.util.Scanner;
3
4 public class ejercicio28 {
5     public static void main(String[] args) {
6         Scanner scanner = new Scanner(System.in);
7         System.out.print("Ingrese el primer lado: ");
8         double a = scanner.nextDouble();
9
10        System.out.print("Ingrese el segundo lado: ");
11        double b = scanner.nextDouble();
12
13        System.out.print("Ingrese el tercer lado: ");
14        double c = scanner.nextDouble();
15
16        if (a + b > c && a + c > b && b + c > a) {
17            System.out.println("Sí forman un triángulo.");
18
19            if (a == b && b == c) {
20                System.out.println("Es un triángulo equilátero.");
21            } else if (a == b || a == c || b == c) {
22                System.out.println("Es un triángulo isósceles.");
23            } else {
24                System.out.println("Es un triángulo escaleno.");
25            }
26
27            double s = (a + b + c) / 2;
28            double area = Math.sqrt(s * (s - a) * (s - b) * (s - c));
29            System.out.printf("El área del triángulo es: %.2f\n", area);
30
31        } else {
32            System.out.println("Los valores NO forman un triángulo.");
33        }
34
35        scanner.close();
36    }
37 }

```

```

Ingrese el primer lado: 2
Ingrese el segundo lado: 5
Ingrese el tercer lado: 6
Sí forman un triángulo.
Es un triángulo escaleno.
El área del triángulo es: 4,68

```

**29. Determinar si un punto pertenece a la recta  $Y = 3X + 5$ .**

Hacer un algoritmo que entre la ordenada (Y) y la abscisa (X) de un punto de un plano cartesiano y, determine si pertenece o no a la recta  $Y = 3X + 5$ .

```
1 package taller1;
2 import java.util.Scanner;
3 public class ejercicio29 {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         System.out.print("Ingrese la abscisa (X): ");
8         double x = scanner.nextDouble();
9
10        System.out.print("Ingrese la ordenada (Y): ");
11        double y = scanner.nextDouble();
12
13        double yEsperado = 3 * x + 5;
14
15        if (y == yEsperado) {
16            System.out.println("El punto (" + x + ", " + y + ") pertenece a la recta Y = 3X + 5.");
17        } else {
18            System.out.println("El punto (" + x + ", " + y + ") NO pertenece a la recta Y = 3X + 5.");
19        }
20
21        scanner.close();
22    }
23
24 }
25
26 }
27 }
```

```
Ingrese la abscisa (X): 3
Ingrese la ordenada (Y): 4
El punto (3.0, 4.0) NO pertenece a la recta Y = 3X + 5.
```

**30. Verificar si un punto pertenece al área entre la parábola  $Y=4 - X^2$  y la recta  $Y=X - 3$ .**

```

1 package taller1;
2 import java.util.Scanner;
3 public class ejercicio30 {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         System.out.print("Ingrese la abscisa (X): ");
8         double x = scanner.nextDouble();
9
10        System.out.print("Ingrese la ordenada (Y): ");
11        double y = scanner.nextDouble();
12
13        double yParabola = 4 - Math.pow(x, 2);
14        double yRecta = x - 3;
15
16
17        if (x >= -2.82 && x <= 1.82 && y >= yRecta && y <= yParabola) {
18            System.out.println("El punto (" + x + ", " + y + ") está dentro del área entre la parábola y la recta.");
19        } else {
20            System.out.println("El punto (" + x + ", " + y + ") NO está dentro del área.");
21        }
22
23        scanner.close();
24    }
25 }
26

```

Ingrese la abscisa (X): 4  
Ingrese la ordenada (Y): 6  
El punto (4.0, 6.0) NO está dentro del área.

**31. Determinar si un punto pertenece al área comprendida entre  $Y = 2X - 2$ ,  $Y = X + 1$  y  $X = 20$ .**

```

1 package taller1;
2 import java.util.Scanner;
3 public class ejercicio31 {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         System.out.print("Ingrese la coordenada A (X): ");
8         double x = scanner.nextDouble();
9
10        System.out.print("Ingrese la coordenada B (Y): ");
11        double y = scanner.nextDouble();
12
13        double ySuperior = 2 * x - 2;
14        double yInferior = x + 1;
15
16        if (x <= 20 && y >= yInferior && y <= ySuperior) {
17            System.out.println("El punto (" + x + ", " + y + ") está dentro del área.");
18        } else {
19            System.out.println("El punto (" + x + ", " + y + ") NO está dentro del área.");
20        }
21
22        scanner.close();
23    }
24
25
26 }
27

```

