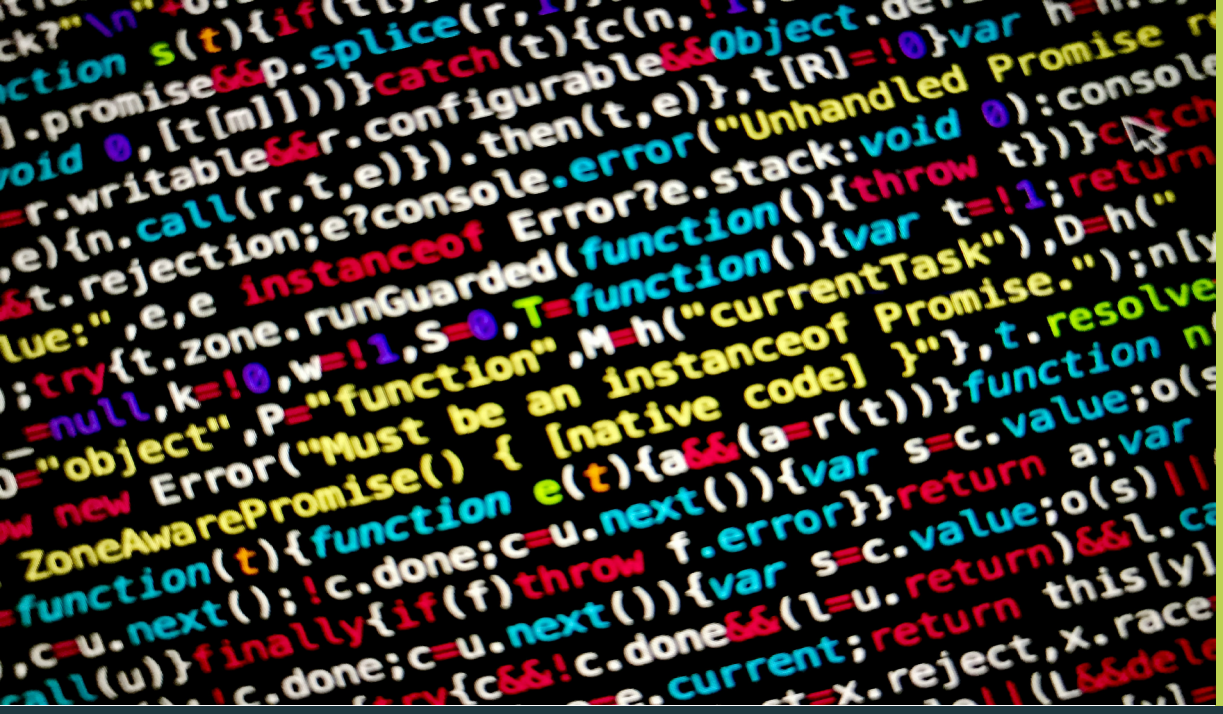

Unidad 2

Identificación de los elementos de un programa informático

Contenidos

- Parte 1: Estructura y bloques fundamentales 3
- Parte 2: Identificadores 9
- Parte 3: Palabras Reservadas 13
- Parte 4: Variables: Declaración, Inicialización y Utilización 18
- Parte 5: Tipos de Datos 24
- Parte 6: Literales 30
- Parte 7: Constantes 36
- Parte 8: Operadores y expresiones. Precedencia de operadores 42
- Parte 9: Conversiones de tipo. Implícitas y explicitas (casting). 48
- Parte 10: Comentarios 54
- Conclusión del Bloque 59

Parte 1: Estructura y bloques fundamentales

La estructura de un programa es su esqueleto: define cómo está organizado y cómo interactúan los distintos bloques que lo componen. En Java, la estructura es clara y rígida, lo que garantiza la consistencia y facilita el mantenimiento del código.

1. Estructura general de un programa Java

Un programa en Java sigue esta estructura básica:

```
// Declaración del paquete (opcional)
package nombreDelPaquete;
// Importación de clases (opcional)
import paquete.Clase;
// Declaración de la clase
public class NombreClase {
    // Declaración de atributos y métodos
    // Método principal
    public static void main(String[] args) {
        // Bloque de código principal
    }
}
```

2. Elementos clave de la estructura

1. Paquete (`package`):

- Define un grupo de clases relacionadas.
- Ayuda a organizar el código y evitar conflictos de nombres.
- Es opcional pero altamente recomendable en proyectos grandes.

```
package com.miProyecto.utilidades;
```

2. Importaciones (`import`):

- Permiten utilizar clases o paquetes de bibliotecas externas o del propio JDK.
- Son opcionales si no necesitas importar nada externo.

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

3. Clases:

- Son las unidades principales de un programa.
- Contienen atributos (estado) y métodos (comportamiento).
- Toda aplicación Java comienza con al menos una clase pública que contiene el método `main`.

```
public class Calculadora {
    // Atributos y métodos de la clase
}
```

4. Método principal (`main`):

- Es el punto de entrada de cualquier programa Java.
- La JVM busca este método para iniciar la ejecución del programa.

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

3. Bloques fundamentales

Los bloques en Java son secciones de código delimitadas por llaves `{}`. Cada bloque tiene un propósito específico y puede contener otros bloques anidados.

1. Bloque de declaración:

- Define variables, constantes, métodos y clases.
- Está presente en todos los niveles de un programa.

```
public class Persona {
    String nombre; // Declaración de variable
    int edad;    // Declaración de variable
}
```

2. Bloque de código:

- Contiene las instrucciones ejecutables.
- Generalmente se encuentra dentro de métodos.

```
public void saludar() {
    System.out.println("¡Hola!");
}
```

3. Bloque estático (`static`):

- Se ejecuta una vez al cargar la clase.
- Se utiliza para inicializar variables o realizar configuraciones iniciales.

```
static {
    System.out.println("Bloque estático ejecutado.");
}
```

4. Bloque de inicialización:

- Se ejecuta cada vez que se instancia un objeto.
- Se utiliza para inicializar atributos antes de ejecutar el constructor.

```
{
    System.out.println("Bloque de inicialización ejecutado.");
}
```

4. Ejemplo completo

Este programa utiliza todos los bloques mencionados:

```
package com.miProyecto;
import java.util.Scanner;
public class EjemploEstructura {
    // Bloque estático
    static {
        System.out.println("Bloque estático: Configuración inicial.");
    }
    // Bloque de inicialización
    {
        System.out.println("Bloque de inicialización: Preparando instancia.");
    }
    // Método principal
    public static void main(String[] args) {
        System.out.println("Método principal ejecutándose.");
        // Crear una instancia para activar el bloque de inicialización
        EjemploEstructura ejemplo = new EjemploEstructura();
        ejemplo.mostrarMensaje();
    }
    // Método de ejemplo
    public void mostrarMensaje() {
        System.out.println("Este es un mensaje desde un método.");
    }
}
```

Salida esperada:

```
Bloque estático: Configuración inicial.
Método principal ejecutándose.
Bloque de inicialización: Preparando instancia.
Este es un mensaje desde un método.
```

5. Ejercicios prácticos

Ejercicio 1: Identificar bloques

Dado el siguiente programa, indica:

1. ¿Qué tipo de bloque es cada uno?

2. ¿En qué orden se ejecutan?

```
public class Bloques {
    static {
        System.out.println("Bloque estático.");
    }
    {
        System.out.println("Bloque de inicialización.");
    }
    public Bloques() {
        System.out.println("Constructor.");
    }
    public static void main(String[] args) {
        System.out.println("Método principal.");
        Bloques ejemplo = new Bloques();
    }
}
```

Ejercicio 2: Crear un programa estructurado

1. Crea un programa con la siguiente estructura:

- Un paquete llamado `miPrograma`.
- Una clase llamada `Operaciones`.
- Un método principal que imprima "Programa iniciado".
- Un método llamado `sumar` que reciba dos números e imprima el resultado.

6. Preguntas de Evaluación

1. ¿Cuál es el propósito del método `main` en Java?

a) Es el punto de entrada del programa.

b) Define el nombre del paquete.

c) Declara las variables globales.

d) Importa bibliotecas externas.

