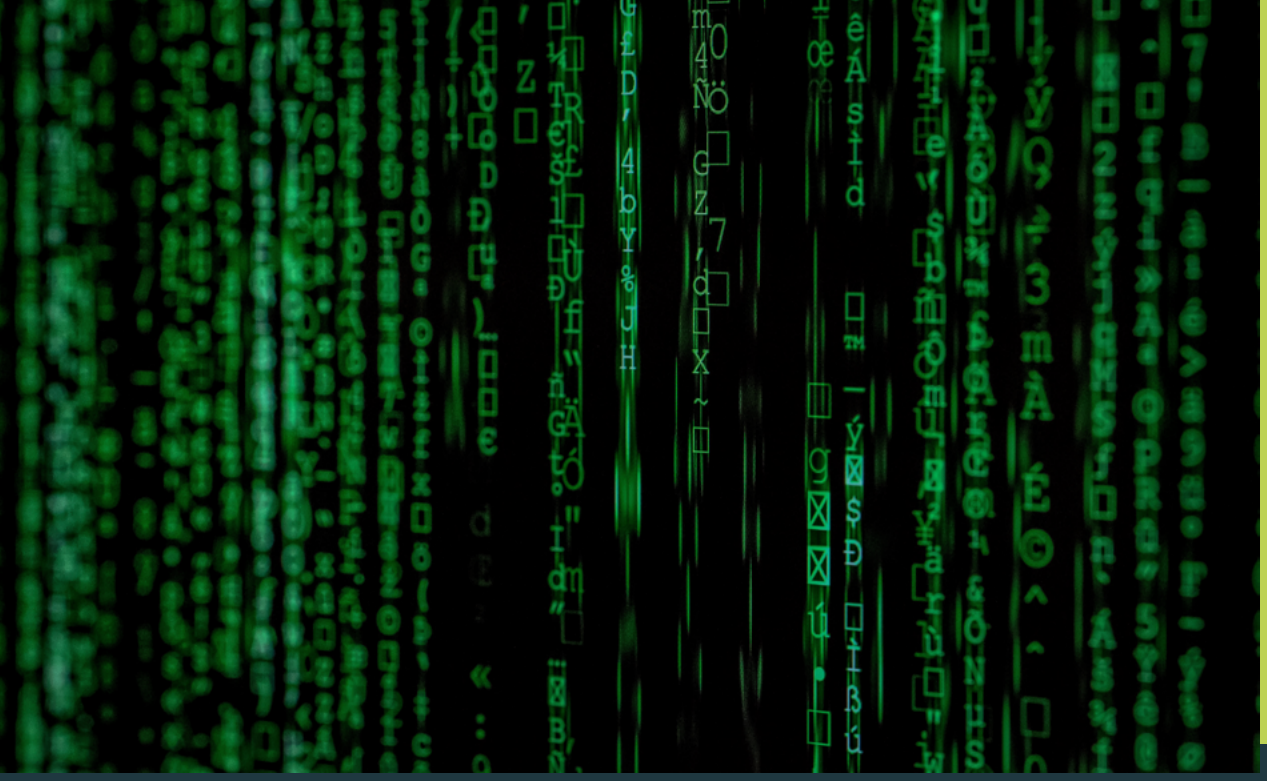

Unidad 0

Primeros pasos

Bloque: Instalaciones, Hola Mundo! y comentario

Parte 1: Guía de Configuración y Desarrollo en Java

Introducción

El desarrollo en Java requiere una configuración básica que incluye herramientas esenciales como el **JDK (Java Development Kit)** y un **IDE (Entorno de Desarrollo Integrado)**. Estas herramientas simplifican la escritura, compilación y ejecución de programas. Esta guía cubre todo lo necesario para instalar, configurar y comenzar a programar en Java, incluyendo un ejercicio introductorio.

1. Componentes necesarios para programar en Java

1.1. JDK (Java Development Kit)

El JDK es el kit de herramientas necesario para programar en Java. Incluye:

- **Compilador (`javac`)**: Convierte el código fuente en bytecode, el formato que entiende la JVM.
- **Máquina Virtual de Java (JVM)**: Ejecuta los programas compilados.
- **Herramientas adicionales**: Como `java` (para ejecutar programas) y `javadoc` (para generar documentación).

1.2. IDE o Editor de Texto

Un IDE facilita la programación al incluir funciones como:

- Autocompletado de código.
- Depuración (debugging).
- Organización de proyectos.

Recomendaciones de IDE:

- **Eclipse**: Ideal para principiantes y gratuito.
- **IntelliJ IDEA**: Popular y con edición gratuita (Community).
- **Visual Studio Code**: Versátil y con soporte para múltiples lenguajes, incluyendo Java.

2. Guía de instalación del JDK

2.1. Instalación en Windows

1. Descargar el JDK:

- Ve a la página oficial: Descargar JDK.
- Elige el instalador para Windows.

2. Instalar el JDK:

- Ejecuta el archivo descargado y sigue las instrucciones.
- Por defecto, el JDK se instalará en: `C:\Program Files\Java\jdk<versión>`.

3. Configurar Variables de Entorno:

- Abre las configuraciones avanzadas del sistema.
- En "Variables de entorno", añade:
  - Variable `JAVA_HOME`: Apunta a la carpeta del JDK.
  - Modifica la variable `Path` para incluir `%JAVA_HOME%\bin`.

4. Validar instalación:

- Abre el terminal (o PowerShell) y ejecuta:

```
java -version
javac -version
```

- Si ves la versión del JDK, la instalación fue exitosa.

2.2. Instalación en macOS

1. Descargar el JDK:

- Descarga el archivo `.dmg` desde la página oficial de Oracle.

2. Instalar:

- Abre el archivo descargado y sigue las instrucciones.

3. Configurar Variables de Entorno:

- Edita el archivo de configuración del shell (`~/.zshrc` o `~/.bash_profile`):

```
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk<versión>/Contents/Home
```

- Aplica los cambios con `source ~/.zshrc`.

4. Validar instalación:

- Ejecuta en la terminal:

```
java -version
javac -version
```

- Si ves la versión del JDK, está correctamente instalado.

3. Configuración de un IDE

3.1. Eclipse

1. Descarga Eclipse desde Eclipse Downloads.
2. Descomprime el archivo descargado e instala.
3. Ejecuta `eclipse.exe` y selecciona un espacio de trabajo.

3.2. IntelliJ IDEA

1. Descarga la edición Community desde JetBrains IntelliJ IDEA.
2. Instala y abre el IDE.
3. Crea un proyecto nuevo seleccionando "Java" como tipo de proyecto.

3.3. Visual Studio Code

1. Descarga Visual Studio Code desde Visual Studio Code.
2. Instala la extensión "Java Extension Pack".
3. Configura el entorno siguiendo las indicaciones del asistente.

4. Escribe y ejecuta tu primer programa en Java

4.1. Código: Hola Mundo

Crea un archivo llamado `HolaMundo.java` y escribe el siguiente código:

```
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println("¡Hola, Mundo!");
    }
}
```

4.2. Ejecuta el programa

- **Desde el terminal**:

  1. Compila el programa:

```
javac HolaMundo.java
```

  2. Ejecuta el programa:

```
java HolaMundo
```

- **Desde un IDE**:
  1. Crea un nuevo proyecto.
  2. Añade una clase con el código anterior.
  3. Haz clic en “Run” para ejecutar el programa.

4.3. Resultado esperado

```
¡Hola, Mundo!
```

5. Ejercicio práctico

Objetivo: Escribir un programa que calcule el área de un rectángulo.

Código sugerido:

```java
import java.util.Scanner;
public class AreaRectangulo {
    public static void main(String[] args) {
          Scanner scanner = new Scanner(System.in);
          System.out.print("Introduce la base del rectángulo: ");
          double base = scanner.nextDouble();
          System.out.print("Introduce la altura del rectángulo: ");
          double altura = scanner.nextDouble();
          double area = base * altura;
          System.out.println("El área del rectángulo es: " + area);
    }
}
```

Pasos para ejecutarlo:

1. Escribe y guarda el código en un archivo llamado `AreaRectangulo.java`.

2. Compila y ejecuta usando la terminal o un IDE.

6. Test de evaluación

1. ¿Qué herramienta se utiliza para compilar un programa en Java?

a) `java`
b) `javac`
c) `javadoc`

2. ¿Qué variable de entorno debe configurarse para el JDK?

a) `JAVA_PATH`
b) `JAVA_HOME`
c) `CLASSPATH`

3. ¿Qué comando verifica la instalación de Java?

a) `java -version`
b) `javac -ver`
c) `java -config`

4. ¿Cuál es la función del IDE en el desarrollo en Java?

a) Ejecutar programas en código máquina.
b) Proveer herramientas integradas para escribir y depurar código.
c) Servir como compilador de sistemas.

Parte 2: Introducción a Java

Java es un lenguaje de programación de propósito general, creado en 1995 por **James Gosling** en **Sun Microsystems** (actualmente propiedad de Oracle Corporation). Su diseño versátil y robusto lo ha convertido en una de las tecnologías más populares en el desarrollo de software.

