Trabajo Práctico – Patrones de Diseño Estructurales

Objetivo

Aplicar los **patrones estructurales** vistos en clase (**Adapter, Bridge, Composite, Decorator, Facade, Flyweight y Proxy**) en situaciones prácticas, implementando soluciones en **Java** que permitan entender cómo y cuándo usarlos.

Ejercicios

Ejercicio 1 – Adapter

Una empresa tiene un sistema que trabaja con una clase ImpresoraPDF que imprime documentos PDF.
Ahora necesitan también imprimir en impresoras de texto plano, pero sin modificar el sistema existente.

Consigna:

- Crear una clase ImpresoraTexto (incompatible con la actual).
- Usar el patrón **Adapter** para que el sistema pueda imprimir tanto en PDF como en texto plano a través de una misma interfaz Impresora.

Ejercicio 2 – Bridge

Un sistema de **notificaciones** debe enviar mensajes a los usuarios por distintos medios: **Email** y **SMS**.

La notificación puede ser de distintos tipos: **Alerta**, **Recordatorio**, **Promoción**.

Consigna:

- Separar las abstracciones (tipos de notificación) de las implementaciones (canales de envío).
- Usar el patrón **Bridge** para que las notificaciones puedan enviarse por cualquier canal.

Ejercicio 3 – Decorator

Una aplicación de streaming necesita calcular el **costo de suscripción** de un usuario:

- Plan básico: $1000
- Opcionalmente, puede agregar **HD** (+$500), **UltraHD** (+$1000), **Descargas Offline** (+$700).

Consigna:

- Usar el patrón **Decorator** para ir agregando dinámicamente características al plan básico.
- Probar distintas combinaciones (ejemplo: Básico + HD + Descargas).

Ejercicio 4 – Facade

Una tienda online tiene un proceso de **compra** que involucra varias clases:

- Carrito (gestiona productos)
- Pago (procesa el pago)
- Envio (coordina la entrega)

Consigna:

- Implementar un **Facade** (TiendaFacade) que simplifique todo el proceso en un solo método comprar().
- Probar la compra de un producto desde el main usando solamente el Facade.

Ejercicio 5 – Flyweight

Se está desarrollando un juego donde aparecen miles de **árboles** en el mapa. Cada árbol tiene propiedades **intrínsecas** (tipo de árbol, textura, color) y **extrínsecas** (posición X,Y en el mapa).

Consigna:

- Implementar el patrón **Flyweight** para optimizar el uso de memoria.
- Crear una FabricaDeArboles que reutilice los objetos compartidos.
- Simular la creación de 1 millón de árboles con pocos tipos de árbol.

Ejercicio 6 – Proxy

Un sistema de acceso a archivos restringidos tiene la clase ArchivoReal, que abre y lee un archivo.
Se quiere implementar control de acceso para que solo ciertos usuarios puedan abrirlos.

Consigna:

- Implementar un **Proxy** (ArchivoProxy) que controle el acceso según permisos de usuario.
- Si el usuario no tiene permisos, mostrar un mensaje de acceso denegado.

Ejercicio 7 – Composite

Un sistema gestiona **elementos de un menú de restaurante**:

- **Platos individuales** (Plato)
- **Menús completos** que agrupan varios platos (Menu)

Consigna:

- Crear una **interfaz común** ElementoMenu para platos y menús.
- Implementar el patrón **Composite** para que se puedan manejar **platos individuales y menús completos de la misma forma**.
- Probar en el main creando un menú que contenga varios platos y un submenú con otros platos, y mostrar todos los elementos de manera uniforme.

Entregable

- Código Java de cada ejercicio.
- En cada ejercicio: breve explicación en comentarios sobre **cómo aplicaron el patrón**.