2. ¿Qué bloque se ejecuta una vez al cargar la clase?

a) Bloque de inicialización.
b) Bloque estático.
c) Método principal.
d) Constructor.

3. ¿Qué ocurre si un programa Java no tiene un método `main` válido?

a) Se ejecuta correctamente.
b) El programa no compila.

- c) El programa lanza un error en tiempo de ejecución.

d) Nada, pero muestra una advertencia.

4. ¿Cuál es la principal diferencia entre un bloque de inicialización y un bloque estático?

a) El bloque estático solo se ejecuta al crear una instancia.
b) El bloque de inicialización se ejecuta una vez por clase.
c) El bloque de inicialización se ejecuta cada vez que se crea una instancia.
d) No hay diferencias significativas.

Parte 2: Identificadores

Los **identificadores** son los nombres que asignamos a los elementos del programa, como clases, métodos, variables y constantes. Son esenciales porque permiten referenciar estos elementos en el código de manera clara y estructurada.

1. ¿Qué es un identificador?

Un identificador es un nombre que:

1. Representa un elemento del programa.

2. Permite interactuar con clases, métodos, atributos y otros componentes.

Ejemplo de identificadores comunes

- Nombre de una clase: `Persona`
- Nombre de un método: `calcularEdad`
- Nombre de una variable: `edad`

```
public class Persona {  // Identificador de la clase
    int edad;           // Identificador de la variable
    public void saludar() {  // Identificador del método
          System.out.println("Hola");
    }
}
```

---

2. Reglas para nombrar identificadores en Java

1. Debe comenzar con una letra, el símbolo `_` o `$`.

- Ejemplo: `_variable`, `$precio`, `nombre`

2. No puede comenzar con un número.

- Ejemplo válido: `variable1`
- Ejemplo inválido: `1variable`

3. No puede contener caracteres especiales, como `@`, `#`, `%`, etc.

- Ejemplo válido: `miVariable`
- Ejemplo inválido: `mi#Variable`

4. No puede ser una palabra reservada del lenguaje.

- Ejemplo válido: `classNombre`
- Ejemplo inválido: `class` (es una palabra reservada)

5. Es sensible a mayúsculas y minúsculas.

- `edad` y `Edad` son identificadores diferentes.

Ejemplos de identificadores válidos e inválidos

```
// Válidos
int numero;
double _precio;
String $nombre;
int valor1;
// Inválidos
int 1variable; // Comienza con un número
String public; // Palabra reservada
double precio@;   // Contiene un carácter especial
```

3. Convenciones para nombrar identificadores

Aunque Java no impone reglas estrictas más allá de las mencionadas, existen **convenciones** que mejoran la legibilidad y el mantenimiento del código:

1. Clases: Usar el estilo PascalCase.

- Ejemplo: `Persona`, `CuentaBancaria`

2. Métodos y variables: Usar el estilo camelCase.

- Ejemplo: `calcularEdad`, `nombreCompleto`

3. Constantes: Usar mayúsculas y separar palabras con guion bajo.

- Ejemplo: `PI`, `VELOCIDAD_MAXIMA`

4. Evitar nombres genéricos como `a`, `b` o `data` (excepto para variables temporales o en bucles).

4. Buenas prácticas al elegir identificadores

1. Nombres significativos:

- Evita nombres genéricos como `x`, `y` o `temp` (excepto en casos muy específicos).
- Usa nombres que describan el propósito de la variable o método.

Ejemplo:

```
java
Copiar código
int edad = 25;           // Claro y significativo
double precioTotal = 99.99; // Indica el propósito
```

2. Usa prefijos o sufijos para indicar el propósito:

- Prefijos como `is` para booleanos: `isActivo`, `isDisponible`.
- Sufijos como `List` para colecciones: `empleadosList`.

3. Mantén consistencia en el estilo de nombres:

- No mezcles estilos: si usas `camelCase`, no cambies a `snake_case`.

5. Ejercicio práctico

Ejercicio 1: Identificar errores

Corrige los errores en los siguientes identificadores:

```
int 2edad;            // Inválido
String nombre@;      // Inválido
boolean isActive;    // Válido
double total_precios; // Válido pero no recomendado según las convenciones
```

Ejercicio 2: Mejorar nombres

Cambia los nombres genéricos por nombres significativos:

```
int a = 25;          // Cambiar por un nombre más claro
String b = "Juan";   // Cambiar por un nombre más descriptivo
boolean c = true;    // Cambiar por un nombre adecuado
```

Ejercicio 3: Crear identificadores

Diseña identificadores válidos y significativos para:

1. Una variable que almacene el nombre de un cliente.

2. Un método que calcule el total de una compra.

3. Una constante que represente el valor del IVA (21%).

6. Preguntas de evaluación

1. ¿Qué identificador es válido en Java?

a) `2variable`
b) `_nombre`
c) `public`
d) `precio@`

2. ¿Qué convención se recomienda para nombrar las clases?

a) camelCase
b) snake_case
c) PascalCase
d) UPPER_CASE

3. ¿Cuál de las siguientes opciones NO es una regla para los identificadores?

a) No pueden contener espacios.
b) No pueden ser palabras reservadas.
c) Deben comenzar con una letra, `_` o `$`.
d) Pueden contener cualquier carácter especial.

4. ¿Qué sucede si usas una palabra reservada como identificador?

a) El programa compila, pero lanza un error en tiempo de ejecución.
b) El programa compila sin problemas.
c) El programa no compila y lanza un error de sintaxis.
d) No sucede nada, pero muestra una advertencia.

Parte 3: Palabras Reservadas

Las **palabras reservadas** son términos predefinidos que tienen un significado especial en el lenguaje de programación Java. Estas palabras no pueden ser utilizadas como identificadores porque están reservadas para realizar funciones específicas dentro del lenguaje.

1. ¿Qué son las palabras reservadas?

1. Definición:

- Son términos que forman la sintaxis básica de Java.
- Tienen un propósito específico en el lenguaje, como definir clases, estructuras de control, tipos de datos, etc.

2. Uso:

- No pueden ser utilizadas para nombrar variables, métodos, clases o cualquier identificador.

2. Lista de palabras reservadas en Java

1. Palabras para la definición de estructura:

- `class`: Declara una clase.
- `interface`: Declara una interfaz.
- `enum`: Declara un enumerado.
- `extends`: Indica herencia entre clases.
- `implements`: Indica que una clase implementa una interfaz.
- `package`: Declara el paquete al que pertenece una clase.
- `import`: Permite incluir clases o paquetes externos.

2. Palabras para el flujo de control:

- `if`, `else`: Condicionales.
- `switch`, `case`, `default`: Estructuras de selección.
- `for`, `while`, `do`: Bucles.
- `break`: Sale de un bucle o `switch`.
- `continue`: Salta a la siguiente iteración de un bucle.
- `return`: Finaliza un método y devuelve un valor.
- `try`, `catch`, `finally`, `throw`, `throws`: Manejo de excepciones.

3. Palabras para tipos de datos y modificadores:

- **Tipos de datos primitivos**: `int`, `float`, `double`, `boolean`, `char`, `byte`, `short`, `long`.
- **Modificadores de acceso**: `public`, `private`, `protected`.
- **Otros modificadores**: `static`, `final`, `abstract`, `synchronized`, `volatile`, `transient`, `strictfp`, `native`.

4. Palabras para valores constantes:

- `true`, `false`: Valores booleanos.
- `null`: Valor nulo para objetos.

5. Palabras relacionadas con el manejo de memoria y referencias:

- `new`: Crea una nueva instancia de una clase.
- `this`: Referencia al objeto actual.
- `super`: Referencia a la superclase inmediata.