1. Características principales:

1. **Versatilidad: Java se utiliza en aplicaciones móviles, web, empresariales, videojuegos, sistemas embebidos y más.**
2. **Portabilidad: Gracias a la JVM (Java Virtual Machine), los programas escritos en Java pueden ejecutarse en cualquier sistema operativo compatible.**
3. **Orientación a Objetos: Este enfoque facilita la reutilización y organización del código, promoviendo buenas prácticas de desarrollo.**

"Escribe una vez, ejecuta en cualquier lugar"

El lema principal de Java, **"Write Once, Run Anywhere" (WORA)**, refleja su capacidad de portabilidad. Este concepto es posible gracias a:

- **Java Virtual Machine (JVM)**:
  - Una máquina virtual que interpreta el **bytecode**, un formato independiente del sistema operativo generado al compilar el código Java.
  - Permite ejecutar el mismo programa en diferentes plataformas sin necesidad de modificaciones.

Ejemplo de flujo de desarrollo:

1. **Escribes el código fuente en un archivo `.java`.**
2. **Compilas el código con `javac`, generando un archivo `.class` con bytecode.**
3. **La JVM interpreta el bytecode y lo ejecuta en el sistema.**

2. ¿Por qué aprender Java?

Aprender Java tiene múltiples beneficios tanto para principiantes como para desarrolladores avanzados. Sus principales ventajas son:

1. **Simplicidad: Su sintaxis es clara y fácil de entender.**
2. **Orientación a Objetos: Facilita la organización lógica y reutilización del código.**
3. **Portabilidad: Funciona en múltiples plataformas sin necesidad de ajustes.**
4. **Seguridad: Protege el entorno de ejecución frente a errores y vulnerabilidades.**
5. **Comunidad activa: Amplia documentación, foros y recursos para resolver dudas.**
6. **Aplicabilidad universal: Desde aplicaciones móviles (como Android) hasta soluciones empresariales complejas.**

3. Cómo funciona Java

El proceso de desarrollo en Java sigue tres pasos principales:

1. Escritura del código

- El código fuente se escribe en un archivo con extensión `.java`. Este archivo contiene instrucciones en un lenguaje comprensible para el programador.

2. Compilación

- El compilador de Java (`javac`) traduce el código fuente en **bytecode**, un formato intermedio que puede ser ejecutado en cualquier sistema con una JVM.

3. Ejecución

- La JVM interpreta y ejecuta el bytecode, adaptándolo al sistema operativo donde se ejecuta.

4. Tu primer programa en Java: "¡Hola, Mundo!"

```
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println("¡Hola, Mundo!");
    }
}
```

Explicación:

1. Clase (`class`): Define la estructura del programa. En este caso, la clase se llama `HolaMundo`.

2. Método `main`: Es el punto de entrada del programa. Java siempre comienza la ejecución aquí.

3. `System.out.println`: Es el comando utilizado para imprimir texto en la consola.

Resultado esperado:

```
¡Hola, Mundo!
```

5. Conceptos clave

- **Clase**: Estructura fundamental que organiza el código en Java.
- **Método `main`**: Es el punto donde comienza la ejecución del programa.
- **Compilación**: Proceso de convertir código fuente (.java) en bytecode (.class).
- **JVM**: Interpreta el bytecode y lo ejecuta en cualquier sistema operativo compatible.

6. Ejercicio: Imprimir múltiples mensajes

Objetivo: Crear un programa que imprima tres mensajes diferentes en la consola.

```
public class Mensajes {
    public static void main(String[] args) {
        System.out.println("¡Bienvenido a Java!");
        System.out.println("Java es un lenguaje poderoso.");
        System.out.println("Vamos a aprender juntos.");
    }
}
```

Salida esperada:

```
¡Bienvenido a Java!
Java es un lenguaje poderoso.
Vamos a aprender juntos.
```

7. Ejercicio: Operaciones matemáticas

Objetivo: Escribir un programa que sume dos números y muestre el resultado.

```
public class Suma {
    public static void main(String[] args) {
        int numero1 = 8;
        int numero2 = 12;
        int resultado = numero1 + numero2;
        System.out.println("La suma de " + numero1 + " y " + numero2 + " es
" + resultado);
    }
}
```

Salida esperada:

```
La suma de 8 y 12 es 20
```

8. Test de evaluación

1. ¿Qué hace posible el lema “Escribe una vez, ejecuta en cualquier lugar”?

a) Compatibilidad con todos los lenguajes.
b) Compatibilidad con cualquier sistema con JVM instalada.
c) Evitar errores durante la compilación.

2. ¿Cuál es el propósito del método `main`?

a) Declarar variables globales.
b) Especificar el punto de inicio del programa.
c) Realizar operaciones matemáticas.

3. ¿Qué componente convierte el código fuente en bytecode?

a) JVM.
b) JRE.
c) Compilador `javac`.

4. ¿Cuál es la línea correcta para imprimir “¡Hola, Mundo!” en Java?

a) `System.print(“¡Hola, Mundo!”);`
b) `System.out.println(“¡Hola, Mundo!”);`
c) `Console.write(“¡Hola, Mundo!”);`

5. ¿Qué extensión tiene el archivo generado tras compilar un programa en Java?

a) `.java`.
b) `.class`.
c) `.exe`.

Parte 3: Hola Mundo!

El programa **"¡Hola, Mundo!"** es el punto de partida clásico en cualquier lenguaje de programación. Su simplicidad lo convierte en una herramienta ideal para que los desarrolladores se familiaricen con el entorno de desarrollo, la sintaxis y los conceptos básicos de Java.

```
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println("¡Hola, Mundo!");
    }
}
```

1. Pasos para Crear y Ejecutar el Programa

1. Abrir un IDE o Editor de Texto

- Si utilizas un IDE como **IntelliJ IDEA**, **Eclipse**, o **Visual Studio Code**, crea un nuevo proyecto y selecciona Java como el lenguaje.
- Si prefieres un editor de texto simple (como **Notepad++**), guarda tu archivo con la extensión `.java`.

2. Crear el Archivo Fuente

- Nombra tu archivo como `HolaMundo.java`. Recuerda que el nombre del archivo debe coincidir exactamente con el nombre de la clase pública del programa.

3. Escribir el Código

- Copia el código anterior en tu archivo.

4. Compilar y Ejecutar

- **Desde un IDE**:
  - Haz clic en "Run" (Ejecutar) o usa atajos como `Shift + F10` en IntelliJ IDEA.
- **Desde la Terminal**:
  1. Navega al directorio donde guardaste el archivo.
  2. Compila el programa con el comando:

```
javac HolaMundo.java
```

  3. Ejecuta el programa con:

```
java HolaMundo
```

- **Salida Esperada**:

```
¡Hola, Mundo!
```

2. Explicación del Código

1. `public class HolaMundo`

- Declara una clase pública llamada `HolaMundo`. El nombre debe coincidir con el archivo fuente (`HolaMundo.java`).

2. `public static void main(String[] args)`

- Este es el método principal y el punto de entrada del programa.
  - `public`: Permite que el método sea accesible desde cualquier lugar.
  - `static`: No necesita una instancia de la clase para ejecutarse.
  - `void`: Indica que el método no devuelve ningún valor.
  - `String[] args`: Permite pasar argumentos al programa desde la línea de comandos.

3. `System.out.println("¡Hola, Mundo!");`

- Imprime el texto `"¡Hola, Mundo!"` en la consola y agrega un salto de línea al final.

3. Variaciones del Programa "¡Hola, Mundo!"

1. Imprimir Varios Mensajes

```
public class HolaMundo {
    public static void main(String[] args) {
          System.out.println("¡Hola, Mundo!");
          System.out.println("Java es genial.");
    }
}
```

Salida:

```
¡Hola, Mundo!
Java es genial.
```

2. Imprimir Sin Saltos de Línea

```
public class HolaMundo {
    public static void main(String[] args) {
        System.out.print("Hola");
        System.out.print(", Mundo!");
    }
}
```

Salida:

```
Hola, Mundo!
```

3. Imprimir Caracteres Especiales

```
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println("Bienvenido a \"Java\".");
        System.out.println("Usa \\ para imprimir una barra invertida.");
    }
}
```

Salida:

```
Bienvenido a "Java".
Usa \ para imprimir una barra invertida.
```

4. Errores Comunes y Soluciones

1. Error de Sintaxis

Problema:

```
System.out.println("¡Hola, Mundo!") // Falta el punto y coma.
```

Solución:

- Asegúrate de terminar todas las instrucciones con un punto y coma.

2. Error en el Nombre del Archivo

Problema:

- Guardar el archivo como `holaMundo.java` (con minúscula) en lugar de `HolaMundo.java`.

Solución:

- El nombre del archivo debe coincidir exactamente con el nombre de la clase pública.