```

Ingrese la coordenada A (X): 4
Ingrese la coordenada B (Y): 5
El punto (4.0, 5.0) está dentro del área.

```

### 32. Cálculo del valor a pagar por escritorios con descuento por cantidad.

Un almacén de escritorios hace los siguientes descuentos: si el cliente compra menos de 5 unidades se le da un descuento del 10% sobre la compra; si el número de unidades es mayor o igual a cinco pero menos de 10 se le otorga un 20% y, si son 10 o más se le da un 40%. Hacer un algoritmo que determine cuánto debe pagar un cliente si el valor de cada escritorio es de \$800.000.

```

import java.util.Scanner;
public class ej_32 {

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

        // Precio fijo por escritorio
        final float PRECIO_ESCRITORIO = 800000f;

        // Entrada: número de escritorios comprados
        System.out.print("Ingrese el número de escritorios comprados: ");
        int cantidad = Teclado.nextInt();

        // Calcular el valor total sin descuento
        float totalSinDescuento = cantidad * PRECIO_ESCRITORIO;

        float descuento;

        // Determinar porcentaje de descuento según la cantidad
        if (cantidad < 5) {
            descuento = 0.10f; // 10%
        } else if (cantidad < 10) {
            descuento = 0.20f; // 20%
        } else {
            descuento = 0.40f; // 40%
        }

        float valorDescuento = totalSinDescuento * descuento;
        float totalPagar = totalSinDescuento - valorDescuento;

        System.out.println("\n==> FACTURA ==");
        System.out.println("Cantidad de escritorios: " + cantidad);
        System.out.println("Precio unitario: $" + PRECIO_ESCRITORIO);
        System.out.println("Descuento aplicado: " + (descuento * 100) + "%");
        System.out.println("Valor del descuento: $" + valorDescuento);
        System.out.println("Total a pagar: $" + totalPagar);
    }
}

```

### 33. Juego de preguntas tipo “Sí” o “No”.

En un juego de preguntas que se responde “SI” o “NO”, gana quien responda correctamente las tres preguntas. Si se responde mal cualquiera de ellas, ya no se pregunta la siguiente y termina el juego. Las preguntas son:

1. ¿Simón Bolívar libertó a Colombia?
2. ¿Camilo Torres fue un guerrillero?
3. ¿El Binomio de Oro es un grupo de música vallenata?

Diseñe el registro de entrada.

```

package ejercicios;
import java.util.Scanner;
public class ej_33 {

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

        System.out.println("==> JUEGO DE PREGUNTAS ==>");
        System.out.println("Responde con 'SI' o 'NO'\n");

        // Pregunta 1
        System.out.print("1. ¿Simón Bolívar libertó a Colombia? ");
        String respuesta1 = Teclado.nextLine();

        if (respuesta1.equalsIgnoreCase("SI")) {
            // Pregunta 2
            System.out.print("2. ¿Camilo Torres fue un guerrillero? ");
            String respuesta2 = Teclado.nextLine();

            if (respuesta2.equalsIgnoreCase("SI")) {
                // Pregunta 3
                System.out.print("3. ¿El Binomio de Oro es un grupo de música vallenata? ");
                String respuesta3 = Teclado.nextLine();

                if (respuesta3.equalsIgnoreCase("SI")) {
                    System.out.println(" Has ganado, todas las respuestas son correctas.");
                } else {
                    System.out.println("Respuesta incorrecta. Has perdido el juego.");
                }
            } else {
                System.out.println("Respuesta incorrecta. Has perdido el juego.");
            }
        } else {
            System.out.println("Respuesta incorrecta. Has perdido el juego.");
        }
    }
}

```

### 34. Cálculo del precio total de manzanas con descuento según cantidad comprada.

Una frutería ofrece las manzanas con descuento según la siguiente tabla:

No. de manzanas compradas % descuento

0 – 2 0%

3 – 5 10%

6 – 10 15%

11 en adelante 20%

Determinar cuánto pagará una persona que compre manzanas en esa frutería.