3. Palabras reservadas vs identificadores

Ejemplo válido:

```
int valor = 10; // "valor" es un identificador válido.
```

Ejemplo inválido:

```
int class = 10; // Error: "class" es una palabra reservada.
```

4. Palabras reservadas y contexto

Algunas palabras reservadas se utilizan en contextos específicos, como:

1. Definir clases y herencias:

```
public class Persona {
    // Código de la clase
}
```

2. Estructuras de control:

```
java
Copiar código
if (condicion) {
    // Bloque de código
} else {
    // Bloque alternativo
}
```

3. Manejo de excepciones:

```
java
Copiar código
try {
    // Código que puede lanzar una excepción
} catch (Exception e) {
    // Manejo de la excepción
}
```

5. Ejercicio práctico

Ejercicio 1: Identificar palabras reservadas

Dado el siguiente fragmento de código, identifica cuáles son palabras reservadas y cuáles son identificadores:

```
public class Ejemplo {
    private int edad;
    public void calcularEdad() {
          if (edad > 18) {
                System.out.println("Mayor de edad");
          } else {
                System.out.println("Menor de edad");
          }
    }
}
```

Ejercicio 2: Corrige los errores

Encuentra y corrige los errores en los siguientes fragmentos de código:

1. `javaCopiar códigoint if = 10;`
2. **`javaCopiar códigopublic static void new(String[] args) { System.out.println("Hola, Mundo");**

}`

3. `javaCopiar códigoboolean null = true;`

Ejercicio 3: Uso de palabras reservadas

Crea un programa que utilice las siguientes palabras reservadas:

- `class`
- `public`
- `if`
- `return`

6. Preguntas de evaluación

1. ¿Cuál de las siguientes es una palabra reservada en Java?

a) `valor`
b) `public`
c) `variable`
d) `imprimir`

2. ¿Qué ocurre si intentas usar una palabra reservada como identificador?

a) El programa compila pero lanza un error en tiempo de ejecución.
b) El programa compila sin problemas.
c) El programa no compila y lanza un error de sintaxis.
d) No sucede nada, pero muestra una advertencia.

3. ¿Qué palabra reservada se utiliza para crear una instancia de una clase?

a) `this`
b) `new`
c) `static`
d) `super`

4. ¿Qué palabra reservada define una condición en Java?

a) `for`
b) `if`
c) `switch`
d) `else`

5. ¿Cuál es el propósito de la palabra reservada `null`?

a) Indica un valor numérico vacío.

b) Declara una nueva clase.

c) Representa la ausencia de un valor o referencia.

d) Detiene la ejecución de un bucle.

Parte 4: Variables: Declaración, Inicialización y Utilización

Las **variables** son uno de los elementos más fundamentales de cualquier programa. Representan espacios en memoria donde se almacena información que el programa puede manipular.

1. ¿Qué es una variable?

Una **variable** es un contenedor que guarda un valor temporalmente mientras se ejecuta el programa. Cada variable tiene:

1. **Un nombre: El identificador para referirse a ella.**
2. **Un tipo de dato: Define el tipo de información que puede almacenar.**
3. **Un valor: El dato actual almacenado en la variable.**

2. Declaración de variables

La **declaración** de una variable consiste en reservar un espacio en memoria y definir el tipo de dato que contendrá.

Sintaxis básica

```
tipo nombreVariable;
```

- `tipo`: Indica el tipo de dato (por ejemplo, `int`, `double`, `String`).
- `nombreVariable`: Identificador único para la variable.

Ejemplo de declaración

```
int edad;           // Variable de tipo entero
double precio;   // Variable de tipo decimal
String nombre;   // Variable de tipo texto
```

3. Inicialización de variables

La inicialización consiste en asignar un valor a una variable. Puede hacerse al declararla o después.

Inicialización al declarar

```
int edad = 25;
```

Inicialización posterior

```
int edad;
edad = 25;
```

4. Utilización de variables

Una vez declarada e inicializada, puedes usar una variable para realizar operaciones, mostrar su valor, o asignarle un nuevo valor.

Ejemplo práctico

```
public class Variables {
    public static void main(String[] args) {
          int edad = 25;           // Declaración e inicialización
          System.out.println(edad); // Utilización: imprimir el valor
          edad = 30;               // Asignación de un nuevo valor
          System.out.println(edad); // Utilización: imprimir el nuevo valor
    }
}
```

Salida:

```
25
30
```

5. Ámbito de una variable

El **ámbito** de una variable determina dónde es accesible dentro del programa.

1. Variable local:

- Declarada dentro de un método.
- Solo accesible dentro de ese método.

  Ejemplo:

```
public void saludar() {
    String mensaje = "Hola";
    System.out.println(mensaje);
}
```

2. Variable de instancia:

- Declarada en la clase, pero fuera de cualquier método.
- Cada instancia de la clase tiene su propia copia.

  Ejemplo:

```
public class Persona {
    int edad;  // Variable de instancia
}
```

3. Variable estática:

- Compartida por todas las instancias de la clase.
- Declarada con la palabra clave `static`.

  Ejemplo:

```
public class Contador {
    static int total; // Variable estática
}
```

6. Buenas prácticas al usar variables

1. Usa nombres significativos:

- En lugar de `int x = 10;`, utiliza `int edad = 10;`.

2. Inicializa las variables antes de usarlas:

- Java no permite usar variables locales sin inicializarlas.

3. Evita nombres de una sola letra, excepto en bucles:

- Ejemplo: Usar `i` en `for (int i = 0; i < 10; i++)`.

4. Declara variables en el ámbito más pequeño posible:

- Si solo necesitas la variable dentro de un método, no la declares como atributo de la clase.

7. Ejemplo completo

Crea un programa que utilice diferentes tipos de variables y muestra sus valores:

```
public class EjemploVariables {
    // Variable estática
    static String lenguaje = "Java";
    // Variable de instancia
    int edad = 25;
    public static void main(String[] args) {
          // Variable local
          String mensaje = "Bienvenido a ";
          // Utilización de variables
          System.out.println(mensaje + lenguaje);
          // Crear una instancia para usar la variable de instancia
          EjemploVariables ejemplo = new EjemploVariables();
          System.out.println("Edad: " + ejemplo.edad);
    }
}
```

Salida:

```
Bienvenido a Java
Edad: 25
```

8. Ejercicios prácticos

Ejercicio 1: Declarar e inicializar variables

Declara e inicializa variables para representar:

1. El nombre de una persona.

2. Su edad.

3. Su altura (en metros).

4. Si está empleado o no (booleano).

Ejercicio 2: Uso y modificación

1. Declara una variable para almacenar un número.

2. Muestra su valor inicial.

3. Modifica el valor y vuelve a mostrarlo.

Ejercicio 3: Ámbito de variables

Analiza el siguiente código y responde:

1. ¿Qué sucede si intentas acceder a `mensaje` fuera del método `mostrarMensaje`?
2. ¿Qué sucede si cambias `edad` a una variable local dentro de `main`?

```
public class EjemploAmbito {
    int edad = 30; // Variable de instancia
    public void mostrarMensaje() {
          String mensaje = "Hola, Java!";
          System.out.println(mensaje);
    }
    public static void main(String[] args) {
          EjemploAmbito ejemplo = new EjemploAmbito();
          ejemplo.mostrarMensaje();
          System.out.println(ejemplo.edad);
    }
}
```

9. Preguntas de evaluación

1. ¿Cuál es la diferencia entre declarar y inicializar una variable?

a) Declarar crea una variable; inicializar asigna un valor.
b) Declarar asigna un valor; inicializar la crea.
c) No hay diferencia.
d) Declarar es opcional en Java.

2. ¿Qué sucede si intentas usar una variable local sin inicializar?

a) El programa lanza un error en tiempo de ejecución.
b) El programa no compila.
c) Se asigna un valor predeterminado automáticamente.
d) La variable toma el valor de otra variable.

3. ¿Qué palabra clave se utiliza para declarar una variable compartida por todas las instancias de una clase?

a) `public`
b) `private`
c) `final`
d) `static`

4. ¿Cuál es el ámbito de una variable declarada dentro de un método?

a) Clase.

b) Método.

c) Paquete.

d) Global.

Parte 5: Tipos de Datos

En Java, los **tipos de datos** son fundamentales para declarar variables y definir el tipo de información que estas pueden almacenar. Se dividen en **tipos primitivos** y **tipos no primitivos**.

1. Clasificación de los tipos de datos

1. Tipos primitivos:

- Representan datos básicos como números, caracteres y valores booleanos.
- Son los más eficientes en términos de memoria y rendimiento.

2. Tipos no primitivos:

- Incluyen objetos, cadenas de texto y arrays.
- Proporcionan una mayor flexibilidad y funcionalidad.