3. Error en el Método `main`

Problema:

```
java
Copiar código
public static void Main(String[] args) {
    System.out.println("¡Hola, Mundo!");
}
```

Solución:

- El método debe llamarse `main` con minúsculas, ya que Java es sensible a mayúsculas y minúsculas.

5. Ejercicio Práctico

Problema

Escribe un programa que imprima el siguiente mensaje en la consola:

```
¡Hola, Estudiante!
Bienvenido al curso de programación en Java.
¡Vamos a comenzar!
```

Solución

```
public class Bienvenida {
    public static void main(String[] args) {
          System.out.println("¡Hola, Estudiante!");
          System.out.println("Bienvenido al curso de programación en Java.");
          System.out.println("¡Vamos a comenzar!");
    }
}
```

Salida:

```
¡Hola, Estudiante!
Bienvenido al curso de programación en Java.
¡Vamos a comenzar!
```

Parte 4: Conceptos fundamentales

1. Conceptos fundamentales: Variables

1. ¿Qué es una variable?

En programación, una **variable** es como un contenedor donde se almacena un dato. Este dato puede ser de distintos tipos (número, texto, etc.) y cambiar durante la ejecución del programa.

Propiedades de una variable

1. **Nombre: Identifica a la variable (como una etiqueta única).**
2. **Tipo: Determina qué clase de datos puede almacenar.**
3. **Valor: El dato que contiene en un momento dado.**

Declaración de una variable

En Java, para declarar una variable, se usa la siguiente estructura:

```
tipo nombre = valor;
```

- **tipo**: Especifica el tipo de dato (como `int`, `String`, `double`).
- **nombre**: Es el identificador de la variable (cómo la llamaremos en el código).
- **valor**: El dato que almacenará la variable (opcional en la declaración).

Ejemplo:

```
int edad = 25;
String nombre = “Ana”;
double salario = 45000.50;
```

2. Uso de una variable

Puedes realizar diferentes operaciones con una variable:

- **Asignar un valor**:

```
int numero = 10;
```

- **Modificar su valor**:

```
numero = 20;
```

-

- **Leer su valor**:

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

Ejemplo 1: Declarar y usar variables

```
public class VariablesEjemplo {
    public static void main(String[] args) {
        // Declarar variables
        int edad = 25;
        double salario = 45000.50;
        String nombre = "Ana";
        // Imprimir valores
        System.out.println("Nombre: " + nombre);
        System.out.println("Edad: " + edad);
        System.out.println("Salario: $" + salario);
        // Cambiar valores
        edad = 30;
        salario = 50000.75;
        // Imprimir nuevos valores
        System.out.println("\nDespués de la actualización:");
        System.out.println("Edad: " + edad);
        System.out.println("Salario: $" + salario);
    }
}
```

Salida:

```
Nombre: Ana
Edad: 25
Salario: $45000.5
Después de la actualización:
Edad: 30
Salario: $50000.75
```

3. Tipos de errores comunes al usar variables

1. No inicializar la variable:

```
int numero;
System.out.println(numero); // Error: la variable no está inicializada.
```

2. Usar un nombre no permitido:

```
java
Copiar código
int 2numero; // Error: el nombre no puede comenzar con un número.
```

4. Operaciones básicas con variables

Este ejemplo muestra cómo usar variables en operaciones matemáticas y cómo su valor cambia.

```
public class OperacionesConVariables {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        // Operaciones básicas
        int suma = a + b;
        int resta = b - a;
        int multiplicacion = a * b;
        int division = b / a;
        // Imprimir resultados
        System.out.println("Suma: " + suma);
        System.out.println("Resta: " + resta);
        System.out.println("Multiplicación: " + multiplicacion);
        System.out.println("División: " + division);
        // Cambiar valores y recalcular
        a = 30;
        b = 15;
        suma = a + b;
        System.out.println("\nNueva suma con a = 30 y b = 15: " + suma);
    }
}
```

Salida:

```
Suma: 30
Resta: 10
Multiplicación: 200
División: 2
Nueva suma con a = 30 y b = 15: 45
```

4. Trabajar con variables de texto

Las cadenas de texto (`String`) permiten manipular palabras o frases.

```
public class VariablesDeTexto {
    public static void main(String[] args) {
        String saludo = "Hola";
        String nombre = "Carlos";
        // Concatenar cadenas
        String mensaje = saludo + ", " + nombre + "!";
        System.out.println(mensaje);
        // Cambiar el valor de la variable
        nombre = "Laura";
        mensaje = saludo + ", " + nombre + "!";
        System.out.println(mensaje);
    }
}
```

Salida:

```
Hola, Carlos!
Hola, Laura!
```

5. Ejercicio Guiado

Objetivo: Declarar variables de diferentes tipos, realizar operaciones con ellas y mostrar los resultados en la consola.

Enunciado

1. Declara tres variables: un número entero, un número decimal y una cadena de texto.

2. Asigna valores iniciales a las variables.

3. Realiza las siguientes operaciones:

- Suma el número entero y el número decimal.
- Concadena el número entero con la cadena de texto.

4. Imprime los resultados de las operaciones.

Solución

```
public class EjercicioVariables {
    public static void main(String[] args) {
        // Declarar variables
        int entero = 10;
        double decimal = 15.5;
        String texto = "El resultado es: ";
        // Operaciones
        double suma = entero + decimal;
        String concatenacion = texto + entero;
        // Imprimir resultados
        System.out.println("Suma: " + suma);
        System.out.println(concatenacion);
    }
}
```

Salida esperada:

```
Suma: 25.5
El resultado es: 10
```

6. Ampliaciones para el alumno

1. Crea variables adicionales:

- Declara una variable `boolean` para almacenar un valor de verdad (`true` o `false`).
- Usa `System.out.println` para imprimir si el número entero es mayor que el número decimal.

2. Usa `Scanner` para entrada de datos:

- Permite que el usuario introduzca los valores de las variables.

3. Valida tipos de datos:

- Intenta usar una variable con un tipo incorrecto para ver qué errores muestra el compilador.

7. Preguntas de evaluación

1. ¿Cuál es la diferencia entre declarar una variable y asignarle un valor?

2. ¿Qué sucede si intentas usar una variable no inicializada?

3. ¿Es válido usar el mismo nombre de variable en diferentes partes del código?

2. Conceptos fundamentales: Tipos de Datos

En Java, los **tipos de datos** definen el tipo de valor que una variable puede almacenar. Estos se dividen principalmente en:

1. **Tipos primitivos: Datos básicos y de tamaño fijo.**
2. **Tipos por referencia: Representan objetos y estructuras de datos más complejas, como cadenas (`String`) y listas.**

1. Tipos Primitivos

Lista de tipos primitivos en Java

Tipo	Tamaño	Valores posibles	Ejemplo
`byte`	8 bits	-128 a 127	`byte edad = 25;`
`short`	16 bits	-32,768 a 32,767	`short año = 2023;`
`int`	32 bits	-2,147,483,648 a 2,147,483,647	`int numero = 100;`
`long`	64 bits	-9,223,372,036,854,775,808 a ...	`long distancia = 100000L;`
`float`	32 bits	Decimales simples	`float precio = 10.5f;`
`double`	64 bits	Decimales dobles	`double peso = 70.8;`
`char`	16 bits	Un carácter	`char inicial = 'A';`
`boolean`	1 bit	`true` o `false`	`boolean esMayor = true;`

Ejemplo práctico: Tipos primitivos

```
public class TiposPrimitivos {
    public static void main(String[] args) {
        byte edad = 25;
        short anio = 2023;
        int numero = 1000;
        long distancia = 100000000L;
        float precio = 49.99f;
        double peso = 70.5;
        char inicial = 'J';
        boolean esVerdadero = true;
        // Imprimir valores
        System.out.println("Edad: " + edad);
        System.out.println("Año: " + anio);
        System.out.println("Número: " + numero);
        System.out.println("Distancia: " + distancia);
        System.out.println("Precio: $" + precio);
        System.out.println("Peso: " + peso + " kg");
        System.out.println("Inicial: " + inicial);
        System.out.println("Es verdadero: " + esVerdadero);
    }
}
```

Salida:

```
Edad: 25
Año: 2023
Número: 1000
Distancia: 100000000
Precio: $49.99
Peso: 70.5 kg
Inicial: J
Es verdadero: true
```

2. Tipos por Referencia

Los **tipos por referencia** apuntan a un objeto o estructura compleja en memoria. Los más comunes son:

1. `String`: **Almacena texto.**
2. **Arrays: Almacenan múltiples valores del mismo tipo.**
3. **Clases personalizadas: Diseñadas por el programador.**

Ejemplo: Tipo `String`

```
public class TipoString {
    public static void main(String[] args) {
          String saludo = "Hola, Java!";
          String nombre = "Carlos";
          // Concatenación
          String mensaje = saludo + " Soy " + nombre + ".";
          System.out.println(mensaje);
          // Operaciones comunes
          System.out.println("Longitud del mensaje: " + mensaje.length());
          System.out.println("En mayúsculas: " + mensaje.toUpperCase());
          System.out.println("¿Empieza con 'Hola'? " + mensaje.startsWith("Ho-
la"));
    }
}
```