```
package ejercicios;
import java.util.Scanner;
public class ej_34 {

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

        // Precio por manzana (puedes cambiarlo si se desea otro valor)
        final float PRECIO_MANZANA = 1000f;

        System.out.println("==> FRUTERÍA LAS DELICIAS ==>");
        System.out.print("Ingrese la cantidad de manzanas compradas: ");
        int cantidad = Teclado.nextInt();

        float totalSinDescuento = cantidad * PRECIO_MANZANA;

        // Determinar el descuento según la cantidad
        float descuento;

        if (cantidad >= 0 && cantidad <= 2) {
            descuento = 0f; // 0%
        } else if (cantidad <= 5) {
            descuento = 0.10f; // 10%
        } else if (cantidad <= 10) {
            descuento = 0.15f; // 15%
        } else {
            descuento = 0.20f; // 20%
        }

        float valorDescuento = totalSinDescuento * descuento;
        float totalPagar = totalSinDescuento - valorDescuento;

        System.out.println("\n==> FACTURA ==>");
        System.out.println("Cantidad de manzanas: " + cantidad);
        System.out.println("Precio unitario: $" + PRECIO_MANZANA);
        System.out.println("Descuento aplicado: " + (descuento * 100) + "%");
        System.out.println("Valor del descuento: $" + valorDescuento);
        System.out.println("Total a pagar: $" + totalPagar);
    }
}
```

### 35. Determinación de créditos y descuentos para estudiantes según promedio y tipo de programa.

Cierta universidad tiene un programa para estimular a los estudiantes con buen rendimiento académico. Si el promedio es de 4,5 o más y el alumno es de pregrado entonces cursará 28 créditos y se le hará un 25% de descuento. Si el promedio es mayor o igual a 4,0 pero menor que 4,5 y el alumno es de pregrado, entonces cursará 25 créditos y se le hará un 10% de

descuento. Si el promedio es mayor que 3,5 y menor que 4,0 y es de pregrado, cursará 20 créditos y no tendrá ningún descuento. Si el promedio es mayor o igual a 2,5 y menor que 3,5 y es de pregrado, cursará 15 créditos y no tendrá descuento. Si el promedio es menor de 2,5 y es de pregrado, no podrá matricularse. Si el promedio es mayor o igual a 4,5 y es de posgrado, cursará 20 créditos y se le hará un 20% de descuento. Si el promedio es menor de 4,5 y es de posgrado cursará 10 créditos y no tendrá descuento.

```

package ejercicios;
import java.util.Scanner;
public class ej_35 {

    public static void main(String[] args) {
        Scanner Teclado = new Scanner(system.in);

        System.out.println("== UNIVERSIDAD - ESTÍMULO ACADÉMICO ==");

        // Entrada de datos
        System.out.print("Ingrese el promedio del estudiante: ");
        float promedio = Teclado.nextFloat();
        Teclado.nextLine();
        System.out.print("Ingrese el tipo de programa (pregrado / posgrado): ");
        String tipo = Teclado.nextLine();

        int creditos = 0;
        float descuento = 0f;

        // Verificar tipo de estudiante
        if (tipo.equalsIgnoreCase("pregrado")) {
            if (promedio >= 4.5f) {
                creditos = 28;
                descuento = 0.25f; // 25%
            } else if (promedio >= 4.0f && promedio < 4.5f) {
                creditos = 25;
                descuento = 0.10f; // 10%
            } else if (promedio > 3.5f && promedio < 4.0f) {
                creditos = 20;
                descuento = 0f; // sin descuento
            } else if (promedio >= 2.5f && promedio <= 3.5f) {
                creditos = 15;
                descuento = 0f; // sin descuento
            } else {
                System.out.println("Promedio insuficiente. No puede matricularse.");
                return; // termina el programa
            }
        } else if (tipo.equalsIgnoreCase("posgrado")) {

        } else if (tipo.equalsIgnoreCase("posgrado")) {

            if (promedio >= 4.5f) {
                creditos = 20;
                descuento = 0.20f; // 20%
            } else {
                creditos = 10;
                descuento = 0f; // sin descuento
            }
        } else {
            System.out.println("Tipo de programa no válido. Debe escribir 'pregrado' o 'posgrado'.");
            return;
        }
        System.out.println("\n== RESULTADOS ==");
        System.out.println("Tipo de estudiante: " + tipo);
        System.out.println("Promedio: " + promedio);
        System.out.println("Créditos asignados: " + creditos);
        System.out.println("Descuento aplicado: " + (descuento * 100) + "%");

        System.out.println("Proceso finalizado correctamente.");
    }
}

```

### 38. Suma de números impares comprendidos entre 1 y $N$ .