Tipos primitivos

Tipo	Descripción	Tamaño	Valores posibles
`byte`	Entero de 8 bits	1 byte	-128 a 127
`short`	Entero de 16 bits	2 bytes	-32,768 a 32,767
`int`	Entero de 32 bits	4 bytes	-2,147,483,648 a 2,147,483,647
`long`	Entero de 64	8 bytes	-2^63 a 2^63-1
`float`	Decimal de precisión simple	4 bytes	Aproximadamente 7 decimales
`double`	Decimal de doble precisión	8 bytes	Aproximadamente 15 decimales
`char`	Carácter Unicode	2 bytes	Cualquier carácter
`boolean`	Valor lógico	1 bit	`true` o `false`

Ejemplo de uso de tipos primitivos

```
public class TiposPrimitivos {
    public static void main(String[] args) {
          int edad = 25;                // Entero
          float altura = 1.75f;     // Decimal con "f" para float
          double peso = 70.5;          // Decimal de doble precisión
          char inicial = 'J';          // Carácter
          boolean esEstudiante = true; // Valor lógico
          System.out.println("Edad: " + edad);
          System.out.println("Altura: " + altura + " m");
          System.out.println("Peso: " + peso + " kg");
          System.out.println("Inicial: " + inicial);
          System.out.println("¿Es estudiante? " + esEstudiante);
    }
}
```

Salida:

```
Edad: 25
Altura: 1.75 m
Peso: 70.5 kg
Inicial: J
¿Es estudiante? true
```

Tipos no primitivos

1. Cadenas de texto (`String`):

- Almacenan texto de cualquier longitud.
- No es un tipo primitivo, pero se utiliza ampliamente.

  Ejemplo:

```
String nombre = "Juan";
System.out.println("Hola, " + nombre + "!");
```

2. Arrays:

- Colecciones de elementos del mismo tipo.
- Su tamaño es fijo una vez definido.

  Ejemplo:

```
int[] numeros = {1, 2, 3, 4, 5};
System.out.println("Primer número: " + numeros[0]);
```

3. Objetos:

- Representan instancias de clases.
- Se crean con la palabra clave `new`.

  Ejemplo:

```
Persona persona = new Persona("Ana", 30);
```

2. Conversión entre tipos de datos

Conversión implícita

Ocurre automáticamente cuando un tipo más pequeño se asigna a un tipo más grande.

Ejemplo:

```
int numero = 10;
double numeroDecimal = numero; // Conversión implícita
System.out.println(numeroDecimal); // 10.0
```

Conversión explícita (casting)

Se requiere cuando un tipo más grande se asigna a uno más pequeño. Puede perderse precisión.

Ejemplo:

```
double numeroDecimal = 10.5;
int numero = (int) numeroDecimal; // Conversión explícita
System.out.println(numero); // 10
```

3. Valores predeterminados

En Java, las variables de instancia tienen valores predeterminados si no se inicializan.

Tipo	Valor predeterminado
`int`	0
`double`	0.0
`boolean`	`false`
`char`	`\u0000` (vacío)
`String`	`null`

4. Ejemplo completo

Este programa utiliza varios tipos de datos, realiza conversiones y muestra valores predeterminados:

```
public class TiposDatos {
    int numero;            // Valor predeterminado: 0
    double decimal;   // Valor predeterminado: 0.0
    boolean esActivo;    // Valor predeterminado: false
    String nombre;     // Valor predeterminado: null
    public static void main(String[] args) {
          TiposDatos datos = new TiposDatos();
          System.out.println("Número: " + datos.numero);
          System.out.println("Decimal: " + datos.decimal);
          System.out.println("Activo: " + datos.esActivo);
          System.out.println("Nombre: " + datos.nombre);
          // Conversión implícita
          int entero = 50;
          double resultado = entero;
          System.out.println("Resultado implícito: " + resultado);
          // Conversión explícita
          double valor = 99.99;
          int enteroCasteado = (int) valor;
          System.out.println("Resultado explícito: " + enteroCasteado);
    }
}
```

Salida:

```
Número: 0
Decimal: 0.0
Activo: false
Nombre: null
Resultado implícito: 50.0
Resultado explícito: 99
```

5. Ejercicios prácticos

Ejercicio 1: Declarar y mostrar tipos primitivos

1. Declara variables para representar:

- La edad de una persona (`int`).
- La temperatura actual (`float`).
- Un carácter inicial (`char`).
- Si es estudiante (`boolean`).

2. Asigna valores e imprime cada uno.

Ejercicio 2: Conversión entre tipos

1. Declara un número decimal (`double`) con valor 45.8.
2. Convierte este número a un entero (`int`) usando casting.
3. Imprime ambos valores.

Ejercicio 3: Valores predeterminados

1. Crea una clase con atributos:

- `int` para número de empleados.
- `boolean` para estado activo.
- `String` para nombre de la empresa.

2. Imprime los valores predeterminados de cada atributo.

6. Preguntas de evaluación

1. ¿Qué tipo de dato usarías para almacenar el salario de un empleado?

a) `boolean`
b) `String`
c) `float`
d) `int`

2. ¿Qué sucede si intentas asignar un valor `double` a un `int` sin casting?

a) El programa lanza un error en tiempo de ejecución.
b) El programa lanza un error en tiempo de compilación.
c) La conversión se realiza automáticamente.
d) El valor se trunca.

3. ¿Cuál es el valor predeterminado de una variable `boolean` no inicializada?

a) `true`
b) `false`
c) `null`
d) No tiene valor predeterminado.

4. ¿Qué palabra clave se utiliza para crear una instancia de un objeto?

a) `new`
b) `this`
c) `static`
d) `return`

5. ¿Qué sucede si intentas convertir un número decimal a entero usando casting?

a) Lanza un error.
b) Se redondea automáticamente.
c) Se trunca el valor decimal.
d) El valor decimal se conserva.

Parte 6: Literales

Los **literales** son valores constantes que se escriben directamente en el código fuente y representan un tipo de dato específico. En Java, cada literal pertenece a un tipo de dato, como enteros, decimales, caracteres, cadenas de texto, o valores booleanos.

1. Tipos de literales

1. Literales enteros

Representan números enteros. Por defecto, los literales enteros son de tipo `int`, pero pueden convertirse a `long` añadiendo una `L` al final.

Ejemplos:

```
int numero = 100;   // Literal entero
long numeroLargo = 100000L; // Literal entero tipo long
```

2. Literales de punto flotante

Representan números decimales. Por defecto, los literales de punto flotante son de tipo `double`. Para definir un `float`, se añade una `F` o `f`.

Ejemplos:

```
double precio = 19.99;    // Literal decimal tipo double
float temperatura = 36.6F; // Literal decimal tipo float
```

3. Literales booleanos

Solo pueden tener dos valores: `true` o `false`.

Ejemplo:

```
boolean esActivo = true; // Literal booleano
boolean esMayor = false; // Literal booleano
```

4. Literales de caracteres

Representan un solo carácter Unicode. Se escriben entre comillas simples (`'`).

Ejemplos:

```
char inicial = 'A';   // Literal carácter
char simbolo = '#';   // Literal carácter
```

También se pueden usar caracteres especiales como:

- `\n`: Nueva línea
- `\t`: Tabulación
- `\\`: Barra invertida
- `\'`: Comilla simple
- `\"`: Comilla doble

Ejemplo:

```
char nuevaLinea = '\n'; // Representa un salto de línea
```

5. Literales de cadenas

Representan texto, delimitado por comillas dobles (`"`).

Ejemplo:

```
String mensaje = "Bienvenido a Java";
```

6. Literales nulos

El literal `null` representa la ausencia de valor o referencia en objetos.

Ejemplo:

```
String sinValor = null;
```

2. Literales en diferentes sistemas numéricos

Java permite representar números en diferentes sistemas numéricos:

1. Decimal (base 10):

- Es el sistema numérico más común.

  Ejemplo: `int numero = 123;`

2. Binario (base 2):

- Se indica con el prefijo `0b`.

  Ejemplo:

```
int binario = 0b1101; // Representa el número 13
```

3. Octal (base 8):

- Se indica con el prefijo `0`.

  Ejemplo:

```
int octal = 017; // Representa el número 15 en decimal
```

4. Hexadecimal (base 16):

- Se indica con el prefijo `0x`.