Salida:

```
Hola, Java! Soy Carlos.
Longitud del mensaje: 20
En mayúsculas: HOLA, JAVA! SOY CARLOS.
¿Empieza con 'Hola'? true
```

3. Conversión entre tipos

En ocasiones, es necesario convertir un tipo de dato a otro. Esto puede hacerse de dos maneras:

1. Conversión implícita (widening): Java convierte automáticamente de un tipo más pequeño a uno más grande.

2. Conversión explícita (casting): El programador indica la conversión.

Ejemplo de conversión implícita

```
public class ConversionImplicita {
    public static void main(String[] args) {
          int numero = 100;
          double decimal = numero; // Conversión implícita
          System.out.println("Entero: " + numero);
          System.out.println("Decimal: " + decimal);
    }
}
```

Salida:

```
Entero: 100
Decimal: 100.0
```

Ejemplo de conversión explícita

```
public class ConversionExplicita {
    public static void main(String[] args) {
        double precio = 49.99;
        int precioEntero = (int) precio; // Conversión explícita
        System.out.println("Precio original: $" + precio);
        System.out.println("Precio entero: $" + precioEntero);
    }
}
```

Salida:

```
Precio original: $49.99
Precio entero: $49
```

4. Comparación entre Tipos

El tipo de una variable determina:

- **El rango de valores que puede almacenar.**
- **Las operaciones que se pueden realizar.**

Ejemplo: Problema por desbordamiento

Un tipo primitivo tiene un tamaño fijo. Si intentas almacenar un valor fuera de su rango, ocurre un **desbordamiento**.

```
public class Desbordamiento {
    public static void main(String[] args) {
        byte valor = 127; // Valor máximo para byte
        valor++; // Intento de incrementar más allá del rango
        System.out.println("Valor después de desbordamiento: " + valor);
    }
}
```

Salida:

```
Valor después de desbordamiento: -128
```

5. Ejercicio Guiado

Objetivo: Practicar la declaración y uso de variables con diferentes tipos de datos.

1. Declara variables de todos los tipos primitivos (`byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`).

2. Realiza las siguientes acciones:

- Suma un número entero y un decimal.
- Convierte un decimal a un entero usando conversión explícita.
- Declara un texto (`String`) y muestra su longitud.

Solución

```
public class EjercicioTipos {
    public static void main(String[] args) {
        // Declarar variables
        byte edad = 30;
        short anio = 2023;
        int poblacion = 1000000;
        long distancia = 5000000000L;
        float temperatura = 36.5f;
        double peso = 70.8;
        char inicial = 'C';
        boolean esActivo = true;
        // Operaciones
        double suma = poblacion + peso;
        int pesoEntero = (int) peso;
        String mensaje = "Bienvenidos a Java!";
        int longitudMensaje = mensaje.length();
        // Imprimir resultados
        System.out.println("Suma de poblacion y peso: " + suma);
        System.out.println("Peso convertido a entero: " + pesoEntero);
        System.out.println("Mensaje: " + mensaje);
        System.out.println("Longitud del mensaje: " + longitudMensaje);
    }
}
```

Salida esperada:

```
Suma de poblacion y peso: 1000070.8
Peso convertido a entero: 70
Mensaje: Bienvenidos a Java!
Longitud del mensaje: 19
```

6. Preguntas de Evaluación

1. ¿Qué diferencia hay entre los tipos primitivos `float` y `double`?

2. ¿Qué sucede si intentas almacenar un número decimal en una variable `int` sin usar conversión explícita?

3. Explica la diferencia entre una conversión implícita y una explícita. ¿Cuándo se usa cada una?

3.Conceptos fundamentales: Clases

En Java, una **clase** es el modelo o plantilla para crear objetos. Define el **estado** (atributos) y el **comportamiento** (métodos) que los objetos tendrán. Es uno de los conceptos fundamentales de la programación orientada a objetos.

1. ¿Qué es una Clase?

Una **clase** puede entenderse como una definición o estructura que describe un conjunto de objetos que comparten características comunes.

Sintaxis básica de una clase

```
java
Copiar código
public class NombreClase {
    // Atributos: características del objeto
    tipo nombreAtributo;
    // Métodos: acciones que realiza el objeto
    tipo nombreMetodo(parámetros) {
         // Código
    }
}
```

2. Crear y usar clases

Ejemplo 1: Clase simple

```
public class Persona {
    // Atributos
    String nombre;
    int edad;
    // Métodos
    public void mostrarInformacion() {
        System.out.println("Nombre: " + nombre + ", Edad: " + edad);
    }
}
```

Clase principal para usar la clase `Persona`:

```
public class Main {
    public static void main(String[] args) {
        // Crear un objeto de tipo Persona
        Persona persona1 = new Persona();
        // Asignar valores a los atributos
        persona1.nombre = "Carlos";
        persona1.edad = 30;
        // Llamar a un método
        persona1.mostrarInformacion();
    }
}
```

Salida:

```
Nombre: Carlos, Edad: 30
```

3. Constructores

Un **constructor** es un método especial que se utiliza para inicializar un objeto al momento de crearlo. Tiene el mismo nombre que la clase y no tiene tipo de retorno.

Ejemplo: Uso de constructores

```
public class Persona {
    // Atributos
    String nombre;
    int edad;
    // Constructor
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
    // Método para mostrar información
    public void mostrarInformacion() {
        System.out.println("Nombre: " + nombre + ", Edad: " + edad);
    }
}
```

Clase principal:

```
public class Main {
    public static void main(String[] args) {
        // Crear un objeto utilizando el constructor
        Persona persona1 = new Persona("Laura", 25);
        // Llamar al método para mostrar información
        persona1.mostrarInformacion();
    }
}
```

Salida:

```
Nombre: Laura, Edad: 25
```

4. Métodos Getters y Setters

Los **getters** y **setters** son métodos que permiten acceder y modificar los atributos de un objeto, respetando el principio de **encapsulación**.

Ejemplo: Implementación de getters y setters

```
public class Persona {
    private String nombre; // Atributo privado
    private int edad;
    // Constructor
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
    // Getter para obtener el nombre
    public String getNombre() {
        return nombre;
    }
    // Setter para cambiar el nombre
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    // Getter para obtener la edad
    public int getEdad() {
        return edad;
    }
    // Setter para cambiar la edad
    public void setEdad(int edad) {
        if (edad > 0) {
            this.edad = edad;
        } else {
            System.out.println("La edad debe ser positiva.");
        }
    }
    // Método para mostrar información
    public void mostrarInformacion() {
        System.out.println("Nombre: " + nombre + ", Edad: " + edad);
    }
}
```

Clase principal:

```
public class Main {
    public static void main(String[] args) {
        // Crear un objeto de tipo Persona
        Persona persona1 = new Persona("Ana", 20);
        // Mostrar información inicial
        persona1.mostrarInformacion();
        // Usar setters para modificar atributos
        persona1.setNombre("Carlos");
        persona1.setEdad(30);
        // Mostrar información actualizada
        persona1.mostrarInformacion();
    }
}
```

Salida:

```
Nombre: Ana, Edad: 20
Nombre: Carlos, Edad: 30
```

5. Relaciones entre Clases

En la programación orientada a objetos, las clases pueden interactuar entre sí. Esto se logra mediante **relaciones** como **composición** (una clase contiene objetos de otra clase).

Ejemplo: Composición

Clase `Direccion`**:**

```
public class Direccion {
    String calle;
    String ciudad;
    // Constructor
    public Direccion(String calle, String ciudad) {
        this.calle = calle;
        this.ciudad = ciudad;
    }
    public void mostrarDireccion() {
        System.out.println("Calle: " + calle + ", Ciudad: " + ciudad);
    }
}
```

Clase `Persona`:

```
public class Persona {
    String nombre;
    int edad;
    Direccion direccion; // Composición
    // Constructor
    public Persona(String nombre, int edad, Direccion direccion) {
        this.nombre = nombre;
        this.edad = edad;
        this.direccion = direccion;
    }
    // Método para mostrar información
    public void mostrarInformacion() {
        System.out.println("Nombre: " + nombre + ", Edad: " + edad);
        direccion.mostrarDireccion(); // Delegar en el método de Direccion
    }
}
```

Clase principal:

```
public class Main {
    public static void main(String[] args) {
        // Crear un objeto Direccion
        Direccion direccion1 = new Direccion("Av. Principal", "Madrid");
        // Crear un objeto Persona con la dirección
        Persona persona1 = new Persona("Laura", 28, direccion1);
        // Mostrar información
        persona1.mostrarInformacion();
    }
}
```

Salida:

```
Nombre: Laura, Edad: 28
Calle: Av. Principal, Ciudad: Madrid
```

6. Ejercicio Guiado

Objetivo: Crear una clase `Coche` con atributos y métodos, y una clase principal para interactuar con ella.

