UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Facultad de Ciencias

Equipo NullPointerException:
Diego Angel Rosas Franco - 318165330
Adrián Aguilera Moreno - 421005200
Marco Antonio Rivera Silva - 318183583

Modelado y programación

Práctica 1

Principios de diseño esenciales del patrón:
Strategy:

1. El comportamiento de una clase cambia en tiempo de ejecución.
2. Se crea una familia de algoritmos que representan varias estrategias para resolver un conjunto de problemas que dependen de algunos comportamientos en común, cada estrategia corresponde a un elemento en este conjunto de problemas y esta relación debe ser biyectiva.
3. Cambia comportamientos de manera dinámica (en construcción) sin condicionar la clase de dependencias de las estrategias.

Observer:

1. Los Sujetos guardan una lista de Observadores a los cuales tienen que comunicarles un mensaje.
2. Los Sujetos tienen una forma de agregar y remover Observadores en cualquier punto de la ejecución.
3. Los Sujetos tienen una forma de notificar por igual a todos sus Observadores el mismo mensaje.
4. Los Observadores tienen una forma de agregarse o eliminarse por su cuenta.
5. La comunicación debe hacerse de uno a muchos (Sujeto-Observador), pero esto no es una limitante para tener más de un Sujeto por Observador.

Menciona una desventaja de cada patrón:
Desventajas en el patrón Observer:

1. El almacenamiento podría llegar ha ser la gran desventaja en el patrón Observer, pues cuando se anexan nuevos observadores estos se tienen que declarar de explícita, esto no supone un verdadero problema cuando los observadores son pocos, pero en cuanto el número de observadores crece, los recursos computacionales se ven reducidos (no necesariamente de manera drástica).
2. Al tener una comunicación de uno a muchos debemos elegir correctamente qué problemas se pueden abordar con este patrón, pues una mala decisión nos imposibilita el poder realizar una comunicación particular (en caso de ser necesaria).

Desventajas en el patrón Strategy:

1. Un problema de automatización llegaría a ser una desventaja en este patrón, pues no podemos tratar de manera general a los sujetos dependientes de Strategy.
2. Cuando se trabaje con los elementos que implementan el patrón de diseño Strategy se debe trabajar con cada uno por separado (esto es, se trabaja particularmente), así el desarrollador se debe tomar el tiempo de realizar modificaciones pertinentes clase a clase si lo que necesita no es suficientemente general (si hay una propiedad a añadir que solo afecte a alguna de las clases que implementan este patrón).
3. En caso de tener una estrategia definida que no necesite todos los métodos obligatorios (métodos de interfaz o clase abstracta/hereda) ésta debe implementar todos los métodos, incluso si dejan de tener funcionalidad (muchas veces podemos darle la vuelta a este tipo de cosas, pero esto podría implicar el caer en 2).

Instrucciones de instalación, compilación y ejecución. Se dará por hecho que el usuario sabe moverse en terminal, sabe usar git.

Requerimientos previos:

- Se debe contar con Java en su computadora. De preferencia la versión más reciente.

Ejecución del proyecto:

- Si está leyendo esto significa que desempaquetó con éxito el proyecto.
- Abra su terminal y diríjase a la ruta donde desempaquetó el proyecto.
- Una vez estando en la ruta `Practica01_NullPointerException`, diríjase a `Practica01_NullPointerException/src/fciencias/modelado/`
- Ejecute: "javac Practica01.java", esto generará los .class del proyecto.
- Ejecute: "java Practica01", esto ejecutará el proyecto mostrándole en consola un log de prueba del proyecto especificado en el pdf de la práctica.

Sobre el proyecto:

Strategy:
El patrón Strategy se implementó de la siguiente manera en el proyecto:
Cada Servicio tendría un Cobrador propio, por ejemplo: CobradorMemeflix, CobradorMomazon, etc. Y estos Cobradores podrán cambiar su estrategia de cobro en tiempo de ejecución según la suscripción que se vaya a cobrar, y justo estas estrategia son los tipos de cobro para cada tipo de suscripción que tiene el servicio.
Por lo que el patrón esta presente 5 veces en el proyecto, uno por cada servicio.
Observer:
El patrón Observer se implementó con las clases Servicio y Suscripcion, donde Servicio es el Sujeto y Suscripción los Observadores.
Cada Servicio tiene una lista de Suscripciones, es decir, una lista de Observadores.
Los mensajes que el Servicio estará comunicando a sus Suscripciones serán dos: uno podríamos describirlo como "paga lo que me debes" y el otro es una recomendación. Estos mensajes se estarán dando mensualmente acorde al contexto del proyecto.
Clase Suscripcion y CuentaBancaria:
Consideramos que sería útil manejar a los usuarios de los servicios mediante suscripciones, de esta forma un usuario puede estar suscrito a varios servicios, y además puede suscribirse dos veces al mismo, esto mediante un correo, porque en la vida real sucede que una persona puede tener más de una cuenta en un servicio con distintos correos.
En cambio, la parte de CuentaBancaria, solo será una para cada usuario, y esta clase nos permitirá pagar cosas y delegarle a esta clase esta responsabilidad y no a algún método de Usuario.