Unidad 1

# Introducción a la programación

## Contenidos

# *Parte 1: Datos*

## 1. Introducción a los datos

Los **datos** son piezas fundamentales en el mundo de la programación. Todo lo que hace un programa se reduce a procesar datos: recibirlos, manipularlos, analizarlos y producir resultados.

### ¿Qué son los datos?

Los datos son piezas de información que representan algo en el mundo real. En programación, estos datos se representan mediante valores que pueden ser almacenados y procesados por una computadora. Por ejemplo:

- El nombre de una persona.
- La temperatura en una ciudad.
- La puntuación de un jugador en un videojuego.

### Ejemplo cotidiano:

Piensa en un formulario de inscripción para una clase de baile:

- **Datos requeridos**: Tu nombre, edad, número de teléfono, nivel de experiencia.
- Estos datos se usan para:
    - Saber quién eres (nombre).
    - Ver si cumples con la edad mínima (edad).
    - Contactarte (teléfono).
    - Asignarte al nivel adecuado (nivel de experiencia).

En programación, estos datos se capturan, almacenan y manipulan para ofrecer soluciones automatizadas.

## 2. Tipos de datos en Java

En Java, los datos deben tener un **tipo definido**. Esto le dice a la computadora qué clase de información se está manejando y qué operaciones se pueden realizar con ella.

### 2.1. Tipos de datos primitivos

Los tipos primitivos son los bloques de construcción básicos de Java. Estos son simples, eficientes y esenciales para cualquier programa. Hay 8 tipos primitivos en Java:

| Tipo | Tamaño | Descripción | Ejemplo |
|---|---|---|---|
| `byte` | 1 byte | Números pequeños entre -128 y 127. | 100 |
| `short` | 2 bytes | Números entre -32,768 y 32,767. | 15000 |
| `int` | 4 bytes | Números enteros más grandes. | 1, 200, -15 |
| `long` | 8 bytes | Números enteros muy grandes. | 1000000000L |
| `float` | 4 bytes | Números decimales de precisión simple | 3.14f |
| `double` | 8 bytes | Números decimales de precisión doble. | 3.14159 |
| `char` | 2 bytes | 2 bytes \| Un solo carácter Unicode. | 'A', '9' \| |
| `boolean` | 1 bit | Valores de verdadero (`true`) o falso (`false`). | true |

**Ejemplo práctico (tipos primitivos):**

```
public class TiposPrimitivos {
    public static void main(String[] args) {
        // Enteros
        int edad = 25;
        long poblacionMundial = 7800000000L;
        // Decimales
        float piAproximado = 3.14f;
        double piPreciso = 3.14159265359;
        // Caracteres
        char inicial = 'J';
        // Booleanos
        boolean esProgramador = true;
        // Mostrar en pantalla
        System.out.println("Edad: " + edad);
        System.out.println("Población Mundial: " + poblacionMundial);
        System.out.println("Valor de PI (float): " + piAproximado);
        System.out.println("Valor de PI (double): " + piPreciso);
        System.out.println("Inicial: " + inicial);
        System.out.println("¿Es programador? " + esProgramador);
    }
}
```

### 2.2. Tipos de datos no primitivos (Referenciados)

Los tipos no primitivos son más complejos y pueden almacenar múltiples valores o estructuras más avanzadas.

- **Cadenas de texto (`String`)**:
  - Representan palabras o frases.
  - Ejemplo: `"Hola, mundo"`.
- **Arreglos (`Array`)**:
  - Contienen colecciones de elementos del mismo tipo.
  - Ejemplo: `[1, 2, 3]` o `["rojo", "azul"]`.
- **Clases y objetos**:
  - Definidos por el programador para modelar datos más complejos.
  - Ejemplo: Un objeto `Persona` que contenga nombre, edad y dirección.

**Ejemplo práctico (String y Arrays):**

```
public class DatosCompuestos {
    public static void main(String[] args) {
        // Cadenas de texto
        String saludo = "¡Hola, mundo!";
        String nombre = "Carlos";
        // Arreglo de números
        int[] numeros = {1, 2, 3, 4, 5};
        // Arreglo de colores
        String[] colores = {"Rojo", "Verde", "Azul"};
        // Mostrar en pantalla
        System.out.println(saludo);
        System.out.println("Nombre: " + nombre);
        System.out.println("Primer número: " + numeros[0]);
        System.out.println("Primer color: " + colores[0]);
    }
}
```

## 3. Operaciones con datos

Manipular datos es fundamental en programación. Algunas operaciones clave incluyen:

### 3.1. Operaciones matemáticas

- **Suma** (`+`): Une dos números.
- **Resta** (`-`): Encuentra la diferencia.
- **Multiplicación** (`*`): Encuentra el producto.
- **División** (`/`): Divide un número entre otro.

