1. Clases Abstractas

Una clase abstracta es una clase que no puede ser instanciada directamente, sino que sirve como modelo o plantilla para otras clases.

Características principales:

Puede contener métodos abstractos (sin implementación) y métodos concretos (con código).

Se utiliza cuando varias clases comparten atributos o comportamientos comunes, pero requieren que ciertos métodos sean implementados de forma específica en cada subclase.

Permite herencia simple en Java (una clase solo puede heredar de una clase abstracta).

Ejemplo de uso: Una clase abstracta Animal con el método abstracto hacerSonido(), que será implementado de forma distinta en Perro y Gato.

2. Interfaces

Una interfaz es una estructura que define un conjunto de métodos sin implementación. Representa un contrato que las clases que la implementen deben cumplir.

Características principales:

Todos los métodos definidos en una interfaz son abstractos por defecto (a partir de Java 8 también pueden incluir métodos default y static).

Una clase puede implementar múltiples interfaces, lo que permite simular la herencia múltiple.

Se usa para garantizar que diferentes clases tengan un mismo conjunto de comportamientos, aunque no estén relacionadas jerárquicamente.

Ejemplo de uso: Una interfaz Volador con el método volar(), que puede ser implementada tanto por la clase Ave como por la clase Avión.

3. Patrones de Diseño

Los patrones de diseño son soluciones reutilizables y probadas para problemas comunes en el desarrollo de software. Ayudan a mejorar la eficiencia, mantenibilidad y escalabilidad del código.

Clasificación principal:

Patrones creacionales → se centran en la creación de objetos (ej. Singleton, Factory Method, Builder).

Patrones estructurales → se enfocan en cómo organizar clases y objetos para formar estructuras más grandes (ej. Adapter, Decorator, Composite).

Patrones de comportamiento → definen cómo los objetos interactúan y se comunican entre sí (ej. Observer, Strategy, Command).

Ejemplo de uso: El patrón Singleton garantiza que solo exista una instancia de una clase, útil en sistemas de configuración o manejo de base de datos.

Conclusión

Las clases abstractas y las interfaces son pilares fundamentales de la programación orientada a objetos, ya que permiten la reutilización de código, la flexibilidad y la abstracción. Por su parte, los patrones de diseño aportan buenas prácticas que facilitan la solución de problemas comunes de programación, asegurando un software más robusto y fácil de mantener.