  Ejemplo:

```
int hexadecimal = 0x1A; // Representa el número 26 en decimal
```

3. Literales especiales

1. Separadores de números:

- Puedes usar `_` como separador visual en literales numéricos para mejorar la legibilidad.

  Ejemplo:

```
int grande = 1_000_000; // Más legible que 1000000
```

2. Literals para números negativos:

- Se pueden definir utilizando el operador .

  Ejemplo:

```
int negativo = -50;
```

4. Ejemplo completo

El siguiente programa utiliza varios tipos de literales:

```
public class EjemploLiterales {
    public static void main(String[] args) {
        // Literales enteros
        int decimal = 100;
        int binario = 0b1101;
        int octal = 017;
        int hexadecimal = 0x1A;
        // Literales de punto flotante
        double precio = 19.99;
        float temperatura = 36.6F;
        // Literales booleanos
        boolean esActivo = true;
        // Literales de caracteres
        char letra = 'A';
        char saltoLinea = '\n';
        // Literales de cadenas
        String mensaje = "Hola, Mundo";
        // Literales nulos
        String sinValor = null;
        // Imprimir valores
        System.out.println("Decimal: " + decimal);
        System.out.println("Binario: " + binario);
        System.out.println("Octal: " + octal);
        System.out.println("Hexadecimal: " + hexadecimal);
        System.out.println("Precio: " + precio);
        System.out.println("Temperatura: " + temperatura);
        System.out.println("Activo: " + esActivo);
        System.out.println("Letra: " + letra);
        System.out.println("Mensaje: " + mensaje);
        System.out.println("Sin valor: " + sinValor);
    }
}
```

Salida:

```
Decimal: 100
Binario: 13
Octal: 15
Hexadecimal: 26
Precio: 19.99
Temperatura: 36.6
Activo: true
Letra: A
Mensaje: Hola, Mundo
Sin valor: null
```

5. Ejercicios prácticos

Ejercicio 1: Literales básicos

Declara e imprime literales para:

1. **Un número binario que represente 5.**
2. **Una cadena con el mensaje `"Java es divertido"`.**
3. **Un carácter especial como `\n`.**

Ejercicio 2: Literales en sistemas numéricos

1. **Declara una variable para cada sistema numérico: decimal, binario, octal y hexadecimal.**
2. **Convierte y muestra el valor en decimal.**

Ejercicio 3: Uso de separadores en números grandes

Declara un número grande como `10,000,000` utilizando separadores `_` y muestra su valor.

6. Preguntas de evaluación

1. ¿Qué tipo de literal es `true` en Java?

a) Entero
b) Booleano
c) Carácter
d) Cadena

2. ¿Qué prefijo se utiliza para representar números en binario?

a) `0b`
b) `0x`
c) `0`
d) `bin`

3. ¿Cuál de los siguientes es un literal válido?

a) `'Hola'`
b) `123_456`
c) `false_`
d) `null123`

4. ¿Qué representa el literal `null`?

a) Una cadena vacía.
b) Un valor booleano falso.
c) La ausencia de valor.
d) Un número negativo.

5. ¿Cuál es el resultado de ejecutar el siguiente código?

```
int valor = 0b1010;
System.out.println(valor);
```

a) 10
b) 5
c) 1010
d) Error de compilación

Parte 7: Constantes

Las **constantes** son elementos cuyo valor no cambia durante la ejecución del programa. En Java, se definen utilizando la palabra clave `final` y, por convención, se escriben en mayúsculas.

1. ¿Qué es una constante?

1. Definición:

- Una constante es una variable con un valor fijo, que no puede ser modificado una vez asignado.
- Sirve para representar valores inmutables, como pi, la gravedad, o límites de aplicación.

2. Características:

- Debe ser inicializada en el momento de su declaración.
- Su valor no puede cambiar después de asignado.

2. Sintaxis para declarar una constante

Para declarar una constante en Java:

- Usa la palabra clave `final`.
- Escribe el nombre en mayúsculas con palabras separadas por guiones bajos (_), según las convenciones de código.

Ejemplo: Declaración de constantes

```
public class Constantes {
    public static final double PI = 3.14159; // Constante global
    public static final int MAX_USUARIOS = 100;
    public static void main(String[] args) {
          System.out.println("Valor de PI: " + PI);
          System.out.println("Máximo de usuarios: " + MAX_USUARIOS);
    }
}
```

Salida:

```
Valor de PI: 3.14159
Máximo de usuarios: 100
```

3. Ventajas de usar constantes

1. Legibilidad:

- Los nombres descriptivos de las constantes mejoran la claridad del código.
- Ejemplo: `MAX_USUARIOS` es más claro que un número "mágico" como `100`.

2. Mantenibilidad:

- Si necesitas cambiar el valor de la constante, solo lo haces en un lugar.

3. Prevención de errores:

- Garantiza que los valores críticos no se modifiquen accidentalmente.

4. Ámbitos de las constantes

1. Constantes locales:

- Declaradas dentro de un método.
- Solo son accesibles dentro de ese método.

Ejemplo:

```
public class Ejemplo {
    public static void main(String[] args) {
        final int LIMITE = 50;
        System.out.println("Límite local: " + LIMITE);
    }
}
```

2. Constantes globales:

- Declaradas como `static final` en una clase.
- Accesibles desde cualquier parte del programa si son públicas.

Ejemplo:

```
java
Copiar código
public class Configuracion {
    public static final String APP_NOMBRE = "MiAplicacion";
}
public class Main {
    public static void main(String[] args) {
        System.out.println("Nombre de la aplicación: " + Configuracion.APP_
NOMBRE);
    }
}
```

Salida:

```
Copiar código
Nombre de la aplicación: MiAplicacion
```

5. Constantes y enumeraciones

En Java, puedes usar `enum` para definir un conjunto de constantes relacionadas.

Ejemplo: Usando `enum`

```
public enum DiaSemana {
    LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO;
}
public class EjemploEnum {
    public static void main(String[] args) {
        DiaSemana dia = DiaSemana.MARTES;
        System.out.println("Día seleccionado: " + dia);
        // Usar en una estructura de control
        switch (dia) {
            case LUNES:
                System.out.println("Inicio de la semana");
                break;
            case VIERNES:
                System.out.println("¡Casi fin de semana!");
                break;
            default:
                System.out.println("Otro día");
        }
    }
}
```

Salida:

```
Día seleccionado: MARTES
Otro día
```

6. Ejemplo completo

Este programa utiliza constantes para configurar una aplicación:

```
  public class Configuracion {
      // Constantes globales
      public static final String APP_NOMBRE = “MiAplicacion”;
      public static final int MAX_INTENTOS = 3;
      public static final double IVA = 0.21;
      public static void main(String[] args) {
            System.out.println(“Nombre de la aplicación: “ + APP_NOMBRE);
            System.out.println(“Intentos máximos permitidos: “ + MAX_INTENTOS);
            System.out.println(“Impuesto aplicado: “ + (IVA * 100) + “%”);
            // Constante local
            final int LIMITE_USUARIOS = 10;
            System.out.println(“Límite de usuarios locales: “ + LIMITE_USUARI-
OS);
      }
  }
```

Salida:

```
  Nombre de la aplicación: MiAplicacion
  Intentos máximos permitidos: 3
  Impuesto aplicado: 21.0%
  Límite de usuarios locales: 10
```

7. Ejercicios prácticos

Ejercicio 1: Crear constantes globales

Crea una clase llamada `Configuracion` con las siguientes constantes:

1. `NOMBRE_EMPRESA` **(nombre de la empresa).**
2. `MAX_PRODUCTOS` **(máximo número de productos permitidos).**
3. `TASA_CAMBIO` **(tasa de cambio del dólar al euro).**

En la clase principal, imprime los valores de estas constantes.