**Ejemplo práctico (matemáticas):**

```
public class Matematicas {
    public static void main(String[] args) {
        int a = 8, b = 3;
        System.out.println("Suma: " + (a + b));
        System.out.println("Resta: " + (a - b));
        System.out.println("Multiplicación: " + (a * b));
        System.out.println("División: " + (a / b));
    }
}
```

### 3.2. Concatenación de cadenas

Une textos utilizando el operador `+`.

**Ejemplo práctico (concatenación):**

```
public class Concatenacion {
    public static void main(String[] args) {
        String nombre = "Ana";
        String saludo = "Hola, " + nombre;
        System.out.println(saludo);
    }
}
```

## 4. Referencias con ejemplos cotidianos

- **Datos en la vida diaria**:
  - En una biblioteca: Los libros tienen datos como título, autor, y género.
  - En un supermercado: Los productos tienen datos como precio, peso y código de barras.

## 5. Ejercicios

**1. Crea un programa que calcule la suma de dos números enteros proporcionados por el usuario.**

**2. Escribe un programa que pida al usuario su nombre y edad, y luego imprima un mensaje que diga "Hola, [nombre], tienes [edad] años."**

## 6. Test de evaluación

**1. ¿Qué tipo de dato usarías para representar una letra?**

a) int

b) char

c) String

**2. ¿Cuál es el resultado de la operación 10 / 3 en Java con enteros?**

a) 3.33

b) 3

c) Error

**3. ¿Qué operador se usa para concatenar cadenas en Java?**

a) +

b) -

c) *

# *Parte 2: Algoritmos*

## 1. ¿Qué es un algoritmo?

Un **algoritmo** es una secuencia finita, ordenada y lógica de pasos que describe cómo resolver un problema o completar una tarea. Es la base del pensamiento computacional y el núcleo de cualquier programa de software.

**Imagina esto:** Cuando buscas algo en Internet, como "¿Cuál es la capital de Francia?", tu navegador ejecuta un algoritmo que procesa tu búsqueda, encuentra la respuesta (París) y la muestra en pantalla. Este es un ejemplo práctico de un algoritmo en acción.

### Definición técnica

Un algoritmo debe cumplir ciertas condiciones para ser considerado como tal:

1. **Precisión: Cada paso debe estar claramente definido.**
2. **Finitud: El algoritmo debe terminar después de un número definido de pasos.**
3. **Definición: Para una misma entrada, siempre producirá el mismo resultado.**
4. **Entrada y salida: Debe recibir datos iniciales (entrada) y generar resultados (salida).**
5. **Eficiencia: La cantidad de pasos y recursos necesarios para completarlo debe ser razonable.**

### Ejemplo cotidiano de un algoritmo

Hacer un té es un algoritmo simple:

1. **Calentar agua.**
2. **Poner una bolsita de té en una taza.**
3. **Verter el agua caliente sobre la bolsita.**
4. **Esperar 5 minutos.**
5. **Retirar la bolsita y agregar azúcar, si se desea.**

En este caso:

- **Entrada**: Agua, taza, bolsita de té, azúcar (opcional).
- **Proceso**: Calentar, mezclar, esperar.
- **Salida**: Una taza de té lista para beber.

## 2. ¿Por qué son importantes los algoritmos?

Los algoritmos son fundamentales porque:

**1. Automatizan procesos: Permiten a las computadoras realizar tareas repetitivas o complejas.**
**2. Resuelven problemas: Desde calcular rutas en Google Maps hasta sugerir películas en Netflix.**
**3. Optimización: Encuentran la forma más eficiente de realizar tareas.**
**4. Adaptabilidad: Los mismos principios se aplican a diferentes problemas y áreas.**

## 3. Componentes de un algoritmo

Todo algoritmo se compone de tres partes esenciales:

### 1. Entrada

Son los datos necesarios para que el algoritmo comience. Sin entrada, el algoritmo no tendría contexto para funcionar.

**Ejemplo:**

- Para calcular el área de un rectángulo, necesitas las dimensiones: base y altura.
- Entrada: `base = 5` y `altura = 10`.

### 2. Proceso

Es el conjunto de operaciones que transforma los datos de entrada en resultados. Estas operaciones pueden incluir:

- Cálculos matemáticos.
- Comparaciones lógicas.
- Repetición de pasos (bucles).

**Ejemplo:**

- Multiplicar la base por la altura.
- Proceso: `área = base * altura`.

### 3. Salida

Es el resultado final obtenido después de procesar los datos.

**Ejemplo:**

- Salida: El área del rectángulo es 50.

## 4. ¿Cómo se representan los algoritmos?

Antes de convertir un algoritmo en código, es fundamental planificar y estructurarlo. Las dos formas más comunes de representar un algoritmo son:

### 1. Pseudocódigo

El pseudocódigo es una descripción textual que combina lenguaje natural y términos de programación. No sigue las reglas de un lenguaje de programación específico, pero facilita la traducción a cualquier lenguaje.