```
package ejercicios;
import java.util.Scanner;
public class ej_38 {

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

        System.out.println("==> SUMA DE NÚMEROS IMPARES ==>");
        System.out.print("Ingrese el valor de N: ");
        int N = Teclado.nextInt();

        int suma = 0;

        for (int i = 1; i <= N; i++) {
            if (i % 2 != 0) { // Verifica si i es impar
                suma += i;    // Acumula los impares
            }
        }

        System.out.println("La suma de los números impares entre 1 y " + N + " es: " + suma);
    }
}
```

### 39. Cálculo de la desviación típica de un conjunto de datos positivos.

```
package ejercicios;
import java.util.Scanner;
public class ej_39 {

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

        System.out.println("==> CÁLCULO DE LA DESVIACIÓN TÍPICA ==>");
        System.out.print("Ingrese la cantidad de datos: ");
        int n = Teclado.nextInt();

        float[] datos = new float[n];
        float suma = 0;

        // Leer los datos
        for (int i = 0; i < n; i++) {
            System.out.print("Datos " + (i + 1) + ": ");
            datos[i] = Teclado.nextFloat();
            suma += datos[i];
        }

        // Calcular la media
        float media = suma / n;

        float sumaDiferencias = 0;
        for (int i = 0; i < n; i++) {
            float diferencia = datos[i] - media;
            sumaDiferencias += diferencia * diferencia;
        }

        // Calcular la desviación típica
        double desviacion = Math.sqrt(sumaDiferencias / n); //math.sqrt = raíz cuadrada

        System.out.println("\nLa media es: " + media);
        System.out.println("La desviación típica es: " + desviacion);
    }
}
```

#### 40. Cálculo de la raíz cuadrada, cuadrado y cubo de números enteros positivos.

```
package ejercicios;
import java.util.Scanner;
public class ej_40 {

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

        System.out.println("==> RAÍZ CUADRADA, CUADRADO Y CUBO DE NÚMEROS ==>");
        System.out.print("¿Cuántos números desea ingresar?: ");
        int n = Teclado.nextInt();

        int[] numeros = new int[n];

        // Leer los números
        for (int i = 0; i < n; i++) {
            System.out.print("Ingrese el número " + (i + 1) + ": ");
            numeros[i] = Teclado.nextInt();
        }

        System.out.println("\nResultados:");
        System.out.println("Número | Raíz Cuadrada | Cuadrado | Cubo");

        // Calcular y mostrar resultados
        for (int i = 0; i < n; i++) {
            int numero = numeros[i];
            float raiz = (float) Math.sqrt(numero); // Raíz Cuadrada
            float cuadrado = numero * numero; // Cuadrado
            float cubo = numero * numero * numero; // Cubo

            System.out.println("Número: " + numero +
                               " Raíz Cuadrada: " + raiz +
                               " Cuadrado: " + cuadrado +
                               " Cubo: " + cubo);
        }
    }
}
```

#### 41. Determinar el mayor valor dentro de un grupo de datos positivos.

```
package ejercicios;
import java.util.Scanner;
public class ej_41 {
    public static void main(String[] args) {
        Scanner Teclado = new Scanner(System.in);

        System.out.println("== ENCONTRAR EL MAYOR VALOR ==");
        System.out.print("Ingrese la cantidad de datos: ");
        int n = Teclado.nextInt();

        float[] datos = new float[n];
        float mayor;

        System.out.print("Ingrese el dato 1: ");
        datos[0] = Teclado.nextFloat();
        mayor = datos[0];

        for (int i = 1; i < n; i++) {
            System.out.print("Ingrese el dato " + (i + 1) + ": ");
            datos[i] = Teclado.nextFloat();

            if (datos[i] > mayor) {
                mayor = datos[i];
            }
        }

        System.out.println("\nEl mayor valor ingresado es: " + mayor);
    }
}
```

#### 42. Sumar el valor mayor y menor entre un conjunto de datos positivos.