Ejercicio 2: Usar `enum` para días de la semana

1. **Declara un** `enum` **para los días de la semana.**
2. **En la clase principal, selecciona un día y muestra un mensaje asociado.**

Ejercicio 3: Calcular con constantes

Usa constantes para:

1. Definir el valor de `PI`.

2. Calcular el área de un círculo con un radio dado.

8. Preguntas de evaluación

1. ¿Qué palabra clave se utiliza para declarar una constante en Java?

a) `const`
b) `static`
c) `final`
d) `constant`

2. ¿Qué sucede si intentas modificar el valor de una constante?

a) El programa lanza un error en tiempo de ejecución.
b) El programa no compila.
c) La constante toma el nuevo valor sin problemas.
d) Se lanza una advertencia pero compila.

3. ¿Qué convención se recomienda para nombrar las constantes?

a) camelCase
b) snake_case
c) PascalCase
d) UPPER_CASE

4. ¿Cuál es el resultado del siguiente código?

```
public class Prueba {
    public static final int LIMITE = 10;
    public static void main(String[] args) {
          LIMITE = 20;
          System.out.println(LIMITE);
    }
}
```

a) 20
b) 10
c) Error de compilación
d) Error en tiempo de ejecución

5. ¿Qué se puede usar para definir un conjunto de constantes relacionadas?

a) `final`
b) `enum`
c) `abstract`
d) `static`

Parte 8: Operadores y expresiones. Precedencia de operadores

Los **operadores** son símbolos o palabras que indican operaciones a realizar sobre uno o más operandos. Las **expresiones** son combinaciones de operadores y operandos que producen un resultado. La **precedencia de operadores** define el orden en que se evalúan las expresiones.

1. Clasificación de operadores en Java

1. Operadores aritméticos

Se utilizan para realizar operaciones matemáticas.

Operador	Descripción	Ejemplo	Resultado
`+`	Suma	`3 + 2`	`5`
`-`	Resta	`5 - 3`	`2`
`*`	Multiplicación	`4 * 2`	`8`
`/`	División	`10 / 2`	`5`
`%`	Módulo (resto)	`10 % 3`	`1`

2. Operadores relacionales

Comparan valores y devuelven un resultado booleano.

Operador	Descripción	Ejemplo	Resultado
`==`	Igualdad	`5 == 5`	`true`
`!=`	Diferente	`5 != 3`	`true`
`>`	Mayor que	`5 > 3`	`true`
`<`	Menor que	`3 < 5`	`true`
`>=`	Mayor o igual que	`5 >= 5`	`true`
`<=`	Menor o igual que	`3 <= 5`	`true`

3. Operadores lógicos

Se utilizan para combinar condiciones.

Operador	Descripción	Ejemplo	Resultado
`&&`	AND lógico	`true && false`	`false`
`	`		`	OR lógico
`!`	NOT lógico	`!true`	`false`

4. Operadores de asignación

Asignan valores a variables.

Operador	Descripción	Ejemplo	Equivalente
`=`	Asignación	`x = 5`	---
`+=`	Suma y asigna	`x += 3`	`x = x + 3`
`-=`	Resta y asigna	`x -= 2`	`x = x - 2`
`*=`	Multiplica y asigna	`x *= 4`	`x = x * 4`
`/=`	Divide y asigna	`x /= 2`	`x = x / 2`
`%=`	Módulo y asigna	`x %= 3`	`x = x % 3`

5. Operadores unarios

Operan sobre un solo operando.

Operador	Descripción	Ejemplo	Resultado
`+`	Indica positivo	`+5`	`5`
`-`	Invierte el signo	`-5`	`-5`
`++`	Incremento	`x++`	`x = x + 1`
`--`	Decremento	`x--`	`x = x - 1`

6. Operador ternario

Es una forma compacta de escribir una condición `if-else`.

Sintaxis:

```
variable = (condición) ? valor_si_verdadero : valor_si_falso;
```

Ejemplo:

```
int edad = 18;
String mensaje = (edad >= 18) ? “Mayor de edad” : “Menor de edad”;
System.out.println(mensaje); // Mayor de edad
```

2. Precedencia de operadores

Cuando hay múltiples operadores en una expresión, **Java sigue un orden de precedencia** para evaluarlos.

Tabla de precedencia

Nivel	Operadores	Asociatividad
1	`++`, `--`	De derecha a izquierda
2	`*`, `/`, `%`	De izquierda a derecha
3	`+`, `-`	De izquierda a derecha
4	`<`, `<=`, `>`, `>=`	De izquierda a derecha
5	`==`, `!=`	De izquierda a derecha
6	`&&`	De izquierda a derecha
7	`	
8	`? :` (ternario)	De derecha a izquierda
9	`=`, `+=`, `-=`, `*=`, `/=`, `%=`	De derecha a izquierda

Nota: Puedes usar paréntesis para forzar el orden de evaluación.

3. Ejemplo completo

Este programa utiliza varios operadores y muestra cómo afectan las expresiones:

```
  public class Operadores {
      public static void main(String[] args) {
            // Operadores aritméticos
            int a = 10, b = 5;
            System.out.println("Suma: " + (a + b));
            System.out.println("Resta: " + (a - b));
            System.out.println("Multiplicación: " + (a * b));
            System.out.println("División: " + (a / b));
            System.out.println("Módulo: " + (a % b));
            // Operadores relacionales
            System.out.println("¿a es mayor que b? " + (a > b));
            System.out.println("¿a es igual a b? " + (a == b));
            // Operadores lógicos
            boolean resultado = (a > b) && (b > 0);
            System.out.println("Resultado lógico: " + resultado);
            // Operador ternario
            String mensaje = (a > b) ? "a es mayor" : "b es mayor o igual";
            System.out.println(mensaje);
            // Precedencia de operadores
            int resultadoPrecedencia = a + b * 2; // b * 2 se evalúa primero
            System.out.println("Resultado con precedencia: " + resultadoPrece-
dencia);
      }
  }
```

Salida:

```
Suma: 15
Resta: 5
Multiplicación: 50
División: 2
Módulo: 0
¿a es mayor que b? true
¿a es igual a b? false
Resultado lógico: true
a es mayor
Resultado con precedencia: 20
```

4. Ejercicios prácticos

Ejercicio 1: Operadores básicos

1. Declara dos números enteros.
2. Realiza operaciones aritméticas (`+`, , , `/`, `%`) y muestra los resultados.

Ejercicio 2: Operadores relacionales y lógicos

1. Declara tres variables (`a`, `b`, `c`) con valores diferentes.
2. Escribe una expresión que:

- Compruebe si `a` es mayor que `b` y `b` es mayor que `c`.
- Compruebe si alguno de los números es negativo.

Ejercicio 3: Operador ternario

Crea un programa que:

1. Solicite al usuario su edad.
2. Utilice un operador ternario para determinar si es mayor o menor de edad.

Ejercicio 4: Evaluación con precedencia

Evalúa las siguientes expresiones y explica el orden de ejecución:

1. `10 + 2 * 5`
2. `(10 + 2) * 5`
3. `10 > 5 && 5 < 3`

5. Preguntas de evaluación

1. ¿Qué operador devuelve el resto de una división?

a) `+`
b) `%`
c) `/`
d)

2. ¿Cuál es el resultado de la expresión `5 > 3 && 3 < 2`?

a) `true`
b) `false`
c) `null`
d) Error de compilación

3. ¿Qué operador tiene la mayor precedencia?

a) `+`
b)
c) `&&`
d) `++`

4. ¿Qué hace el operador ternario `? :`?

a) Divide dos números.
b) Compara dos valores.
c) Evalúa una condición y devuelve un resultado.
d) Asigna un valor.

5. ¿Cuál es el resultado de `10 + 2 * 3` en Java?

a) `36`
b) `16`
c) `12`
d) `20`

Parte 9: Conversiones de tipo. Implícitas y explicitas (casting).

En Java, las conversiones de tipo son procesos para cambiar el tipo de dato de una variable o expresión a otro tipo compatible. Esto puede ser necesario para realizar cálculos o asignaciones entre tipos diferentes.

1. Tipos de conversiones

Existen dos tipos principales de conversiones en Java:

1. Conversión implícita:

- También llamada promoción de tipo.
- Ocurre automáticamente cuando se asigna un valor de un tipo más pequeño a un tipo más grande.
- No requiere intervención explícita del programador.