**Ejemplo de pseudocódigo para determinar si un número es par o impar:**

```
Inicio
   Leer número
   Si (número % 2 == 0) entonces
      Escribir "El número es par"
   Si no
      Escribir "El número es impar"
Fin
```

### 2. Diagramas de flujo

Un diagrama de flujo utiliza símbolos gráficos para mostrar la lógica de un algoritmo:

- **Óvalo**: Inicio o fin.
- **Rectángulo**: Proceso.
- **Rombo**: Decisión.
- **Flechas**: Flujo del proceso.

## 5. Algoritmos en la vida diaria

Los algoritmos no son exclusivos de la programación. Muchas actividades cotidianas se pueden descomponer en algoritmos. Veamos ejemplos:

**Ejemplo 1: Preparar un sándwich**

**1. Tomar dos rebanadas de pan.**

**2. Untar una con mantequilla y otra con mermelada.**

**3. Juntar ambas rebanadas.**

**4. Servir.**

**Ejemplo 2: Decidir qué ropa usar**

**1. Verificar la temperatura.**

**2. Si hace frío:**

- Elegir ropa abrigada.

**3. Si hace calor:**

- Elegir ropa ligera.

Estos ejemplos muestran cómo los algoritmos estructuran decisiones y acciones en cualquier contexto.

## 6. Algoritmos en programación

En programación, los algoritmos resuelven problemas utilizando instrucciones que una computadora puede entender y ejecutar. Vamos a ver algunos ejemplos básicos en Java.

**Ejemplo práctico 1: Determinar si un número es par o impar**

```
import java.util.Scanner;
public class ParOImpar {
    public static void main(String[] args) {
          Scanner scanner = new Scanner(System.in);
          // Entrada
          System.out.print("Introduce un número entero: ");
          int numero = scanner.nextInt();
          // Proceso y salida
          if (numero % 2 == 0) {
                System.out.println("El número es par.");
          } else {
                System.out.println("El número es impar.");
          }
    }
}
```

**Ejemplo práctico 2: Calcular el promedio de tres números**

```
import java.util.Scanner;
public class Promedio {
    public static void main(String[] args) {
          Scanner scanner = new Scanner(System.in);
          // Entrada
          System.out.print("Introduce el primer número: ");
          int num1 = scanner.nextInt();
          System.out.print("Introduce el segundo número: ");
          int num2 = scanner.nextInt();
          System.out.print("Introduce el tercer número: ");
          int num3 = scanner.nextInt();
          // Proceso
          int suma = num1 + num2 + num3;
          double promedio = suma / 3.0;
          // Salida
          System.out.println("El promedio es: " + promedio);
    }
}
```

## 7. Ejercicios para practicar

1. **Escribe un algoritmo en pseudocódigo que determine si un número ingresado por el usuario es positivo, negativo o cero.**
2. **Implementa un programa en Java que calcule el área de un triángulo dado su base y altura.**
3. **Crea un algoritmo para decidir si una persona puede votar, basándote en su edad.**

## 8. Test de comprensión

**1. ¿Qué parte de un algoritmo representa el resultado final?**

a) Entrada.

b) Proceso.

c) Salida.

**2. ¿Qué es un diagrama de flujo?**

a) Un lenguaje de programación.

b) Una representación gráfica de un algoritmo.

c) Un programa que ejecuta instrucciones.

**3. ¿Cuál de los siguientes NO es una característica de un algoritmo?**

a) Precisión.

b) Finitud.

c) Aleatoriedad.

# *Parte 3: Lenguajes de programación*

## 1. ¿Qué es un lenguaje de programación?

Un **lenguaje de programación** es un conjunto de reglas, símbolos y palabras clave que permiten a los programadores escribir instrucciones que una computadora puede entender y ejecutar.

Las computadoras no entienden directamente el lenguaje humano. Solo entienden el **código máquina**, que está compuesto de ceros y unos (binario). Los lenguajes de programación actúan como un puente entre los seres humanos y las computadoras, traduciendo nuestras ideas en un formato que las máquinas pueden procesar.

**Ejemplo cotidiano**:

Imagina que estás en un país donde se habla un idioma diferente al tuyo. Necesitas un traductor que convierta tus palabras a ese idioma. El lenguaje de programación es como ese traductor, que convierte las instrucciones del programador en un lenguaje que la computadora puede entender.

## 2. Clasificación de los lenguajes de programación

Los lenguajes de programación se pueden clasificar de varias maneras. Una de las formas más comunes es según su **nivel de abstracción**:

### 2.1. Lenguajes de bajo nivel

Estos lenguajes están más cerca del lenguaje máquina (binario). Son más difíciles de entender para los humanos, pero extremadamente eficientes para la computadora.

- **Ejemplo**: Lenguaje ensamblador.
- **Ventajas**:
  - Alta eficiencia.
  - Control directo sobre el hardware.
- **Desventajas**:
  - Difíciles de aprender y escribir.
  - Menos portables (dependen del hardware).

**Ejemplo de lenguaje ensamblador:**

```
MOV AX, 5
ADD AX, 3
```

Esto indica a la computadora que cargue el valor 5 en un registro llamado AX y luego le sume 3.