```
package ejercicios;
import java.util.Scanner;
public class ej_42 {
    public static void main(String[] args) {
        Scanner Teclado = new Scanner(System.in);

        System.out.println("== SUMA DEL MAYOR Y MENOR VALOR ==");
        System.out.print("Ingrese la cantidad de datos: ");
        int n = Teclado.nextInt();

        float[] datos = new float[n];

        System.out.print("Ingrese el dato 1: ");
        datos[0] = Teclado.nextFloat();

        float mayor = datos[0];
        float menor = datos[0];

        for (int i = 1; i < n; i++) {
            System.out.print("Ingrese el dato " + (i + 1) + ": ");
            datos[i] = Teclado.nextFloat();

            if (datos[i] > mayor) {
                mayor = datos[i];
            }

            if (datos[i] < menor) {
                menor = datos[i];
            }
        }

        float suma = mayor + menor;

        System.out.println("El valor mayor es: " + mayor);
        System.out.println("El valor menor es: " + menor);
        System.out.println("La suma del mayor y el menor es: " + suma);
    }
}
```

**43. Hacer un algoritmo que determine la cantidad de trios de valores, entre un conjunto de ternas que representen triángulos rectángulos.**

```
*ejer_43.java ×
package cartilla;
import java.util.Scanner;
public class ejer_43 {
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);
        int cantidad; // cantidad de ternas
        int contador = 0; // cuenta los triángulos rectángulos

        System.out.print("Ingrese la cantidad de ternas: ");
        cantidad = teclado.nextInt();

        // Bucle para leer cada terna
        for (int i = 1; i <= cantidad; i++) {
            System.out.println("\nTerna #" + i);
            System.out.print("Ingrese el valor de a: ");
            int a = teclado.nextInt();
            System.out.print("Ingrese el valor de b: ");
            int b = teclado.nextInt();
            System.out.print("Ingrese el valor de c: ");
            int c = teclado.nextInt();
            // Determinar cuál es el mayor (hipotenusa)
            int hip, cat1, cat2;
            if (a >= b && a >= c) {
                hip = a; cat1 = b; cat2 = c;
            } else if (b >= a && b >= c) {
                hip = b; cat1 = a; cat2 = c;
            } else {
                hip = c; cat1 = a; cat2 = b;
            }
            // Verificar si cumple Pitágoras
            if (cat1 * cat1 + cat2 * cat2 == hip * hip) {
                System.out.println("✓ Es un triángulo rectángulo.");
                contador++;
            } else {
                System.out.println("✗ No es un triángulo rectángulo.");
            }
        }
        System.out.println("\nCantidad total de triángulos rectángulos: " + contador);
        teclado.close();
    }
}
```

**44. En cada uno de una serie de registros se encuentran tres valores que posiblemente representan los tres lados de un triángulo.** Hacer un algoritmo que determine cuántos triángulos equiláteros, isósceles y escalenos hay.

```
ejer_43.java    ejer_44.java ×
1 package cartilla;
2
3 import java.util.Scanner;
4
5 public class ejer_44 {
6
7     public static void main(String[] args) {
8         Scanner t = new Scanner(System.in);
9
10        int n, eq = 0, iso = 0, esc = 0;
11
12        System.out.print("Cuantas ternas desea ingresar? ");
13        n = t.nextInt();
14
15        for (int i = 1; i <= n; i++) {
16            System.out.println("\nTerna #" + i);
17            int a = t.nextInt(), b = t.nextInt(), c = t.nextInt();
18
19            if (a + b > c && a + c > b && b + c > a) {
20                if (a == b && b == c) eq++;
21                else if (a == b || a == c || b == c) iso++;
22                else esc++;
23            } else {
24                System.out.println("No forman un triangulo valido.");
25            }
26        }
27
28        System.out.println("\nEquilateros: " + eq);
29        System.out.println("Isosceles: " + iso);
30        System.out.println("Escalenos: " + esc);
31
32        t.close();
33    }
34}
```

**45. Hacer un algoritmo que encuentre la cantidad de valores enteros que hay entre un par de números positivos.**