  Ejemplo:

```
int numeroEntero = 100;
double numeroDecimal = numeroEntero; // Conversión implícita
System.out.println(numeroDecimal); // 100.0
```

2. Conversión explícita (casting):

- Se realiza de forma manual cuando se necesita convertir un tipo más grande a uno más pequeño.
- Puede implicar pérdida de datos si no se realiza con cuidado.
- Utiliza la sintaxis (tipo).

  Ejemplo:

```
double numeroDecimal = 99.99;
int numeroEntero = (int) numeroDecimal; // Conversión explícita
System.out.println(numeroEntero); // 99
```

2. Reglas de conversión implícita

Java permite la conversión implícita solo si no hay riesgo de pérdida de precisión. El orden de promoción es:

```
byte -> short -> int -> long -> float -> double
```

Ejemplo de promoción implícita

```
public class ConversionImplicita {
    public static void main(String[] args) {
          int numeroEntero = 42;
          double numeroDecimal = numeroEntero; // Conversión implícita
          System.out.println(numeroDecimal); // 42.0
    }
}
```

3. Conversión explícita (casting)

Cuando usarla

1. Cuando necesitas reducir el tamaño de un tipo de dato (de double a int).
2. Cuando trabajas con tipos incompatibles y necesitas convertirlos (como char a `int`).

Ejemplo de casting

```
public class ConversionExplicita {
    public static void main(String[] args) {
          double numeroDecimal = 45.76;
          int numeroEntero = (int) numeroDecimal; // Conversión explícita
          System.out.println(numeroEntero); // 45
    }
}
```

Nota: En este ejemplo, la parte decimal (`0.76`) se pierde.

4. Conversión entre tipos incompatibles

Java no permite conversiones automáticas entre tipos incompatibles como `String` a `int`. Para esto, necesitas métodos específicos o clases auxiliares.

Convertir un `String` a `int`

Usa el método `Integer.parseInt`.

Ejemplo:

```
String texto = "123";
int numero = Integer.parseInt(texto);
System.out.println(numero + 2); // 125
```

Convertir un `int` a `String`

Usa el método `String.valueOf` o concatenación con una cadena.

Ejemplo:

```
int numero = 456;
String texto = String.valueOf(numero);
System.out.println("Número como texto: " + texto);
```

5. Conversiones entre tipos de datos primitivos y objetos (autoboxing/unboxing)

Java proporciona **autoboxing** y **unboxing** para convertir automáticamente entre tipos primitivos y sus equivalentes en clase.

1. Autoboxing: Convierte automáticamente un tipo primitivo a su clase envolvente**.**

Ejemplo:

```
int numero = 10;
Integer objetoNumero = numero; // Autoboxing
```

2. Unboxing: Convierte automáticamente una clase envolvente a su tipo primitivo.

Ejemplo:

```
Integer objetoNumero = 20;
int numero = objetoNumero; // Unboxing
```

6. Ejemplo completo

El siguiente programa demuestra conversiones implícitas, explícitas y entre tipos incompatibles:

```
  public class Conversiones {
      public static void main(String[] args) {
            // Conversión implícita
            int numeroEntero = 100;
            double numeroDecimal = numeroEntero; // Implicita
            System.out.println("Implicita: " + numeroDecimal); // 100.0
            // Conversión explícita
            double numeroGrande = 99.99;
            int numeroPequeño = (int) numeroGrande; // Explícita
            System.out.println("Explícita: " + numeroPequeño); // 99
            // Conversión entre String y otros tipos
            String texto = "50";
            int numeroDesdeTexto = Integer.parseInt(texto); // De String a int
            System.out.println("De String a int: " + (numeroDesdeTexto + 10));
// 60
            int numero = 123;
            String textoDesdeNumero = String.valueOf(numero); // De int a String
            System.out.println("De int a String: " + textoDesdeNumero);
      }
  }
```

Salida:

```
  Implicita: 100.0
  Explícita: 99
  De String a int: 60
  De int a String: 123
```

7. Ejercicios prácticos

Ejercicio 1: Conversión implícita

1. Declara dos variables: un `int` y un `double`.
2. Asigna el valor del `int` al `double` y muestra el resultado.

Ejercicio 2: Conversión explícita

1. Declara un `double` con un valor decimal.
2. Convierte este valor a un `int` usando casting.
3. Imprime el resultado y explica qué sucede con la parte decimal.

Ejercicio 3: Conversión entre `String` e `int`

1. Declara una variable `String` que contenga un número.
2. Convierte este valor a un `int` y realiza una operación matemática.
3. Convierte un `int` a `String` y concaténalo con otro texto.

Ejercicio 4: Autoboxing y Unboxing

1. Declara una variable `int` y convierte automáticamente a `Integer` (autoboxing).
2. Convierte de `Integer` a `int` (unboxing) y realiza una operación matemática.

8. Preguntas de evaluación

1. ¿Qué tipo de conversión ocurre automáticamente en Java?

a) Implícita
b) Explícita
c) Casting
d) Ninguna

2. ¿Qué palabra clave se usa para realizar una conversión explícita?

a) `convert`
b) `(tipo)`
c) `as`
d) No se necesita palabra clave

3. ¿Qué sucede si conviertes un `double` a un `int` usando casting?

a) Se redondea automáticamente.

b) Se trunca la parte decimal.

c) Se lanza un error.

d) No sucede nada, conserva el valor original.

4. ¿Cuál es el método para convertir un `String` a `int`?

a) `Integer.toInt`

b) `String.toInt`

c) `Integer.parseInt`

d) `String.valueOf`

5. ¿Qué término describe la conversión automática de un `int` a `Integer`?

a) Boxing

b) Autoboxing

c) Unboxing

d) Casting

Parte 10: Comentarios

Los **comentarios** son textos dentro del código que no son ejecutados por el programa. Su propósito principal es mejorar la legibilidad y la comprensión del código, tanto para el programador como para otros colaboradores.

1. ¿Qué son los comentarios?

1. Definición:

- Son notas explicativas dentro del código.
- Ayudan a documentar el propósito, funcionamiento o contexto de una sección del programa.

2. Tipos de comentarios en Java:

- **Comentarios de línea única**: Comienzan con `//`.
- **Comentarios de múltiples líneas**: Están delimitados por `/*` y `/`.
- **Comentarios de documentación**: Usan `/**` y se utilizan para generar documentación con herramientas como Javadoc.

2. Comentarios de línea única

Se utilizan para agregar notas rápidas o explicar una línea específica del código.

Ejemplo:

```
public class EjemploComentarios {
    public static void main(String[] args) {
          int numero = 10; // Declaración e inicialización de una variable
          System.out.println(numero); // Imprime el número
    }
}
```

3. Comentarios de múltiples líneas

Son útiles para describir bloques de código más grandes o agregar notas detalladas.

Ejemplo:

```
public class EjemploComentarios {
    public static void main(String[] args) {
          /*
           * Este bloque de código inicializa una variable,
           * imprime su valor, y realiza una operación básica.
           */
          int numero = 10;
          System.out.println(numero);
    }
}
```

4. Comentarios de documentación (Javadoc)

Se usan para documentar clases, métodos y atributos, facilitando la generación de documentación técnica mediante herramientas como Javadoc.

Ejemplo:

```
/**
 * Clase que representa un ejemplo básico de comentarios.
 */
public class EjemploComentarios {
    /**
     * Método principal del programa.
     * @param args Argumentos de línea de comandos.
     */
    public static void main(String[] args) {
          int numero = 10;
          System.out.println(numero); // Imprime el número
    }
}
```

Generar documentación Javadoc:

- Usa el comando: `javadoc EjemploComentarios.java`
- Se crea un archivo HTML con la documentación generada automáticamente.

5. Buenas prácticas al usar comentarios

1. Escribe comentarios significativos:

- Explica por qué algo está sucediendo, no qué está haciendo.
- **Ejemplo malo**:

```
// Suma dos números
resultado = a + b;
```

- **Ejemplo bueno**:

```
// Calcula el total con impuestos incluidos
resultado = a + b;
```

2. Evita comentarios redundantes:

- El código claro no necesita comentarios excesivos.
- **Ejemplo malo**:

```
int numero = 5; // Asigna 5 a la variable numero
```

3. Actualiza los comentarios junto con el código:

- Los comentarios desactualizados generan confusión.