### 2.2. Lenguajes de alto nivel

Están diseñados para ser más comprensibles para los humanos. Usan palabras clave y sintaxis que se asemejan al lenguaje natural.

- **Ejemplo**: Java, Python, C++, etc.
- **Ventajas**:
  - Más fáciles de aprender y usar.
  - Portables (pueden ejecutarse en diferentes plataformas).
- **Desventajas**:
  - Menor control sobre el hardware.
  - Menor eficiencia en comparación con los lenguajes de bajo nivel.

**Ejemplo en Java (lenguaje de alto nivel):**

```
int suma = 5 + 3;
System.out.println("La suma es: " + suma);
```

Este código realiza la misma operación que el ejemplo de ensamblador, pero es mucho más fácil de entender.

### 2.3. Lenguajes compilados y lenguajes interpretados

Otra forma de clasificar los lenguajes de programación es según cómo se ejecutan:

**1. Lenguajes compilados:**

- El código fuente (lo que escribes) se traduce a código máquina mediante un programa llamado **compilador**.
- Una vez compilado, el programa puede ejecutarse directamente.
- Ejemplos: C, C++, Java.

**2. Lenguajes interpretados:**

- El código se ejecuta línea por línea mediante un programa llamado **intérprete**.
- Ejemplos: Python, JavaScript.

**Diferencias clave:**

| Característica | Compilados | Interpretados |
|---|---|---|
| Velocidad | Más rápidos | Más lentos |
| Portabilidad | Depende del compilador | Generalmente alta |
| Proceso de ejecución | Requiere compilación previa | Ejecución inmediata |

## 3. ¿Por qué elegir Java?

Java es uno de los lenguajes de programación más populares y ampliamente utilizados en el mundo. Tiene varias características que lo hacen ideal para aprender y desarrollar aplicaciones.

### Ventajas de Java:

**1. Portabilidad:**

- Gracias a la máquina virtual de Java (JVM), el mismo programa puede ejecutarse en diferentes sistemas operativos.
- **Lema**: "Escribe una vez, ejecuta en cualquier lugar."

**2. Orientado a objetos:**

- Java organiza los programas en objetos, lo que facilita la escritura y el mantenimiento del código.

**3. Gran comunidad:**

- Al ser tan popular, hay una enorme cantidad de recursos, tutoriales y foros disponibles.

**4. Seguro:**

- Incluye características que reducen los riesgos de errores o vulnerabilidades en el código.

**5. Versatilidad:**

- Puedes usar Java para aplicaciones de escritorio, móviles, web y más.

### Desventajas de Java:

- Es más lento que algunos lenguajes como C++.
- Su sintaxis puede ser un poco complicada para principiantes en comparación con Python.

## 4. Sintaxis básica de Java

Java tiene una sintaxis clara pero estricta. Esto significa que debes seguir reglas específicas para que el código funcione correctamente.

### Estructura básica de un programa en Java

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

**Desglose del código:**

1. `public class MiPrograma`: **Define una clase llamada `MiPrograma`. En Java, todo el código debe estar dentro de una clase.**
2. `public static void main(String[] args)`: **Este es el punto de entrada del programa. Es donde comienza la ejecución.**
3. `System.out.println("¡Hola, mundo!");`: **Imprime el texto en la pantalla.**

## 5. Ejemplo práctico: Crear tu primer programa

Vamos a escribir un programa que sume dos números:

1. **Problema: El usuario ingresa dos números y el programa calcula la suma.**
2. **Entrada: Dos números.**
3. **Proceso: Sumar los números.**
4. **Salida: Mostrar la suma.**

**Código en Java**:

```
import java.util.Scanner;
public class SumaNumeros {
    public static void main(String[] args) {
          Scanner scanner = new Scanner(System.in);
          // Entrada
          System.out.print("Introduce el primer número: ");
          int num1 = scanner.nextInt();
          System.out.print("Introduce el segundo número: ");
          int num2 = scanner.nextInt();
          // Proceso
          int suma = num1 + num2;
          // Salida
          System.out.println("La suma de los números es: " + suma);
    }
}
```

## 6. Ejercicios para practicar

**1. Escribe un programa en Java que calcule el área de un círculo.**

- Entrada: Radio del círculo.
- Proceso: Calcular el área usando la fórmula `π * r^2`.
- Salida: Mostrar el área.

**2. Crea un programa que convierta grados Celsius a Fahrenheit.**

- Entrada: Grados Celsius.
- Proceso: Usar la fórmula `F = C × 9/5 + 32`.
- Salida: Mostrar los grados Fahrenheit.

**3. Escribe un programa que determine si un número ingresado por el usuario es positivo, negativo o cero.**

## 7. Test de comprensión

**1. ¿Cuál de los siguientes es un lenguaje de alto nivel?**

a) Ensamblador.
b) Java.
c) Código máquina.

**2. ¿Qué característica describe mejor a los lenguajes compilados?**

a) Se ejecutan línea por línea.
b) Son más rápidos en ejecución.