```
1 package cartilla;
2
3 import java.util.Scanner;
4
5 public class ejer_45 {
6
7     public static void main(String[] args) {
8         Scanner t = new Scanner(System.in);
9
10        System.out.print("Ingrese el primer numero: ");
11        int a = t.nextInt();
12
13        System.out.print("Ingrese el segundo numero: ");
14        int b = t.nextInt();
15
16        if (a > 0 && b > 0) {
17            int cantidad = Math.abs(a - b) - 1; // resta absoluta menos los extremos
18            if (cantidad < 0) cantidad = 0;      // si son iguales o negativos
19            System.out.println("Cantidad de enteros entre " + a + " y " + b + ": " + cantidad);
20        } else {
21            System.out.println("Ambos numeros deben ser positivos.");
22        }
23
24        t.close();
25    }
26}
```

**46. Realizar un algoritmo que escriba la posición de la cantidad de puntos que estén ubicados en el primer cuadrante de una circunferencia con centro (0,0).**

```
1 package cartilla;
2
3 import java.util.Scanner;
4
5 public class ejer_46 {
6     public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8         int n, contador = 0;
9
10        System.out.print("Ingrese la cantidad de puntos: ");
11        n = sc.nextInt();
12
13        for (int i = 1; i <= n; i++) {
14            System.out.print("Punto " + i + " (x y): ");
15            double x = sc.nextDouble();
16            double y = sc.nextDouble();
17
18            if (x > 0 && y > 0) {
19                contador++;
20                System.out.println("- El punto " + i + " está en el primer cuadrante.");
21            }
22        }
23
24        System.out.println("\nTotal de puntos en el primer cuadrante: " + contador);
25        sc.close();
26    }
27}
```

**47. Varias ambulancias recorren la ciudad y cuando se recibe en la CENTRAL una llamada se**

informa la ubicación de la emergencia mediante coordenadas, lo mismo que la ubicación de todas las ambulancias.

- La central es el punto (0,0) u origen de las coordenadas.
- Se sabe que existen N ambulancias en servicio.
- Realice un algoritmo que, dada la información necesaria, informe las coordenadas de la ambulancia más cercana al punto de emergencia.

```
package cartilla;
import java.util.Scanner;

public class ejer_47 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Ingrese las coordenadas de la emergencia (x y): ");
        double xE = sc.nextDouble();
        double yE = sc.nextDouble();

        System.out.print("Ingrese la cantidad de ambulancias: ");
        int n = sc.nextInt();

        double minDist = Double.MAX_VALUE;
        double ambX = 0, ambY = 0;

        for (int i = 1; i <= n; i++) {
            System.out.print("Coordenadas de ambulancia " + i + " (x y): ");
            double xA = sc.nextDouble();
            double yA = sc.nextDouble();

            double distancia = Math.sqrt(Math.pow(xE - xA, 2) + Math.pow(yE - yA, 2));

            if (distancia < minDist) {
                minDist = distancia;
                ambX = xA;
                ambY = yA;
            }
        }

        System.out.println("\nLa ambulancia más cercana está en: (" + ambX + ", " + ambY + ")");
        System.out.println("Distancia: " + minDist);
        sc.close();
    }
}
```

**48. Dados N valores, diseñe un algoritmo que haga el siguiente proceso:**

- Si el valor es menor que cero calcular su cubo.
- Si el valor está entre 0 y 100 calcular su cuadrado.
- Si el valor está entre 101 y 1000 calcular su raíz cuadrada.

```
1 package cartilla;
2 import java.util.Scanner;
3
4 public class ejer_48 {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7
8         System.out.print("Ingrese la cantidad de valores N: ");
9         int N = sc.nextInt();
10
11        for (int i = 1; i <= N; i++) {
12            System.out.print("Valor " + i + ": ");
13            double num = sc.nextDouble();
14
15            if (num < 0) {
16                System.out.println("Cubo: " + Math.pow(num, 3));
17            } else if (num <= 100) {
18                System.out.println("Cuadrado: " + Math.pow(num, 2));
19            } else if (num <= 1000) {
20                System.out.println("Raíz cuadrada: " + Math.sqrt(num));
21            } else {
22                System.out.println("Fuera de rango (mayor que 1000)");
23            }
24        }
25
26        sc.close();
27    }
28 }
```

**49. Elaborar un algoritmo que determine cuántos son los intereses generados en cada uno de**

los N períodos, por un capital de X pesos, que se invierte a una cantidad de P por ciento, sin retirar intereses.

