**Semana 1: Introducción a los Patrones de Diseño**

**Temas de la sesión**

**¿Qué es un patrón de diseño?**

Los patrones de diseño son soluciones probadas a problemas comunes en el desarrollo de software. No son código específico, sino plantillas o guías que indican cómo estructurar y organizar el código para mejorar su mantenibilidad, reutilización y escalabilidad.

El concepto de patrones de diseño fue formalizado en el libro *"Design Patterns: Elements of Reusable Object-Oriented Software"*, publicado en 1994 por los autores Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, conocidos como el *Gang of Four (GoF)*. Este libro introdujo una clasificación de patrones en **creacionales**, **estructurales** y **de comportamiento**, cubriendo 23 patrones fundamentales.

Los patrones permiten:

* **Reducir el tiempo de desarrollo**, al reutilizar soluciones probadas.
* **Facilitar la comunicación** entre desarrolladores, usando un lenguaje común.
* **Mejorar la mantenibilidad y escalabilidad** del código.

**Beneficios y uso en el desarrollo de software**

El uso de patrones de diseño en software tiene múltiples ventajas:

1. **Estandarización:** Permite que diferentes desarrolladores trabajen con un mismo enfoque estructurado.
2. **Reutilización:** Evita la necesidad de reinventar soluciones para problemas comunes.
3. **Mantenibilidad:** Reduce el acoplamiento y mejora la organización del código.
4. **Flexibilidad:** Facilita la implementación de nuevas funcionalidades sin afectar la estructura existente.

Sin embargo, los patrones de diseño **no siempre deben ser aplicados** indiscriminadamente. Es fundamental entender el problema antes de decidir qué patrón utilizar.

**Clasificación de los patrones**

Los patrones se dividen en tres grandes familias:

1. **Patrones Creacionales** → Enfocados en la instancia y creación de objetos. Ejemplos:
   * Singleton
   * Factory Method
   * Abstract Factory
   * Builder
   * Prototype
2. **Patrones Estructurales** → Organizan la composición de clases y objetos. Ejemplos:
   * Adapter
   * Bridge
   * Composite
   * Decorator
   * Facade
   * Proxy
   * Flyweight
3. **Patrones de Comportamiento** → Gestionan la comunicación y el flujo de datos entre objetos. Ejemplos:
   * Strategy
   * Observer
   * Command
   * Chain of Responsibility
   * State
   * Mediator
   * Memento
   * Template Method
   * Visitor

# Tarea en Clase

**Ejercicios Conceptuales**

1. Explicar los beneficios de los patrones de diseño: Enumera al menos tres beneficios clave y da un ejemplo de cada uno.
2. Identificar patrones en código existente: Investiga un proyecto real y detecta si se usa algún patrón.
3. Clasificación de patrones: Para cada una de las tres familias, describe cuándo se deben usar.

**Caso de Uso: Plataforma de Reservas de Hoteles**

**Historia de Usuario:**

Como usuario de la plataforma "EasyBook", quiero realizar la reserva de una habitación de hotel sin preocuparme por la lógica interna del sistema, de modo que el proceso sea rápido y eficiente.

**Descripción:** La empresa "EasyBook" ha desarrollado un sistema de reservas de hoteles que conecta con múltiples proveedores. Para estructurar bien el código, el equipo de desarrollo decide aplicar patrones de diseño.

* **Problema:** El código de integración con cada proveedor de hoteles está disperso en varias clases, dificultando su mantenimiento.
* **Solución:** Se decide aplicar el **patrón Facade** para simplificar la interacción con las APIs de los proveedores y un **Factory Method** para la creación flexible de objetos de reserva.

// Facade para simplificar la reserva

class BookingFacade {

private HotelAPI hotelAPI;

private PaymentService paymentService;

public BookingFacade() {

this.hotelAPI = new HotelAPI();

this.paymentService = new PaymentService();

}

public void reservarHabitacion(String usuario, String hotel, double monto) {

hotelAPI.buscarHotel(hotel);

paymentService.procesarPago(usuario, monto);

System.out.println("Reserva completada para: " + usuario);

}

}

// Clases de soporte

class HotelAPI {

public void buscarHotel(String hotel) {

System.out.println("Buscando disponibilidad en " + hotel);

}

}

class PaymentService {

public void procesarPago(String usuario, double monto) {

System.out.println("Procesando pago de $" + monto + " para " + usuario);

}

}

// Cliente

public class Cliente {

public static void main(String[] args) {

BookingFacade booking = new BookingFacade();

booking.reservarHabitacion("Juan Pérez", "Hotel Plaza", 150.00);

}

}

// Cliente

public class Cliente {

public static void main(String[] args) {

BookingFacade booking = new BookingFacade();

booking.reservarHabitacion("Juan Pérez", "Hotel Plaza", 150.00);

}

}