- c) No necesitan un compilador.

**3. ¿Qué hace la JVM en Java?**

a) Traduce el código fuente a código máquina.
b) Permite que los programas sean portables.
c) Ambas respuestas son correctas.

# *Parte 4: Herramientas y entornos para el desarrollo de programas*

## 1. Introducción

El desarrollo de programas no solo requiere conocimientos de programación, sino también un entorno adecuado para escribir, probar y ejecutar el código. Estas herramientas simplifican el trabajo del programador y mejoran la productividad.

En esta sección, aprenderás:

**1. Qué herramientas necesitas para programar en Java.**

**2. Cómo instalar y configurar un entorno de desarrollo.**

**3. Las funciones básicas de las herramientas más comunes.**

## 2. Componentes esenciales del entorno de desarrollo

Para empezar a programar en Java, necesitas configurar algunas herramientas clave:

### 2.1. Java Development Kit (JDK)

El **JDK** es el kit de desarrollo oficial de Java. Contiene todo lo necesario para escribir, compilar y ejecutar programas en Java.

**Incluye**:

- **Compilador**: Convierte el código fuente de Java en bytecode, que es el formato que entiende la JVM.
- **Máquina Virtual de Java (JVM)**: Ejecuta los programas de Java.
- **Librerías estándar**: Conjunto de clases y funciones listas para usar.

**Pasos para instalar el JDK**:

**1. Visita la página oficial de [Oracle Java](https://www.oracle.com/java/technologies/javase-downloads.html) o adopta alternativas como OpenJDK.**

**2. Descarga la versión adecuada para tu sistema operativo.**

**3. Sigue las instrucciones del instalador.**

### 2.2. Un editor de texto o IDE

Un **editor de texto** es donde escribirás tu código. Hay opciones simples y avanzadas, según tus necesidades:

- **Editores de texto básicos**:
  - **Notepad++ (Windows)**: Ligero y fácil de usar.
  - **Visual Studio Code (Multiplataforma)**: Personalizable con extensiones.
- **Entornos de Desarrollo Integrado (IDE)**:

Los **IDEs** son herramientas más completas que incluyen características como:

- Resaltado de sintaxis.
- Compilación automática.
- Depuración de código.

**IDE recomendados para Java**:

**1. Eclipse:**

- Gratuito y ampliamente utilizado.
- Excelente para principiantes.
- Página oficial: Eclipse Downloads.

**2. IntelliJ IDEA:**

- Intuitivo y poderoso.
- Versión gratuita y de pago.
- Página oficial: [IntelliJ IDEA](https://www.jetbrains.com/idea/).

**3. NetBeans:**

- Desarrollado por Oracle.
- Incluye herramientas integradas para Java.
- Página oficial: [NetBeans](https://netbeans.apache.org/).

### 2.3. Terminal o consola

La terminal es una herramienta básica pero poderosa para ejecutar programas, compilar manualmente o diagnosticar errores. Es útil cuando trabajas con configuraciones básicas sin un IDE.

## 3. Configuración paso a paso para principiantes

#### Paso 1: Instalar el JDK

- Descarga el JDK desde el sitio oficial.
- Durante la instalación, asegúrate de que el instalador configure la variable de entorno `JAVA_HOME`. Esto permite que otros programas detecten automáticamente la ubicación de Java.

#### Paso 2: Instalar un IDE o editor de texto

Elige un IDE (por ejemplo, Eclipse) y sigue las instrucciones del instalador. Si prefieres algo más simple, usa Visual Studio Code y agrega la extensión para Java.

**Paso 3: Configurar y probar**

**1. Abre tu editor o IDE.**

**2. Escribe el siguiente programa básico:**

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

**3. Compila y ejecuta el programa:**

- En un IDE: Haz clic en el botón "Run" o "Ejecutar".
- En la terminal:
    1. Guarda el archivo como `HolaMundo.java`.
    2. Compila con el comando: `javac HolaMundo.java`.
    3. Ejecuta con: `java HolaMundo`.

Si ves el mensaje "¡Hola, mundo!", tu entorno está configurado correctamente.

## 4. Herramientas adicionales

### 4.1. Control de versiones: Git y GitHub

- **Git**:
    - Sistema de control de versiones que registra los cambios en tu código.
    - Útil para colaborar con otros programadores y evitar perder tu progreso.
    - Descarga Git en: [git-scm.com](https://git-scm.com/).
- **GitHub**:
    - Plataforma para alojar y compartir tus proyectos usando Git.
    - Crea un repositorio en GitHub y sube tu código para trabajar en equipo.

**Comandos básicos de Git**:

```
bash
Copiar código
git init                # Inicializa un repositorio
git add .              # Agrega archivos al área de preparación
git commit -m “Mensaje” # Registra los cambios
git push origin main  # Sube los cambios a GitHub
```

### 4.2. Herramientas de construcción: Maven y Gradle

En proyectos grandes, herramientas como **Maven** y **Gradle** automatizan tareas repetitivas como la gestión de dependencias o la construcción del proyecto.