1. Crea la clase `Coche` con los siguientes atributos:

- `marca` (tipo `String`)
- `modelo` (tipo `String`)
- `velocidad` (tipo `int`).

2. Añade métodos:

- `acelerar(int incremento)`: Incrementa la velocidad.
- `frenar()`: Establece la velocidad en 0.
- `mostrarEstado()`: Imprime la marca, modelo y velocidad.

3. En la clase principal:

- Crea un objeto de tipo `Coche`.
- Usa los métodos para cambiar y mostrar el estado del coche.

Solución

Clase `Coche`:

```
  public class Coche {
      private String marca;
      private String modelo;
      private int velocidad;
      // Constructor
      public Coche(String marca, String modelo) {
            this.marca = marca;
            this.modelo = modelo;
            this.velocidad = 0; // Inicialmente está detenido
      }
      // Métodos
      public void acelerar(int incremento) {
            velocidad += incremento;
            System.out.println("El coche ha acelerado. Velocidad actual: " + ve-
locidad + " km/h.");
      }
      public void frenar() {
            velocidad = 0;
            System.out.println("El coche se ha detenido.");
      }
      public void mostrarEstado() {
            System.out.println("Marca: " + marca + ", Modelo: " + modelo + ",
Velocidad: " + velocidad + " km/h.");
      }
  }
```

Clase principal:

```
public class Main {
    public static void main(String[] args) {
          Coche coche1 = new Coche(“Toyota”, “Corolla”);
          coche1.mostrarEstado();
          coche1.acelerar(50);
          coche1.mostrarEstado();
          coche1.frenar();
          coche1.mostrarEstado();
    }
}
```

Salida esperada:

```
Marca: Toyota, Modelo: Corolla, Velocidad: 0 km/h.
El coche ha acelerado. Velocidad actual: 50 km/h.
Marca: Toyota, Modelo: Corolla, Velocidad: 50 km/h.
El coche se ha detenido.
Marca: Toyota, Modelo: Corolla, Velocidad: 0 km/h.
```

7. Preguntas de Evaluación

1. ¿Qué diferencia hay entre un atributo y un método?

2. ¿Por qué es importante usar constructores en las clases?

4.Conceptos fundamentales: Métodos

Un **método** es un bloque de código que realiza una tarea específica. Los métodos son fundamentales en Java porque permiten **organizar** el código, **reutilizarlo** y modelar el comportamiento de los objetos.

1. ¿Qué es un método?

Un método en Java:

1. Realiza una acción o calcula un valor.

2. Pertenece a una clase u objeto.

3. Puede aceptar parámetros y devolver un resultado.

Estructura básica de un método

```
tipoDeRetorno nombreDelMetodo(parámetros) {
    // Código del método
    return valor; // (si el método devuelve un resultado)
}
```

- `tipoDeRetorno`: Especifica el tipo de dato que devuelve el método (por ejemplo, `int`, `String`, `void` si no devuelve nada).
- `nombreDelMetodo`: El identificador del método.
- `parámetros`: Valores que el método puede recibir para realizar su tarea (opcional).
- `return`: Devuelve un valor cuando el método finaliza (opcional).

2. Métodos sin parámetros

Un método sin parámetros realiza una tarea que no necesita información adicional para ejecutarse.

Ejemplo: Método para saludar

```
public class Persona {
    String nombre;
    // Método sin parámetros
    public void saludar() {
        System.out.println("Hola, soy " + nombre);
    }
}
```

Clase principal:

```
public class Main {
    public static void main(String[] args) {
        Persona persona1 = new Persona();
        persona1.nombre = "Carlos";
        // Llamar al método saludar
        persona1.saludar();
    }
}
```

Salida:

```
Hola, soy Carlos
```

3. Métodos con parámetros

Un método con parámetros recibe valores para utilizarlos en su lógica.

Ejemplo: Sumar dos números

```
  public class Calculadora {
      // Método con parámetros
      public void sumar(int a, int b) {
            int resultado = a + b;
            System.out.println("La suma de " + a + " y " + b + " es " + resulta-
do);
      }
  }
```

Clase principal:

```
  public class Main {
      public static void main(String[] args) {
            Calculadora calc = new Calculadora();
            // Llamar al método sumar con diferentes valores
            calc.sumar(10, 20);
            calc.sumar(5, 7);
      }
  }
```

Salida:

```
  La suma de 10 y 20 es 30
  La suma de 5 y 7 es 12
```

4. Métodos con valor de retorno

Un método puede devolver un resultado utilizando la palabra clave `return`.

Ejemplo: Calcular el área de un rectángulo

```
  public class Rectangulo {
      // Método que devuelve un valor
      public int calcularArea(int base, int altura) {
            return base * altura;
      }
  }
```

Clase principal:

```
public class Main {
    public static void main(String[] args) {
        Rectangulo rect = new Rectangulo();
        // Calcular áreas con diferentes valores
        int area1 = rect.calcularArea(5, 10);
        int area2 = rect.calcularArea(7, 3);
        // Imprimir resultados
        System.out.println("Área 1: " + area1);
        System.out.println("Área 2: " + area2);
    }
}
```

Salida:

```
Área 1: 50
Área 2: 21
```

5. Sobrecarga de métodos

La **sobrecarga de métodos** permite definir varios métodos con el mismo nombre, pero diferentes parámetros (tipo o cantidad). Esto mejora la flexibilidad del código.

Ejemplo: Calcular el área de formas geométricas

```
public class CalculadoraDeAreas {
    // Área de un cuadrado
    public int calcularArea(int lado) {
        return lado * lado;
    }
    // Área de un rectángulo
    public int calcularArea(int base, int altura) {
        return base * altura;
    }
    // Área de un círculo
    public double calcularArea(double radio) {
        return Math.PI * radio * radio;
    }
}
```

Clase principal:

```
public class Main {
    public static void main(String[] args) {
        CalculadoraDeAreas calc = new CalculadoraDeAreas();
        // Llamar a métodos sobrecargados
        System.out.println("Área del cuadrado: " + calc.calcularArea(4));
        System.out.println("Área del rectángulo: " + calc.calcularArea(5,
10));
        System.out.println("Área del círculo: " + calc.calcularArea(7.5));
    }
}
```

Salida:

```
Área del cuadrado: 16
Área del rectángulo: 50
Área del círculo: 176.71458676442586
```

6. Métodos estáticos

Un método **estático** pertenece a la clase en lugar de a un objeto. Se puede llamar directamente usando el nombre de la clase.

Ejemplo: Método estático para sumar números

```
public class Utilidades {
    public static int sumar(int a, int b) {
        return a + b;
    }
}
```

Clase principal:

```
public class Main {
    public static void main(String[] args) {
        // Llamar al método estático sin crear un objeto
        int resultado = Utilidades.sumar(15, 25);
        System.out.println("Resultado: " + resultado);
    }
}
```

Salida:

```
Resultado: 40
```

7. Ejercicio Guiado

Objetivo: Crear una clase `Coche` con métodos para manejar su comportamiento.

1. Crea la clase `Coche` con los atributos:

- `marca` (tipo `String`).
- `velocidad` (tipo `int`).

2. Añade los siguientes métodos:

- `acelerar(int incremento)`: Incrementa la velocidad en el valor indicado.
- `frenar()`: Reduce la velocidad a 0.
- `mostrarEstado()`: Muestra la marca y la velocidad actual.

3. En la clase principal:

- Crea un objeto `Coche`.
- Usa los métodos para cambiar y mostrar su estado.

Solución

Clase `Coche`:

```
public class Coche {
    private String marca;
    private int velocidad;
    // Constructor
    public Coche(String marca) {
        this.marca = marca;
        this.velocidad = 0; // Inicialmente está detenido
    }
    // Métodos
    public void acelerar(int incremento) {
        velocidad += incremento;
        System.out.println("El coche ha acelerado. Velocidad actual: " + velocidad + " km/h.");
    }
    public void frenar() {
        velocidad = 0;
        System.out.println("El coche se ha detenido.");
    }
    public void mostrarEstado() {
        System.out.println("Marca: " + marca + ", Velocidad: " + velocidad + " km/h.");
    }
}
```

Clase principal:

```
public class Main {
    public static void main(String[] args) {
        Coche coche1 = new Coche("Toyota");
        coche1.mostrarEstado();
        coche1.acelerar(50);
        coche1.mostrarEstado();
        coche1.frenar();
        coche1.mostrarEstado();
    }
}
```

Salida esperada:

```
Marca: Toyota, Velocidad: 0 km/h.
El coche ha acelerado. Velocidad actual: 50 km/h.
Marca: Toyota, Velocidad: 50 km/h.
El coche se ha detenido.
Marca: Toyota, Velocidad: 0 km/h.
```

8. Preguntas de Evaluación

1. ¿Cuál es la diferencia entre un método estático y uno no estático?

2. ¿Qué es la sobrecarga de métodos y cuándo se utiliza?