4. Usa comentarios para marcar secciones importantes:

- Como inicialización, cálculos, validaciones, etc.

6. Ejemplo completo

Este programa utiliza los tres tipos de comentarios:

```java
/**
 * Clase que realiza operaciones básicas con números.
 */
public class Operaciones {
    public static void main(String[] args) {
        // Declaración de variables
        int numero1 = 10;
        int numero2 = 20;
        /*
         * Calcula la suma de dos números
         * y la imprime en pantalla.
         */
        int suma = numero1 + numero2;
        System.out.println("La suma es: " + suma); // Imprime el resultado
    }
}
```

Salida:

```
La suma es: 30
```

7. Ejercicios prácticos

Ejercicio 1: Comentarios de línea

Escribe un programa que imprima tu nombre y edad. Usa comentarios de línea para describir cada paso.

Ejercicio 2: Comentarios de múltiples líneas

Crea un programa que:

1. **Declare dos números.**
2. **Realice las operaciones básicas (+, -, *, /).**
3. **Use comentarios de múltiples líneas para explicar el propósito del programa.**

Ejercicio 3: Comentarios Javadoc

1. **Crea una clase llamada `Persona` con atributos `nombre` y `edad`.**
2. **Documenta la clase y los métodos usando comentarios Javadoc.**
3. **Genera la documentación usando el comando `javadoc`.**

8. Preguntas de evaluación

1. ¿Qué símbolo se usa para comenzar un comentario de línea única en Java?

a) `#`
b) `//`
c) `/*`
d) `/**`

2. ¿Qué tipo de comentario se utiliza para documentar clases y métodos?

a) Comentarios de línea única
b) Comentarios de múltiples líneas
c) Comentarios Javadoc
d) Ninguno de los anteriores

3. ¿Qué herramienta genera documentación a partir de comentarios en Java?

a) DocGenerator
b) Javadoc
c) JavaComments
d) DocumentationBuilder

4. ¿Qué ocurre con los comentarios al compilar un programa?

a) Se ejecutan como parte del código.
b) Se eliminan del programa compilado.
c) Generan un archivo separado con las notas.
d) Generan errores si no están bien formateados.

5. ¿Cuál es el propósito principal de los comentarios?

a) Optimizar el rendimiento del código.
b) Mejorar la comprensión y documentación del código.
c) Crear variables temporales.
d) Depurar el programa.

Conclusión del Bloque

Identificación de los elementos de un programa informático

En este bloque, hemos explorado los fundamentos necesarios para comprender la estructura y los componentes esenciales de un programa en Java. Al dominar estos conceptos, los estudiantes obtienen una base sólida para escribir código limpio, eficiente y comprensible.

1. Puntos Clave del Bloque

1. Estructura y Bloques Fundamentales:

- Un programa en Java se organiza en paquetes, clases, métodos y bloques de código.
- El método `main` es el punto de entrada del programa.

2. Identificadores:

- Los nombres de clases, variables y métodos deben seguir ciertas reglas y buenas prácticas para garantizar legibilidad y evitar errores.

3. Palabras Reservadas:

- Son términos predefinidos que no pueden usarse como identificadores porque tienen funciones específicas en el lenguaje.

4. Variables:

- Representan datos almacenados en memoria y pueden ser de diferentes tipos (primitivos o no primitivos).
- Deben declararse y, preferentemente, inicializarse antes de usarse.

5. Tipos de Datos:

- Definen qué tipo de información puede almacenar una variable, desde números y texto hasta objetos más complejos.

6. Literales:

- Son valores constantes escritos directamente en el código, como números, texto o booleanos.

7. Constantes:

- Son valores inmutables que mejoran la legibilidad y la seguridad del código, especialmente en configuraciones.

8. Operadores y Expresiones:

- Permiten realizar cálculos, comparaciones y operaciones lógicas en el programa.
- La precedencia de operadores garantiza el orden correcto de evaluación.

9. Conversiones de Tipo:

- Las conversiones implícitas simplifican el manejo de datos compatibles, mientras que las explícitas requieren cuidado para evitar pérdida de precisión.

10. Comentarios:

- Ayudan a documentar el código para facilitar su comprensión y mantenimiento.

Con este conocimiento, los estudiantes están listos para abordar problemas más complejos, escribir código estructurado y comprender cómo interactúan los elementos básicos de un programa.

2. Ejercicio Guiado: Crear un Programa Completo

Objetivo

Desarrollar un programa que integre los conceptos clave del bloque, desde la declaración de variables y constantes hasta el uso de operadores, conversiones y comentarios.

Paso 1: Plantear el Programa

Escribe un programa que:

1. Solicite al usuario su nombre y edad.

2. Calcule cuántos años tendrá en 10 años.

3. Verifique si la edad actual es mayor a 18.

4. Use una constante para definir el límite de edad (18 años).

5. Documente cada paso con comentarios.

Paso 2: Código Base

Crea el archivo `MiPrograma.java` con la estructura básica del programa.

```
public class MiPrograma {
    public static void main(String[] args) {
        // Aquí irá el código
    }
}
```

Paso 3: Declarar Constantes y Variables

Define una constante para la mayoría de edad y variables para almacenar los datos del usuario.

```
  public class MiPrograma {
     public static final int MAYORIA_EDAD = 18; // Constante para mayoría de edad
     public static void main(String[] args) {
          String nombre; // Variable para el nombre
          int edad;    // Variable para la edad
     }
  }
```

Paso 4: Usar el Scanner para Leer Datos del Usuario

Agrega el código para leer el nombre y la edad desde la consola.

```
  import java.util.Scanner;
  public class MiPrograma {
     public static final int MAYORIA_EDAD = 18;
     public static void main(String[] args) {
          Scanner scanner = new Scanner(System.in);
          // Solicitar el nombre
          System.out.print("Introduce tu nombre: ");
          String nombre = scanner.nextLine();
          // Solicitar la edad
          System.out.print("Introduce tu edad: ");
          int edad = scanner.nextInt();
     }
  }
```

Paso 5: Calcular Edad Futura

Agrega el cálculo de la edad en 10 años.

```
  int edadEnDiezAnios = edad + 10; // Calcula la edad futura
  System.out.println(nombre + ", en 10 años tendrás " + edadEnDiezAnios + " años.");
```

Paso 6: Verificar Mayoría de Edad

Usa un operador relacional para verificar si el usuario es mayor de edad.

```
  if (edad >= MAYORIA_EDAD) {
      System.out.println("Eres mayor de edad.");
  } else {
      System.out.println("Eres menor de edad.");
  }
```

Paso 7: Documentar con Comentarios

Agrega comentarios explicando cada paso.

```
  import java.util.Scanner;
  /**
   * Programa que solicita el nombre y la edad del usuario,
   * calcula su edad en 10 años y verifica si es mayor de edad.
   */
  public class MiPrograma {
      public static final int MAYORIA_EDAD = 18; // Constante para mayoría de
edad
      public static void main(String[] args) {
            // Crear un objeto Scanner para leer datos del usuario
            Scanner scanner = new Scanner(System.in);
            // Solicitar el nombre del usuario
            System.out.print("Introduce tu nombre: ");
            String nombre = scanner.nextLine();
            // Solicitar la edad del usuario
            System.out.print("Introduce tu edad: ");
            int edad = scanner.nextInt();
            // Calcular la edad en 10 años
            int edadEnDiezAnios = edad + 10;
            System.out.println(nombre + ", en 10 años tendrás " + edadEnDiezA-
nios + " años.");
            // Verificar si es mayor de edad
            if (edad >= MAYORIA_EDAD) {
                  System.out.println("Eres mayor de edad.");
            } else {
                  System.out.println("Eres menor de edad.");
            }
      }
  }
```

Salida del Programa

Entrada:

```
Introduce tu nombre: Juan
Introduce tu edad: 20
```

Salida:

```
Juan, en 10 años tendrás 30 años.
Eres mayor de edad.
```

3. Reflexión Final

Este ejercicio guía al estudiante a:

1. Combinar múltiples conceptos del bloque en un solo programa.
2. Usar comentarios para documentar claramente el código.
3. Aplicar constantes, variables, operadores y estructuras de control.
4. Practicar la lectura de datos desde la consola con el objeto `Scanner`.