- **Maven**:
  - Estructura y organiza proyectos Java.
  - Archivo clave: `pom.xml`.
- **Gradle**:
  - Más flexible que Maven.
  - Archivo clave: `build.gradle`.

## 5. Ejemplo práctico: Configuración y ejecución

Escribe un programa que calcule el cuadrado de un número ingresado por el usuario.

**1. Abre tu IDE y crea un nuevo proyecto.**

**2. Escribe el siguiente código:**

```
java
Copiar código
import java.util.Scanner;
public class Cuadrado {
    public static void main(String[] args) {
          Scanner scanner = new Scanner(System.in);
          System.out.print(“Introduce un número: “);
          int numero = scanner.nextInt();
          int cuadrado = numero * numero;
          System.out.println(“El cuadrado de “ + numero + “ es: “ + cuadrado);
    }
}
```

**3. Ejecuta el programa:**

- En el IDE: Haz clic en “Run”.
- En la terminal: Usa los comandos `javac` y `java`.

## 6. Ejercicios para practicar

**1. Instalación y prueba: Configura el JDK y escribe un programa simple que imprima tu nombre.**

**2. Crea un repositorio en GitHub:**

- Sube tu programa de “¡Hola, mundo!” a GitHub.

**3. Explora tu IDE:**

- Familiarízate con funciones como:
  - Autocompletado.
  - Depuración de código.
  - Navegación por archivos.

## 7. Test de comprensión

**1. ¿Qué es el JDK?**

a) Un editor de texto.
b) Un compilador y entorno para Java.
c) Un sistema operativo.

**2. ¿Qué comando se usa para compilar un archivo Java en la terminal?**

a) java.
b) compile.
c) javac.

**3. ¿Cuál es la función principal de un IDE?**

a) Escribir código en binario.
b) Simplificar el desarrollo con herramientas integradas.
c) Ejecutar comandos en la terminal.

# *Parte 5: Errores y calidad de los programas*

## 1. Introducción

En programación, los errores son inevitables, especialmente cuando estás aprendiendo. La clave para ser un buen programador no es evitar errores, sino aprender a identificarlos, comprenderlos y solucionarlos. Además, garantizar la calidad del código ayuda a minimizar errores y a hacer el software más fácil de mantener.

## 2. Tipos de errores en programación

Los errores en programación se clasifican en tres categorías principales:

### 2.1. Errores de sintaxis

Estos ocurren cuando el código no cumple con las reglas del lenguaje de programación. Son detectados por el compilador antes de que el programa se ejecute.

**Ejemplo de error de sintaxis en Java:**

```
public class Ejemplo {
    public static void main(String[] args) {
        System.out.println(“Hola, mundo!”) // Falta el punto y coma al final.
    }
}
```

**Cómo solucionarlo**:

El compilador mostrará un mensaje de error indicando dónde está el problema. En este caso, falta un punto y coma al final de la línea.

### 2.2. Errores de ejecución

Estos errores ocurren cuando el programa se ejecuta, pero algo sale mal, como intentar dividir un número entre cero o acceder a una posición inexistente en un arreglo.

**Ejemplo de error de ejecución:**

```
public class DivisionPorCero {
    public static void main(String[] args) {
        int a = 10;
        int b = 0;
        System.out.println(a / b); // Esto genera un error porque no se
puede dividir entre cero.
    }
}
```

**Cómo solucionarlo**:

- Añadir validaciones antes de realizar operaciones:

```
if (b != 0) {
    System.out.println(a / b);
} else {
    System.out.println("Error: División por cero.");
}
```

### 2.3. Errores lógicos

Son los más difíciles de detectar, ya que el programa no muestra un error, pero no produce el resultado esperado. Esto ocurre debido a errores en la lógica del algoritmo.

**Ejemplo de error lógico:**

Calcular el promedio de tres números:

```
public class Promedio {
    public static void main(String[] args) {
        int num1 = 10, num2 = 20, num3 = 30;
        int promedio = (num1 + num2 + num3) / 2; // Error: Debe dividirse
entre 3, no 2.
        System.out.println("El promedio es: " + promedio);
    }
}
```

**Cómo solucionarlo**:

- Revisar cuidadosamente la lógica y los cálculos del programa.
- Usar comentarios y depuración para verificar cada paso.

## 3. Cómo identificar y solucionar errores

**1. Leer el mensaje de error:**

- Cuando ocurre un error, el compilador o intérprete genera un mensaje que indica el problema y la línea donde ocurrió.
- Aprende a interpretar estos mensajes para localizar y corregir el error rápidamente.

**2. Usar herramientas de depuración:**

- Los IDEs como Eclipse o IntelliJ IDEA incluyen depuradores que permiten ejecutar el programa paso a paso, inspeccionar valores de variables y encontrar errores.

**3. Escribir código claro y organizado:**

- Usa nombres descriptivos para variables y funciones.
- Divide el código en funciones pequeñas y manejables.
- Comenta el código para explicar su propósito.