3. ¿Por qué es útil devolver valores en los métodos?

5. Conceptos fundamentales: Condicionales

Los **condicionales** son estructuras fundamentales que permiten que un programa tome decisiones basadas en ciertas condiciones. Son esenciales para controlar el flujo de ejecución y dotar al programa de lógica.

1. ¿Qué son los condicionales?

Un condicional evalúa una **expresión lógica** (verdadero o falso) y ejecuta diferentes bloques de código según el resultado.

2. Tipos de condicionales en Java

1. `if`: Ejecuta un bloque de código si la condición es verdadera.

2. `if-else`: Proporciona una alternativa si la condición es falsa.

3. `if-else if`: Permite evaluar múltiples condiciones.

4. `switch`: Evalúa una expresión y ejecuta el caso que coincida.

3. Uso de `if`

```
if (condicion) {
    // Código a ejecutar si la condición es verdadera
}
```

Ejemplo: Verificar si un número es positivo

```
public class CondicionalIf {
    public static void main(String[] args) {
        int numero = 5;
        if (numero > 0) {
            System.out.println("El número es positivo.");
        }
    }
}
```

Salida:

```
El número es positivo.
```

4. Uso de `if-else`

```
if (condicion) {
    // Código a ejecutar si la condición es verdadera
} else {
    // Código a ejecutar si la condición es falsa
}
```

Ejemplo: Verificar si un número es positivo o negativo

```
public class CondicionalIfElse {
    public static void main(String[] args) {
        int numero = -3;
        if (numero > 0) {
            System.out.println("El número es positivo.");
        } else {
            System.out.println("El número es negativo.");
        }
    }
}
```

Salida:

```
El número es negativo.
```

5. Uso de `if-else if`

```
if (condicion1) {
    // Código si la condicion1 es verdadera
} else if (condicion2) {
    // Código si la condicion2 es verdadera
} else {
    // Código si ninguna condición es verdadera
}
```

Ejemplo: Clasificar un número

```
public class CondicionalIfElseIf {
    public static void main(String[] args) {
          int numero = 0;
          if (numero > 0) {
                System.out.println("El número es positivo.");
          } else if (numero < 0) {
                System.out.println("El número es negativo.");
          } else {
                System.out.println("El número es cero.");
          }
    }
}
```

Salida:

```
El número es cero.
```

6. Uso de `switch`

El `switch` es útil cuando se evalúan múltiples valores posibles de una expresión. Es más legible que una serie de `if-else if` en algunos casos.

```
switch (variable) {
    case valor1:
          // Código si variable == valor1
          break;
    case valor2:
          // Código si variable == valor2
          break;
    default:
          // Código si ninguna de las opciones anteriores coincide
}
```

Ejemplo: Mostrar el día de la semana

```
public class CondicionalSwitch {
    public static void main(String[] args) {
        int dia = 3;
        switch (dia) {
            case 1:
                System.out.println("Lunes");
                break;
            case 2:
                System.out.println("Martes");
                break;
            case 3:
                System.out.println("Miércoles");
                break;
            case 4:
                System.out.println("Jueves");
                break;
            case 5:
                System.out.println("Viernes");
                break;
            default:
                System.out.println("Día no válido.");
        }
    }
}
```

Salida:

```
Miércoles
```

7. Operador Ternario

El operador **ternario** es una forma simplificada de escribir una condición `if-else`.

```
variable = (condicion) ? valorSiVerdadero : valorSiFalso;
```

Ejemplo: Determinar si un número es par o impar

```
public class OperadorTernario {
    public static void main(String[] args) {
        int numero = 7;
        String resultado = (numero % 2 == 0) ? “Par” : “Impar”;
        System.out.println(“El número es “ + resultado);
    }
}
```

Salida:

```
El número es Impar
```

8. Ejercicios Guiados

Clasificar edades

Escribe un programa que reciba la edad de una persona y clasifique su etapa de vida:

- Menor de 13: “Niño”.
- Entre 13 y 18: “Adolescente”.
- Más de 18: “Adulto”.

Solución:

```
import java.util.Scanner;
public class ClasificarEdades {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print(“Introduce tu edad: “);
        int edad = scanner.nextInt();
        if (edad < 13) {
            System.out.println(“Eres un niño.”);
        } else if (edad >= 13 && edad <= 18) {
            System.out.println(“Eres un adolescente.”);
        } else {
            System.out.println(“Eres un adulto.”);
        }
        scanner.close();
    }
}
```

Salida esperada (interacción):

```
Introduce tu edad: 15
Eres un adolescente.
```

Calcular el precio con descuento

Escribe un programa que calcule el precio final de un producto después de aplicar un descuento basado en su precio inicial:

- Precio menor a $100: 5% de descuento.
- Precio entre $100 y $500: 10% de descuento.
- Precio mayor a $500: 15% de descuento.

Solución:

```
import java.util.Scanner;
public class CalcularDescuento {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Introduce el precio del producto: ");
        double precio = scanner.nextDouble();
        double descuento = 0;
        if (precio < 100) {
            descuento = 0.05;
        } else if (precio <= 500) {
            descuento = 0.10;
        } else {
            descuento = 0.15;
        }
        double precioFinal = precio - (precio * descuento);
        System.out.println("Precio inicial: $" + precio);
        System.out.println("Descuento aplicado: " + (descuento * 100) +
"%");
        System.out.println("Precio final: $" + precioFinal);
        scanner.close();
    }
}
```

Salida esperada (interacción):

```
Introduce el precio del producto: 300
Precio inicial: $300.0
Descuento aplicado: 10.0%
Precio final: $270.0
```

9. Preguntas de Evaluación

1. ¿Cuál es la diferencia entre un `if-else` y un `switch`?
2. ¿Qué sucede si se olvida el `break` en una estructura `switch`?
3. ¿En qué situaciones usarías un operador ternario en lugar de un `if-else`?

6. Conceptos fundamentales: Arrays

Un **array** es una estructura de datos que permite almacenar **múltiples valores del mismo tipo** en una sola variable. Los arrays son fundamentales para manejar colecciones de datos de tamaño fijo.

1. Características de los Arrays

1. Homogeneidad: Todos los elementos deben ser del mismo tipo.
2. Tamaño fijo: El tamaño del array debe definirse al momento de su creación y no puede cambiar.
3. Índices: Cada elemento del array tiene un índice, comenzando desde `0`.

2. Declaración de un Array

Un array puede declararse de la siguiente manera:

```
tipo[] nombre = new tipo[tamaño];
```

- `tipo`: Especifica el tipo de dato que almacenará el array.
- `nombre`: Es el identificador del array.
- `tamaño`: Indica cuántos elementos puede contener el array.

Ejemplo: Declarar y usar un array

```
public class EjemploArray {
    public static void main(String[] args) {
        // Declarar y asignar valores
        int[] numeros = new int[5];
        numeros[0] = 10;
        numeros[1] = 20;
        numeros[2] = 30;
        numeros[3] = 40;
        numeros[4] = 50;
        // Acceder a elementos por índice
        System.out.println("Primer número: " + numeros[0]);
        System.out.println("Último número: " + numeros[4]);
    }
}
```

Salida:

```
Primer número: 10
Último número: 50
```

3. Inicialización de Arrays

Inicialización directa

Puedes inicializar un array con valores al declararlo:

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

Inicialización vacía

Define un array vacío y luego asigna valores:

```
String[] nombres = new String[3];
nombres[0] = "Ana";
nombres[1] = "Luis";
nombres[2] = "Carlos";
```

4. Recorrer un Array

Para trabajar con todos los elementos de un array, puedes usar bucles como `for` o `foreach`.

Usar un bucle `for`

```
public class RecorrerArray {
    public static void main(String[] args) {
        int[] numeros = {10, 20, 30, 40, 50};
        for (int i = 0; i < numeros.length; i++) {
            System.out.println("Elemento en índice " + i + ": " + numeros[i]);
        }
    }
}
```

Salida:

```
Elemento en índice 0: 10
Elemento en índice 1: 20
Elemento en índice 2: 30
Elemento en índice 3: 40
Elemento en índice 4: 50
```

Usar un bucle `foreach`

El bucle `foreach` simplifica el recorrido de un array.

```
public class RecorrerArrayForeach {
    public static void main(String[] args) {
        String[] nombres = {"Ana", "Luis", "Carlos"};
        for (String nombre : nombres) {
            System.out.println("Nombre: " + nombre);
        }
    }
}
```

Salida:

```
Nombre: Ana
Nombre: Luis
Nombre: Carlos
```

5. Arrays Multidimensionales

Un array multidimensional almacena datos en varias dimensiones, como tablas o matrices.