```

package cartilla;
import java.util.Scanner;

public class ejer_49 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Ingrese el capital inicial (X): ");
        double capital = sc.nextDouble();

        System.out.print("Ingrese la tasa de interés (% P): ");
        double tasa = sc.nextDouble() / 100;

        System.out.print("Ingrese el número de períodos (N): ");
        int N = sc.nextInt();

        for (int i = 1; i <= N; i++) {
            double interes = capital * tasa;
            capital += interes;
            System.out.println("Período " + i + ": Interés generado = " + interes + " | Capital acumulado = " + capital);
        }
        sc.close();
    }
}

```

**50.** Elaborar un algoritmo que encuentre el factorial de los números comprendidos entre 1 y N.

```

package cartilla;
import java.util.Scanner;

public class ejer_50 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Ingrese el valor de N: ");
        int N = sc.nextInt();

        for (int i = 1; i <= N; i++) {
            long factorial = 1;
            for (int j = 1; j <= i; j++) {
                factorial *= j;
            }
            System.out.println("El factorial de " + i + " es: " + factorial);
        }
        sc.close();
    }
}

```

**51.** Hacer un algoritmo que entre dos valores A y B y encuentre AB mediante sumas únicamente.

```

1 package cartilla;
2 import java.util.Scanner;
3
4 public class ejer_51 {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7
8         System.out.print("Ingrese el valor de A: ");
9         int A = sc.nextInt();
10        System.out.print("Ingrese el valor de B: ");
11        int B = sc.nextInt();
12
13        int resultado = 1; // Para almacenar el valor de A^B
14
15        // Repetir B veces la multiplicación (hecha solo con sumas)
16        for (int i = 1; i <= B; i++) {
17            int acumulador = 0;
18            for (int j = 1; j <= A; j++) {
19                acumulador += resultado; // Multiplica usando sumas
20            }
21            resultado = acumulador;
22        }
23
24        System.out.println("El resultado de " + A + "^" + B + " es: " + resultado);
25        sc.close();
26    }
27 }

```

**52. Elaborar un algoritmo que muestre los enteros desde 1 hasta N y sus cuadrados, calculados solamente con sumas y utilizando el método propuesto. Número Cuadrado Método**

1 1 1

2 4 1 + 3

3 9 1 + 3 + 5

4 16 1 + 3 + 5 + 7

5 25 1 + 3 + 5 + 7 + 9

...

...

...

N