**4. Validar entradas:**

- Antes de procesar datos, verifica que sean válidos. Esto previene errores de ejecución.

```
Scanner scanner = new Scanner(System.in);
System.out.print("Introduce un número mayor que 0: ");
int numero = scanner.nextInt();
if (numero > 0) {
    System.out.println("Número válido.");
} else {
    System.out.println("Error: El número debe ser mayor que 0.");
}
```

## 4. Calidad del código

La calidad del código es esencial para garantizar que el programa sea fácil de entender, mantener y escalar. Aquí tienes las mejores prácticas para escribir código de calidad:

### 4.1. Seguir estándares de codificación

Usa convenciones comunes para escribir código, como:

- Nombres de clases en **PascalCase**.
- Nombres de variables y métodos en **camelCase**.
- Indentación consistente.

**Ejemplo de código con buenas prácticas:**

```
public class Calculadora {
    public static void main(String[] args) {
          int numero1 = 10;
          int numero2 = 20;
          int suma = sumar(numero1, numero2);
          System.out.println("La suma es: " + suma);
    }
    public static int sumar(int a, int b) {
          return a + b;
    }
}
```

### 4.2. Documentación

Comenta tu código para explicar lo que hace cada sección. Usa comentarios claros y concisos.

**Ejemplo:**

```
  // Este método calcula la suma de dos números enteros.
  public static int sumar(int a, int b) {
      return a + b;
  }
```

### 4.3. Realizar pruebas

Prueba tu programa en diferentes escenarios para asegurarte de que funcione correctamente.

**Tipos de pruebas**:

- **Pruebas unitarias**: Verifican que cada función individual funcione correctamente.
- **Pruebas de integración**: Comprueban que las diferentes partes del programa trabajen juntas sin problemas.

## 5. Ejemplo práctico: Detectar y solucionar errores

**Problema**: Escribir un programa que calcule el factorial de un número, pero incluye errores comunes.

**Código con errores:**

```
  public class Factorial {
      public static void main(String[] args) {
            int numero = 5;
            int factorial = 0; // Error: El factorial debe comenzar en 1.
            for (int i = 1; i <= numero; i++) {
                  factorial = factorial * i; // Error: Multiplicación con 0
siempre da 0.
            }
            System.out.println("El factorial de " + numero + " es: " + factori-
al);
      }
  }
```

**Corrección:**

```
  public class Factorial {
     public static void main(String[] args) {
           int numero = 5;
           int factorial = 1; // El factorial comienza en 1.
           for (int i = 1; i <= numero; i++) {
                factorial = factorial * i;
           }
           System.out.println("El factorial de " + numero + " es: " + factori-
al);
     }
  }
```

## 6. Ejercicios para practicar

**1. Identificar errores: Revisa el siguiente código y corrige los errores:**

```
  public class Ejercicio {
     public static main(String[] args) { // Falta "void"
           int x = 10;
           System.println(x); // Error en el nombre del método.
     }
  }
```

**2. Validar entradas:**

- Escribe un programa que solicite un número y verifique que esté entre 1 y 100.

**3. Depurar código:**

- Escribe un programa que calcule el promedio de una lista de números, pero incluye errores lógicos. Depura el código para encontrar y corregir los errores.

## 7. Test de comprensión

**1. ¿Qué tipo de error ocurre si olvidas un punto y coma en Java?**

a) Error lógico.

b) Error de sintaxis.

c) Error de ejecución.

**2. ¿Cuál de los siguientes es un error lógico?**

a) Dividir entre cero.

- b) Usar un operador incorrecto en un cálculo.

- c) Olvidar declarar una variable.

**3. ¿Qué herramienta puedes usar para depurar un programa en Java?**

a) Notepad.

b) Eclipse.

c) Compilador.

# *Parte 6: Conclusión del Bloque: Datos, algoritmos y programas*

## 1. Resumen de los conceptos clave

Este bloque abarcó los fundamentos esenciales de la programación, proporcionando una base sólida para continuar aprendiendo Java o cualquier otro lenguaje. Veamos un resumen de los puntos principales:

### 1.1. Datos

- Los datos son la base de cualquier programa.
- Tipos de datos en Java:
  - **Primitivos**: Enteros, decimales, caracteres, booleanos.
  - **No primitivos**: Cadenas (Strings), arreglos, objetos.
- Operaciones comunes:
  - Matemáticas: suma, resta, multiplicación, división.
  - Manipulación de cadenas: concatenación, comparación.

### 1.2. Algoritmos

- Los algoritmos son secuencias de pasos que resuelven problemas.
- Características clave:
  - Precisión, finitud, definición, eficiencia.
- Representación:
  - Pseudocódigo: Descripción textual.
  - Diagramas de flujo: Representación gráfica.
- En programación, los algoritmos se implementan como código.