Ejemplo: Declarar y usar una matriz

```
public class Matriz {
    public static void main(String[] args) {
        // Declarar y asignar valores
        int[][] matriz = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };
        // Acceder a elementos
        System.out.println("Elemento en (0,0): " + matriz[0][0]);
        System.out.println("Elemento en (2,2): " + matriz[2][2]);
        // Recorrer la matriz
        for (int i = 0; i < matriz.length; i++) {
            for (int j = 0; j < matriz[i].length; j++) {
                System.out.print(matriz[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Salida:

```
Elemento en (0,0): 1
Elemento en (2,2): 9
1 2 3
4 5 6
7 8 9
```

6. Ejercicios Guiados

Calcular la suma de los elementos de un array

Escribe un programa que almacene cinco números en un array, los recorra con un bucle y calcule su suma.

Solución:

```
public class SumaArray {
    public static void main(String[] args) {
          int[] numeros = {10, 20, 30, 40, 50};
          int suma = 0;
          // Recorrer y sumar
          for (int numero : numeros) {
                suma += numero;
          }
          System.out.println("La suma de los números es: " + suma);
    }
}
```

Salida:

```
La suma de los números es: 150
```

Buscar un elemento en un array

Escribe un programa que permita buscar un número en un array y determine si existe.

Solución:

```
import java.util.Scanner;
public class BuscarEnArray {
    public static void main(String[] args) {
        int[] numeros = {10, 20, 30, 40, 50};
        Scanner scanner = new Scanner(System.in);
        System.out.print("Introduce el número a buscar: ");
        int buscar = scanner.nextInt();
        boolean encontrado = false;
        for (int numero : numeros) {
            if (numero == buscar) {
                encontrado = true;
                break;
            }
        }
        if (encontrado) {
            System.out.println("El número " + buscar + " está en el ar-
ray.");
        } else {
            System.out.println("El número " + buscar + " no está en el ar-
ray.");
        }
        scanner.close();
    }
}
```

Salida esperada:

```
Introduce el número a buscar: 30
El número 30 está en el array.
```

Matriz de notas

Crea un programa que almacene las notas de 3 estudiantes en 3 asignaturas, calcule el promedio de cada estudiante y lo muestre.

Solución:

```
public class MatrizNotas {
    public static void main(String[] args) {
        double[][] notas = {
            {8.5, 7.0, 9.0}, // Estudiante 1
            {6.0, 5.5, 7.5}, // Estudiante 2
            {9.0, 8.5, 8.0}  // Estudiante 3
        };
        for (int i = 0; i < notas.length; i++) {
            double suma = 0;
            for (int j = 0; j < notas[i].length; j++) {
                suma += notas[i][j];
            }
            double promedio = suma / notas[i].length;
            System.out.println("Promedio del estudiante " + (i + 1) + ": "
+ promedio);
        }
    }
}
```

Salida:

```
Promedio del estudiante 1: 8.166666666666666
Promedio del estudiante 2: 6.333333333333333
Promedio del estudiante 3: 8.5
```

7. Preguntas de Evaluación

1. ¿Qué sucede si intentas acceder a un índice fuera del rango de un array?

2. ¿Cómo se recorren las filas y columnas de un array bidimensional?

3. ¿Qué diferencia hay entre un array de una dimensión y un array multidimensional?

7. Conceptos fundamentales: Bucles

Los **bucles** son estructuras que permiten ejecutar un bloque de código repetidamente, ya sea un número determinado de veces o mientras una condición sea verdadera. Son fundamentales para manejar tareas repetitivas y procesar colecciones de datos como arrays.

1. Tipos de Bucles en Java

En Java, los bucles más comunes son:

1. `for`: **Repite un bloque de código un número fijo de veces.**
2. `while`: **Repite un bloque de código mientras una condición sea verdadera.**
3. `do-while`: **Similar al `while`, pero garantiza que el código se ejecuta al menos una vez.**
4. **Bucle `for-each`: Diseñado para recorrer colecciones como arrays.**

2. Bucle `for`

El bucle `for` es ideal cuando conoces de antemano cuántas veces necesitas repetir el código.

```
for (inicialización; condición; actualización) {
    // Código a ejecutar
}
```

- **Inicialización**: Declara y asigna la variable de control.
- **Condición**: Determina si se sigue ejecutando el bucle.
- **Actualización**: Modifica la variable de control.

Ejemplo: Contar hasta 5

```
public class BucleFor {
    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Número: " + i);
        }
    }
}
```

Salida:

```
Número: 1
Número: 2
Número: 3
Número: 4
Número: 5
```

3. Bucle `while`

El bucle `while` ejecuta un bloque de código mientras una condición sea verdadera. Es útil cuando no conoces de antemano cuántas veces debe ejecutarse.

```
while (condición) {
    // Código a ejecutar
}
```

Imprimir números hasta que lleguen a 10

```
public class BucleWhile {
    public static void main(String[] args) {
          int numero = 1;
          while (numero <= 10) {
                System.out.println("Número: " + numero);
                numero++; // Incrementar el valor
          }
    }
}
```

Salida:

```
Número: 1
Número: 2
Número: 3
...
Número: 10
```

4. Bucle `do-while`

El bucle `do-while` es similar al `while`, pero garantiza que el código se ejecuta **al menos una vez**.

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

Solicitar un número positivo

```
java
Copiar código
import java.util.Scanner;
public class BucleDoWhile {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int numero;
        do {
            System.out.print("Introduce un número positivo: ");
            numero = scanner.nextInt();
        } while (numero <= 0);
        System.out.println("¡Gracias! Número ingresado: " + numero);
        scanner.close();
    }
}
```

Salida esperada (interacción):

```
Introduce un número positivo: -3
Introduce un número positivo: 0
Introduce un número positivo: 5
¡Gracias! Número ingresado: 5
```

5. Bucle `for-each`

El bucle `for-each` simplifica el recorrido de colecciones como arrays. No requiere manejar índices explícitamente.

Ejemplo: Recorrer un array de nombres

```
public class BucleForEach {
    public static void main(String[] args) {
        String[] nombres = {"Ana", "Luis", "Carlos"};
        for (String nombre : nombres) {
            System.out.println("Nombre: " + nombre);
        }
    }
}
```

Salida:

```
Nombre: Ana
Nombre: Luis
Nombre: Carlos
```

6. Control de Bucles: `break` y `continue`

`break`

Detiene el bucle de inmediato, sin importar si la condición sigue siendo verdadera.

Ejemplo: Detener al encontrar un número mayor a 50

```
public class BreakEjemplo {
    public static void main(String[] args) {
        int[] numeros = {10, 20, 50, 70, 90};
        for (int numero : numeros) {
            if (numero > 50) {
                System.out.println("Número mayor a 50 encontrado: " + numero);
                break;
            }
            System.out.println("Número: " + numero);
        }
    }
}
```

Salida:

```
Número: 10
Número: 20
Número mayor a 50 encontrado: 70
```

`continue`

Salta el resto del código en la iteración actual y pasa a la siguiente.

Ejemplo: Ignorar números pares

```
public class ContinueEjemplo {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            if (i % 2 == 0) {
                continue; // Saltar números pares
            }
            System.out.println("Número impar: " + i);
        }
    }
}
```

Salida:

```
Número impar: 1
Número impar: 3
Número impar: 5
...
Número impar: 9
```

7. Ejercicios Guiados

Tablas de multiplicar

Escribe un programa que imprima la tabla de multiplicar del número ingresado por el usuario.

Solución:

```
import java.util.Scanner;
public class TablaMultiplicar {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Introduce un número para su tabla de multiplicar: ");
        int numero = scanner.nextInt();
        for (int i = 1; i <= 10; i++) {
            System.out.println(numero + " x " + i + " = " + (numero * i));
        }
        scanner.close();
    }
}
```

Salida esperada:

```
Introduce un número para su tabla de multiplicar: 5
5 x 1 = 5
5 x 2 = 10
...
5 x 10 = 50
```

Sumar números ingresados

Escribe un programa que permita al usuario ingresar números repetidamente. El programa debe sumar los números ingresados y detenerse si se ingresa un número negativo.

Solución:

```
import java.util.Scanner;
public class SumarNumeros {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int suma = 0;
        int numero;
        do {
            System.out.print("Introduce un número (negativo para salir): ");
            numero = scanner.nextInt();
            if (numero >= 0) {
                suma += numero;
            }
        } while (numero >= 0);
        System.out.println("La suma total es: " + suma);
        scanner.close();
    }
}
```

Salida esperada:

```
Introduce un número (negativo para salir): 10
Introduce un número (negativo para salir): 20
Introduce un número (negativo para salir): -1
La suma total es: 30
```

Números primos

Escribe un programa que imprima los números primos entre 1 y 50.

Solución:

```
public class NumerosPrimos {
    public static void main(String[] args) {
        for (int numero = 2; numero <= 50; numero++) {
            boolean esPrimo = true;
            for (int divisor = 2; divisor <= numero / 2; divisor++) {
                if (numero % divisor == 0) {
                    esPrimo = false;
                    break;
                }
            }
            if (esPrimo) {
                System.out.println(numero + " es primo.");
            }
        }
    }
}
```

Salida esperada:

```
2 es primo.
3 es primo.
5 es primo.
...
47 es primo.
```

8. Preguntas de Evaluación

1. **¿Cuál es la diferencia entre un bucle `for` y un bucle `while`?**
2. **¿En qué casos usarías un bucle `do-while` en lugar de un `while`?**
3. **¿Qué sucede si se olvida el `break` dentro de un bucle infinito?**

8. Conceptos fundamentales: Ejercicios Combinados

Ejercicio 1: Sistema de Gestión de Estudiantes

Objetivo:

Crear un sistema que almacene información de estudiantes, permita calcular su promedio y determine si aprobaron o no.

Enunciado:

1. Crea una clase `Estudiante` con los siguientes atributos:

- `nombre` (tipo `String`).
- `notas` (array de tipo `double` de tamaño 3).

2. Implementa los métodos:

- `calcularPromedio()`: Devuelve el promedio de las notas.
- `mostrarInformacion()`: Muestra el nombre, las notas y el promedio.
- `aprobo()`: Devuelve `true` si el promedio es mayor o igual a 6.

3. En la clase principal:

- Crea un array para almacenar hasta 3 estudiantes.
- Llena la información de cada estudiante.
- Muestra la información y si aprobó o no.

Solución:

Clase `Estudiante`**:**

```
public class Estudiante {
    private String nombre;
    private double[] notas;
    // Constructor
    public Estudiante(String nombre, double[] notas) {
        this.nombre = nombre;
        this.notas = notas;
    }
    // Método para calcular el promedio
    public double calcularPromedio() {
        double suma = 0;
        for (double nota : notas) {
            suma += nota;
        }
        return suma / notas.length;
    }
    // Método para mostrar información
    public void mostrarInformacion() {
        System.out.println("Nombre: " + nombre);
        System.out.print("Notas: ");
        for (double nota : notas) {
            System.out.print(nota + " ");
        }
        System.out.println("\nPromedio: " + calcularPromedio());
    }
    // Método para verificar si aprobó
    public boolean aprobo() {
        return calcularPromedio() >= 6;
    }
}
```

Clase principal:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Estudiante[] estudiantes = new Estudiante[3];
        // Llenar información de los estudiantes
        for (int i = 0; i < estudiantes.length; i++) {
            System.out.print("Introduce el nombre del estudiante " + (i + 1) + ": ");
            String nombre = scanner.nextLine();
            double[] notas = new double[3];
            for (int j = 0; j < notas.length; j++) {
                System.out.print("Introduce la nota " + (j + 1) + " de " + nombre + ": ");
                notas[j] = scanner.nextDouble();
            }
            scanner.nextLine(); // Consumir salto de línea
            estudiantes[i] = new Estudiante(nombre, notas);
        }
        // Mostrar información de los estudiantes
        System.out.println("\n--- Información de los Estudiantes ---");
        for (Estudiante estudiante : estudiantes) {
            estudiante.mostrarInformacion();
            System.out.println(estudiante.aprobo() ? "Resultado: Aprobado\n" : "Resultado: Reprobado\n");
        }
        scanner.close();
    }
}
```

Salida esperada (interacción):

```
Introduce el nombre del estudiante 1: Ana
Introduce la nota 1 de Ana: 8.0
Introduce la nota 2 de Ana: 9.0
Introduce la nota 3 de Ana: 7.5
Introduce el nombre del estudiante 2: Carlos
Introduce la nota 1 de Carlos: 4.0
Introduce la nota 2 de Carlos: 5.5
Introduce la nota 3 de Carlos: 6.0
Introduce el nombre del estudiante 3: Laura
Introduce la nota 1 de Laura: 9.0
Introduce la nota 2 de Laura: 9.5
Introduce la nota 3 de Laura: 8.5
--- Información de los Estudiantes ---
Nombre: Ana
Notas: 8.0 9.0 7.5
Promedio: 8.166666666666666
Resultado: Aprobado
Nombre: Carlos
Notas: 4.0 5.5 6.0
Promedio: 5.166666666666667
Resultado: Reprobado
Nombre: Laura
Notas: 9.0 9.5 8.5
Promedio: 9.0
Resultado: Aprobado
```

Ejercicio 2: Gestión de Inventario

Objetivo:

Desarrollar un sistema básico para gestionar un inventario de productos.

Enunciado:

1. Crea una clase `Producto` con:

- `nombre` (tipo `String`).
- `precio` (tipo `double`).
- `cantidad` (tipo `int`).

2. Implementa los métodos:

- `calcularTotal()`: Devuelve el total de dinero disponible en stock (`precio * cantidad`).
- `mostrarProducto()`: Muestra la información del producto.

3. En la clase principal:

- Crea un array de productos.
- Llena su información (nombre, precio, cantidad).
- Calcula y muestra el total del inventario.

Solución:

Clase `Producto`:

```
public class Producto {
    private String nombre;
    private double precio;
    private int cantidad;
    // Constructor
    public Producto(String nombre, double precio, int cantidad) {
        this.nombre = nombre;
        this.precio = precio;
        this.cantidad = cantidad;
    }
    // Método para calcular el total del producto
    public double calcularTotal() {
        return precio * cantidad;
    }
    // Método para mostrar información del producto
    public void mostrarProducto() {
        System.out.println("Nombre: " + nombre);
        System.out.println("Precio: $" + precio);
        System.out.println("Cantidad: " + cantidad);
        System.out.println("Total: $" + calcularTotal());
    }
}
```

Clase principal:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Producto[] inventario = new Producto[3];
        // Llenar información del inventario
        for (int i = 0; i < inventario.length; i++) {
            System.out.print("Introduce el nombre del producto " + (i + 1)
+ ": ");
            String nombre = scanner.nextLine();
            System.out.print("Introduce el precio de " + nombre + ": ");
            double precio = scanner.nextDouble();
            System.out.print("Introduce la cantidad de " + nombre + ": ");
            int cantidad = scanner.nextInt();
            scanner.nextLine(); // Consumir salto de línea
            inventario[i] = new Producto(nombre, precio, cantidad);
        }
        // Mostrar información del inventario
        double totalInventario = 0;
        System.out.println("\n--- Inventario ---");
        for (Producto producto : inventario) {
            producto.mostrarProducto();
            totalInventario += producto.calcularTotal();
            System.out.println();
        }
        System.out.println("Total del inventario: $" + totalInventario);
        scanner.close();
    }
}
```

Salida esperada (interacción):

```
Introduce el nombre del producto 1: Laptop
Introduce el precio de Laptop: 1000
Introduce la cantidad de Laptop: 5
Introduce el nombre del producto 2: Teclado
Introduce el precio de Teclado: 50
Introduce la cantidad de Teclado: 10
Introduce el nombre del producto 3: Mouse
Introduce el precio de Mouse: 25
Introduce la cantidad de Mouse: 20
--- Inventario ---
Nombre: Laptop
Precio: $1000.0
Cantidad: 5
Total: $5000.0
Nombre: Teclado
Precio: $50.0
Cantidad: 10
Total: $500.0
Nombre: Mouse
Precio: $25.0
Cantidad: 20
Total: $500.0
Total del inventario: $6000.0
```

9. Introducción al Uso de `Scanner`

`Scanner` es una clase en Java que permite leer datos de entrada del usuario desde la consola. Es muy útil para crear programas interactivos que requieren información del usuario.

Pasos para usar `Scanner`

1. Importar la clase `Scanner`:

- Antes de usarla, debes importarla al inicio de tu archivo.

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

2. Crear un objeto de tipo `Scanner`:

- Se crea usando el constructor `Scanner(System.in)`, que indica que se leerán datos desde la entrada estándar (la consola).

```java
Scanner scanner = new Scanner(System.in);
```

3. Leer datos con métodos específicos:

- Dependiendo del tipo de dato, `Scanner` tiene métodos como:
  - `nextLine()`: Lee una línea completa de texto.
  - `nextInt()`: Lee un número entero.
  - `nextDouble()`: Lee un número decimal.
- **Ejemplo**:

```
System.out.print("Introduce tu nombre: ");
String nombre = scanner.nextLine();
System.out.print("Introduce tu edad: ");
int edad = scanner.nextInt();
```

4. Cerrar el objeto `Scanner`:

- Es buena práctica cerrar el objeto para liberar recursos.

```
scanner.close();
```

1. Ejemplo Simple con `Scanner`

Código:

```
import java.util.Scanner;
public class EntradaEjemplo {
    public static void main(String[] args) {
        // Crear objeto Scanner
        Scanner scanner = new Scanner(System.in);
        // Leer un texto
        System.out.print("Introduce tu nombre: ");
        String nombre = scanner.nextLine();
        // Leer un entero
        System.out.print("Introduce tu edad: ");
        int edad = scanner.nextInt();
        // Mostrar la información
        System.out.println("Hola, " + nombre + ". Tienes " + edad + "
años.");
        // Cerrar el Scanner
        scanner.close();
    }
}
```

Salida (Ejemplo de ejecución):

```
Introduce tu nombre: Ana
Introduce tu edad: 25
Hola, Ana. Tienes 25 años.
```