```

1 package cartilla;
2 import java.util.Scanner;
3
4 public class ejer_52 {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7
8         System.out.print("Ingrese el valor de N: ");
9         int N = sc.nextInt();
10
11         int cuadrado = 0; // Guardará el cuadrado actual
12         int impar = 1; // Primer número impar
13
14         System.out.println("Número\tCuadrado\tMétodo");
15
16         for (int i = 1; i <= N; i++) {
17             cuadrado += impar; // Sumar el número impar correspondiente
18             System.out.print(i + "\t" + cuadrado + "\t\t");
19
20             // Mostrar el método (por ejemplo: 1 + 3 + 5 ...)
21             for (int j = 1; j <= i; j++) {
22                 System.out.print((2 * j - 1));
23                 if (j < i) System.out.print(" + ");
24             }
25             System.out.println();
26
27             impar += 2; // Pasar al siguiente número impar
28         }
29
30         sc.close();
31     }
32 }

```

## 16. Referencias

GitHub. (2025). *POO y Manual* [Repositorio en línea]. GitHub. Disponible en: <https://github.com/Asnarck7/POO-y-Manual.git>

Montoya, A. (2025). *Lógica de Programación – Efraín Oviedo* [Archivo PDF]. Universidad Cooperativa de Colombia. Disponible en: [https://campusuccedu-my.sharepoint.com/personal/alexander\\_montoyai\\_campusucc\\_edu\\_co/\\_layouts/15/onedrive.aspx?viewid=9a959662-604f-4451-b6cb-ffb85b572146](https://campusuccedu-my.sharepoint.com/personal/alexander_montoyai_campusucc_edu_co/_layouts/15/onedrive.aspx?viewid=9a959662-604f-4451-b6cb-ffb85b572146)

Programación con Efraín Oviedo. (2025). Introducción a la lógica de programación [Video]. YouTube.

<https://www.youtube.com/watch?v=oVTfqSNOAJk&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=1>

Programación con Efraín Oviedo. (2025). Tipos de datos y variables [Video]. YouTube. <https://www.youtube.com/watch?v=5IKaGP4gpRg&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=3>

Programación con Efraín Oviedo. (2025). Operadores y expresiones [Video]. YouTube. <https://www.youtube.com/watch?v=HvIgH9P000M&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=4>

Programación con Efraín Oviedo. (2025). Estructuras de control [Video]. YouTube. [https://www.youtube.com/watch?v=OIC\\_cKnO3jc&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=5](https://www.youtube.com/watch?v=OIC_cKnO3jc&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=5)

Programación con Efraín Oviedo. (2025). Ciclos y bucles [Video]. YouTube. <https://www.youtube.com/watch?v=p3ZNQrTscck&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=6>

Programación con Efraín Oviedo. (2025). Funciones y procedimientos [Video]. YouTube. [https://www.youtube.com/watch?v=6azC\\_g7Nvgo&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb](https://www.youtube.com/watch?v=6azC_g7Nvgo&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb)

## 269k&index=7

Programación con Efraín Oviedo. (2025). Arreglos y matrices [Video]. YouTube.  
<https://www.youtube.com/watch?v=9ylWF DOS7lg&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=8>

Programación con Efraín Oviedo. (2025). Estructuras de datos [Video]. YouTube.  
<https://www.youtube.com/watch?v=HNybZd2UYRg&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=9>

Programación con Efraín Oviedo. (2025). Algoritmos básicos [Video]. YouTube.  
<https://www.youtube.com/watch?v=EeWsmfXMWcw&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=10>

Programación con Efraín Oviedo. (2025). Depuración y errores comunes [Video]. YouTube.  
<https://www.youtube.com/watch?v=W81Kl7a6IfU&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=11>

Programación con Efraín Oviedo. (2025). Programación orientada a objetos I [Video]. YouTube.  
<https://www.youtube.com/watch?v=REVBA9375uk&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=12>

Programación con Efraín Oviedo. (2025). Programación orientada a objetos II [Video]. YouTube.  
<https://www.youtube.com/watch?v=JTVRLypnMsg&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=13>

Programación con Efraín Oviedo. (2025). Herencia y polimorfismo [Video]. YouTube.  
<https://www.youtube.com/watch?v=HJjcJXBkcT8&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=15>

Programación con Efraín Oviedo. (2025). Manejo de archivos [Video]. YouTube.  
<https://www.youtube.com/watch?v=o986XvrenJE&list=PLIZIYZ60C9rsJp72PVO7Ku0c0L6Eb269k&index=16>

Programación con Efraín Oviedo. (2025). Excepciones y errores [Video]. YouTube.  
<https://www.youtube.com/watch?v=mH5Wjew8ngE&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=17>

Programación con Efraín Oviedo. (2025). Práctica de programación [Video]. YouTube.  
<https://www.youtube.com/watch?v=Lh5esLbPGDk&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=18>

Programación con Efraín Oviedo. (2025). Proyecto final [Video]. YouTube.  
<https://www.youtube.com/watch?v=5ush6eRWv0U&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=19>

Programación con Efraín Oviedo. (2025). Optimización de código [Video]. YouTube.  
<https://www.youtube.com/watch?v=eIxgkttAwvM&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=20>

Programación con Efraín Oviedo. (2025). Consejos y buenas prácticas [Video]. YouTube.  
[https://www.youtube.com/watch?v=-rfoM5HC\\_cw&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=21](https://www.youtube.com/watch?v=-rfoM5HC_cw&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=21)

Programación con Efraín Oviedo. (2025). Recursos adicionales [Video]. YouTube.  
<https://www.youtube.com/watch?v=5K-2FuA5U9s&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=22>

Programación con Efraín Oviedo. (2025). Preguntas frecuentes [Video]. YouTube.  
<https://www.youtube.com/watch?v=nArtCkW4H9A&list=PLIZIYZ60C9rsJp72PVQ7Ku0c0L6Eb269k&index=23>