Trabajo Práctico 3 Primera Parte: Patrones de Diseño

Programación Orientada a Objetos II

Objetivo: El objetivo de este trabajo es que apliquen los patrones de diseño **Adapter**, **Composite** y **Template Method** en situaciones de desarrollo de software. Deberán diseñar y programar soluciones que implementen estos patrones.

Instrucciones:

1. El trabajo se realizará en grupo.
2. Cada grupo debe desarrollar los tres ejercicios utilizando el lenguaje de programación **Java**.
3. Cada ejercicio debe estar acompañado de un **diagrama de clases** y una breve **explicación** del por qué eligieron esa solución.
4. Los programas deben ejecutarse correctamente y cumplir con las funcionalidades solicitadas.
5. Se debe realizar la documentación de lo codificado.
6. Se debe crear un repositorio, subir la resolución del tp al mismo y compartir el link en la tarea.

Ejercicio 1: Patrón Adapter

Enunciado:
Tienes una aplicación que permite a los usuarios gestionar sus listas de reproducción de música. Actualmente, la aplicación utiliza una interfaz `MusicPlayer` para reproducir canciones, pero ahora te piden integrar un servicio externo llamado `ThirdPartyAudioPlayer` que tiene una interfaz diferente para reproducir pistas de audio.

Tareas:
1. Implementa un adaptador que permita que `ThirdPartyAudioPlayer` funcione con la interfaz `MusicPlayer` existente.
2. Crea una clase `MusicApp` que utilice tu adaptador para reproducir canciones desde ambos reproductores.

Requisitos:
- La interfaz `MusicPlayer` debe tener un método `playSong(String fileName)`.
- La clase `ThirdPartyAudioPlayer` tiene un método `startAudio(String file)`.
- Usa el patrón Adapter para integrar ambos sistemas sin modificar la clase `ThirdPartyAudioPlayer`.

Puntos clave a evaluar:

- Aplicación correcta del patrón Adapter.
- Correcta implementación del adaptador para traducir las interfaces.
- Diagrama de clases bien estructurado.

Ejercicio 2: Patrón Composite

Enunciado:
Tienes que diseñar un sistema de gestión de archivos y directorios, donde tanto los archivos como los directorios deben ser tratados de manera uniforme. Los directorios pueden contener tanto archivos como otros directorios.

Tareas:

1. Crea una clase base abstracta `FileSystemComponent` con un método `showDetails()`.
2. Crea las clases `File` y `Directory` que hereden de `FileSystemComponent`. Los objetos de la clase `Directory` pueden contener tanto archivos como otros directorios.
3. Implementa una función en la clase `Directory` para agregar y eliminar componentes (archivos o directorios).
4. Implementa un cliente que permita crear una estructura jerárquica de directorios y archivos, y que muestre los detalles de esta estructura.

Requisitos:

- La clase `File` debe tener un nombre y su tamaño en bytes.
- La clase `Directory` debe poder contener una lista de objetos `FileSystemComponent`.
- El método `showDetails()` debe imprimir el nombre del archivo o directorio, y en el caso de los directorios, debe listar su contenido de manera recursiva.

Puntos clave a evaluar:

- Correcta aplicación del patrón Composite.
- Implementación de la recursividad para mostrar los detalles de los directorios y sus contenidos.
- Diagrama de clases que refleje la jerarquía de objetos compuestos.

Ejercicio 3: Patrón Template Method

Enunciado:
Debes desarrollar un sistema que gestione el proceso de fabricación de diferentes tipos de pasteles. El proceso general para hacer un pastel incluye: preparar los ingredientes, hornear, decorar y empaquetar. Sin embargo, los detalles de cada paso varían según el tipo de pastel (por ejemplo, pastel de chocolate, pastel de vainilla).

Tareas:

1. Crea una clase abstracta `Cake` que defina el método `makeCake()`, que incluye los pasos generales para hacer un pastel.

2. Define métodos abstractos para los pasos que varían entre los diferentes tipos de pasteles (por ejemplo, prepareIngredients(), decorateCake()).
3. Crea subclases como ChocolateCake y VanillaCake que implementen los detalles específicos de estos pasos.
4. Implementa un cliente que cree diferentes tipos de pasteles y los fabrique usando el método makeCake().

Requisitos:

- La clase Cake debe tener el método makeCake(), que no debe ser modificado por las subclases.
- Los pasos como prepareIngredients() y decorateCake() deben ser implementados en las subclases.
- El cliente debe ser capaz de crear tanto un pastel de chocolate como un pastel de vainilla y procesarlos utilizando el mismo flujo de fabricación.

Puntos clave a evaluar:

- Correcta aplicación del patrón Template Method.
- Implementación clara de las variaciones en los pasos del algoritmo entre los diferentes tipos de pasteles.
- Diagrama de clases que ilustre la estructura jerárquica y los métodos.