**1. Características de la Programación Orientada a Objetos (POO)**

La POO es un paradigma que organiza el software en torno a objetos, que combinan datos (atributos) y comportamientos (métodos). Sus principales características son:

**• Encapsulamiento**

Consiste en ocultar los detalles internos de una clase y exponer solo lo necesario mediante métodos públicos (getters y setters).  
**Ventajas**:

* Protege los datos de accesos indebidos o modificaciones accidentales.
* Permite modificar la implementación interna sin afectar al resto del sistema.

public class Persona {

private String nombre;

public String getNombre() {

return nombre;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

}

**• Herencia**

Permite que una clase (subclase) herede atributos y métodos de otra (superclase), promoviendo la **reutilización de código** y una jerarquía clara.

public class Animal {

public void hacerSonido() {

System.out.println("Sonido genérico");

}

}

public class Perro extends Animal {

public void hacerSonido() {

System.out.println("Ladra");

}

}

**• Polimorfismo**

Permite que un mismo método se comporte de manera diferente según el objeto que lo invoque.  
Se puede lograr mediante:

* **Sobrescritura** (@Override) de métodos.
* **Sobrecarga** de métodos con distintos parámetros.

Animal a = new Perro();

a.hacerSonido(); // Imprime: Ladra

**2. Herencia en Java**

**• Tipo de herencia permitida**

Java permite **herencia simple**: una clase solo puede heredar de una única clase (extends).

**• ¿Y la herencia múltiple?**

Java **no permite herencia múltiple de clases**, para evitar ambigüedades. En su lugar, se utilizan **interfaces** para lograr herencia múltiple de comportamiento.

**• Diferencias entre clase abstracta e interfaz**

| **Característica** | **Clase abstracta** | **Interfaz** |
| --- | --- | --- |
| Métodos implementados | Puede tener | Desde Java 8, sí (por defecto) |
| Atributos | Sí, con cualquier modificador | Solo constantes (public static final) |
| Herencia | Una sola clase abstracta | Múltiples interfaces |
| Constructores | Sí | No |

**3. Conceptos básicos de la POO**

**• Clase**

Es un **molde o plantilla** que define los atributos y comportamientos de los objetos.

**• Objeto**

Instancia concreta de una clase. Tiene estado (atributos) y comportamiento (métodos).

**• Atributo**

Variable que representa una propiedad o característica del objeto.

**• Método**

Función definida dentro de una clase que describe un comportamiento del objeto.

**4. Manejo de Excepciones en Java**

**• ¿Qué es una excepción?**

Evento que interrumpe el flujo normal del programa, producido por errores como división por cero o acceso a archivos inexistentes.

**• Bloque try-catch-finally**

try {

// Código que puede lanzar una excepción

} catch (IOException e) {

// Manejo de la excepción

} finally {

// Código que se ejecuta siempre

}

**• throw vs throws**

* throw: lanza una excepción.
* throws: declara que un método puede lanzar excepciones.

public void abrirArchivo() throws IOException {

throw new IOException("Archivo no encontrado");

}

**5. Interfaces gráficas en Java**

**• Bibliotecas gráficas principales**

1. **AWT (Abstract Window Toolkit)**
   * ✅ Básico, parte del JDK
   * ❌ Muy dependiente del sistema operativo
2. **Swing**
   * ✅ Más flexible, multiplataforma
   * ❌ Visualmente menos moderno
3. **JavaFX**
   * ✅ Soporte de CSS, animaciones y FXML
   * ❌ Puede requerir configuración adicional

**• Escuchadores (Listeners)**

Interfaces que **detectan eventos** (como clics o teclas) y ejecutan un bloque de código como respuesta.

button.addActionListener(e -> System.out.println("Botón presionado"));

**• JavaFX: Stage y Scene**

* Stage: ventana principal de la aplicación.
* Scene: representa el contenido visual dentro del Stage.

**6. API de Colecciones en Java**

**• ¿Qué es un genérico?**

Permite trabajar con **tipos parametrizados**, aumentando la seguridad en tiempo de compilación.

List<String> nombres = new ArrayList<>();

**• Tipos principales**

| **Tipo** | **Características** | **Ejemplo de uso** |
| --- | --- | --- |
| List | Ordenado, permite duplicados | Carrito de compras |
| Set | No permite duplicados | Lista de usuarios únicos |
| Map | Pares clave-valor | Agenda de contactos (nombre → número) |

**7. Constructores**

**• ¿Qué es un constructor?**

Método especial que se ejecuta automáticamente al crear un objeto y permite inicializar atributos.

public class Persona {

public Persona(String nombre) {

this.nombre = nombre;

}

}

**• ¿Puede haber múltiples constructores?**

Sí, mediante **sobrecarga** (diferentes firmas):

public Persona() { }

public Persona(String nombre) { }

**• Uso de super()**

Se utiliza para llamar al constructor de la clase padre.

public class Empleado extends Persona {

public Empleado(String nombre) {

super(nombre); // Llama al constructor de Persona

}

}

**8. Sobrescritura de Métodos**

**• ¿Qué es?**

Redefinir un método heredado para modificar su comportamiento en una subclase. Se usa la anotación @Override.

@Override

public void hacerSonido() {

System.out.println("Ladra");

}

**• Diferencia con herencia directa**

* **Heredar**: usar métodos tal como están en la superclase.
* **Sobrescribir**: modificar su comportamiento.

**• Comparar objetos: == vs .equals()**

* == compara referencias (direcciones de memoria).
* .equals() compara el **contenido** del objeto.

String a = new String("hola");

String b = new String("hola");

System.out.println(a == b); // false

System.out.println(a.equals(b)); // true

**9. Relaciones entre Clases**

**• Tipos de relaciones**

| **Tipo** | **Descripción** | **Ejemplo** |
| --- | --- | --- |
| Asociación | Una clase **usa** a otra | Un Profesor enseña a un Alumno |
| Agregación | Relación débil de "todo-parte" | Una Escuela tiene Alumnos |
| Composición | Relación fuerte. Las partes no existen fuera del todo | Una Casa contiene Habitaciones |
| Herencia | Relación "es un" (especialización) | Perro es un Animal |

**• ¿Qué es la cardinalidad?**

Indica cuántas instancias de una clase se relacionan con otra:

* 1:1 → Una persona tiene un DNI.
* 1:N → Un profesor tiene muchos alumnos.
* N:M → Muchos estudiantes cursan muchas materias.

**• Ejemplo simple de asociación:**

class Persona {

String nombre;

}

class Auto {

Persona dueño; // Asociación: el Auto tiene un dueño

}