Practica 1

Luis Geovanni Méndez Ávila

11 de Octubre 2020

1.-

Patrón strategy

Ventajas:

- Abstracción de algoritomos específicos en una interfaz de la que se van a implementar distintos tipos de estrategias para ejecutar esos algoritmos.
- Cambio de estrategias en tiempo de ejecución.
- Facilidad para agregar y quitar estrategias sin que el código principal sea afectado.

Desventajas:

- Aunmenta la cantidad de objetos creados.

Patrón observer

Ventajas:

- Establece una forma de comunicación de uno a muchos.

Desventajas:

- No se conoce la forma en que pueda afectar la actualización.

2.-

Compilar y ejecutar

Descomprimir el zip y con una consola dirigirte hasta la carpeta src. Despues compila con el comando javac *.java , una vez complilado ejecutas con java Main.

Nota

Me faltó aplicar el patrón observer, comúnmente en los ejemplos que veíamos el observador podía guardar la referencia al sujeto al que sigue ya que es de un tipo fijo. Pero en éste problema el tipo del sujeto podía ser disitnto, para ser exacto a cualquier objeto que implementase la interfaz Servicio (Netflis, Hammazon, etc.) lo que hacía que tuvieras dos opciones, o guardabas la referencia al sujeto como si fuera de tipo Servicio como la interfaz o guardar una referencia a cada tipo de servicio, lo cual no se hacía muy lógico.
Al optar por guardar la referencia al sujeto en el observador como de tipo Servicio hace que no pueda acceder a métodos propios de cada servicio que no estén declarados en la interfaz como el método getRecomendacion , el cuál era importante para hacer la actualización del observador.
Intenté solucionar el problema aplicando el patrón strategy para conseguir la recomendación de los servicios pero no quedó de esa manera. Mi única opción era hacer un monton de referencias a los distinros servicios en el observador o modificar la interfaz del sujeto , pero ninguna se me hacía "legal", así que terminé por dejar sin aplicar el patrón observer.

Respecto a la manera de identificar si un usuario ya fue registrado previamente para no cobrar desde 0 desde el primer día, mi idea fue poner una variable de clase de tipo booleana que cambiara a true cuando el usuario se registra a un servicio, sin embargo el problema fue que se quedaba así para cualquier otro serivicio, lo que iba a causar problemas con los otros servicios a los que se suscribiera, no se me ocurrió otra forma de implementar esa parte, pero de todos modos la quité para no cuasar problemas a los demás servicios.

No pude deducir como manejar el curso de los días.

Respecto al diagrama , quise hacerlo de manera un poco más reducida para

evitar tanto enredo con las flechas y los diagramas de clases, así que el diagrama de clase que dice xServicio puede ser cualquiera de los servicios que implementaron la interfaz Servicio, sus métodos estrategiaA(), estrategiaB() , estrategiaN() , las respectivas estrategias con las que se va a asignar la forma de pago y el diagrama de clase de Estrategia representa cuaquiera de esas estrategias que implementaron la interfaz SistemaCobro.