### 1.3. Lenguajes de programación

- Java es un lenguaje de alto nivel, portable y orientado a objetos.
- Entender su sintaxis básica:
  - Clases, métodos, variables.
  - Punto de entrada (`main`).
- Uso de herramientas como el JDK, IDEs, y sistemas de control de versiones (Git).

### 1.4. Errores y calidad

- Tipos de errores: sintaxis, ejecución, lógicos.
- Buenas prácticas:
  - Validar entradas.
  - Escribir código claro y organizado.
  - Documentar y comentar el código.

## 2. Cómo se conectan estos conceptos

Los conceptos que vimos están interrelacionados y juntos forman la base para desarrollar programas. Veamos cómo funcionan en conjunto:

**1. Identificar el problema:**

- Define el problema que quieres resolver y los datos necesarios.

**2. Diseñar el algoritmo:**

- Piensa en los pasos necesarios para transformar los datos de entrada en resultados.

**3. Implementar en código:**

- Usa un lenguaje de programación, como Java, para escribir el algoritmo.

**4. Probar y depurar:**

- Ejecuta el programa, identifica errores y corrige.

**5. Asegurar la calidad:**

- Mejora la legibilidad, reutilización y eficiencia del código.

## 3. Proyecto práctico: Calculadora interactiva

Vamos a realizar un proyecto práctico que combine todo lo aprendido: una calculadora interactiva que realice operaciones básicas.

### Problema

Escribe un programa en Java que permita al usuario realizar las siguientes operaciones:

**1. Sumar.**

**2. Restar.**

**3. Multiplicar.**

**4. Dividir.**

### Pseudocódigo

**1. Mostrar un menú con las operaciones disponibles.**
**2. Leer la opción elegida por el usuario.**
**3. Pedir los dos números necesarios para realizar la operación.**
**4. Realizar la operación seleccionada.**
**5. Mostrar el resultado.**
**6. Repetir el proceso hasta que el usuario elija salir.**

### Diagrama de flujo

**1. Inicio.**
**2. Mostrar el menú.**
**3. Leer la opción.**

- Si la opción es “Salir”, ir al paso 7.

**4. Leer los números.**
**5. Realizar la operación correspondiente.**
**6. Mostrar el resultado y volver al menú.**
**7. Fin.**

```
import java.util.Scanner;
public class Calculadora {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int opcion;
        do {
            // Mostrar menú
            System.out.println("\n--- Calculadora ---");
            System.out.println("1. Sumar");
            System.out.println("2. Restar");
            System.out.println("3. Multiplicar");
            System.out.println("4. Dividir");
            System.out.println("5. Salir");
            System.out.print("Elige una opción: ");
            opcion = scanner.nextInt();
            // Procesar opción
            if (opcion >= 1 && opcion <= 4) {
                System.out.print("Introduce el primer número: ");
                double num1 = scanner.nextDouble();
                System.out.print("Introduce el segundo número: ");
                double num2 = scanner.nextDouble();
                switch (opcion) {
                    case 1:
                        System.out.println("Resultado: " + (num1 +
num2));
                        break;
                    case 2:
                        System.out.println("Resultado: " + (num1 -
num2));
                        break;
                    case 3:
                        System.out.println("Resultado: " + (num1 *
num2));
                        break;
                    case 4:
                        if (num2 != 0) {
                            System.out.println("Resultado: " + (num1
/ num2));
                        } else {
                            System.out.println("Error: No se puede
dividir entre cero.");
                        }
                        break;
                }
```

```
                } else if (opcion != 5) {
                    System.out.println("Opción no válida. Inténtalo de nue-
vo.");
                }
            } while (opcion != 5);
            System.out.println("Gracias por usar la calculadora. ¡Adiós!");
        }
    }
```

## 4. Ejercicios adicionales

**1. Extiende la calculadora:**

- Añade una opción para calcular potencias (base y exponente).

**2. Crea un sistema de validación:**

- Evita que el usuario ingrese números no válidos.

**3. Mejora la interfaz:**

- Usa mensajes más claros y organiza mejor las salidas.

## 5. Test final del bloque

**1. ¿Cuál de las siguientes NO es una característica de un algoritmo?**

a) Debe ser finito.
b) Debe ser preciso.
c) Debe ser aleatorio.

**2. ¿Qué tipo de error ocurre al intentar dividir entre cero?**

a) Error lógico.
b) Error de ejecución.
c) Error de sintaxis.

**3. ¿Qué comando compila un programa en Java desde la terminal?**

a) `javac`.
b) `java`.
c) `compile`.

**4. ¿Qué herramienta de control de versiones se utiliza comúnmente en proyectos de programación?**

a) Git.
b) Eclipse.
c) JVM.

## 6. Conclusión

Este bloque te ha proporcionado los fundamentos para entender cómo funciona la programación:

- **Datos**: Los elementos básicos que manipulan los programas.
- **Algoritmos**: Los pasos para resolver problemas.
- **Lenguajes de programación**: Las herramientas para implementar algoritmos.
- **Errores y calidad**: La importancia de escribir código confiable y corregir errores.