Taller #5

Patrones

Patrón: Estrategias

1. Proyecto

El proyecto “Patrones” es un repositorio de GitHub que contiene la implementación de los 19 patrones de diseño GoF en Java, con ejemplos y explicaciones. El objetivo del proyecto es mostrar cómo se pueden aplicar los patrones de diseño para resolver problemas comunes de programación orientada a objetos. El proyecto se puede consultar desde [https://github.com/sidlors/patrones/tree/a38c7bc5b6b2e05e03f546970dbeb75a517e06f4/16_Estrategia].

2. Explicación conceptual del patrón y del contexto

El patrón de diseño Estrategia permite definir una familia de algoritmos, encapsular cada uno de ellos y hacerlos intercambiables. El patrón Estrategia permite que el algoritmo varíe independientemente de los clientes que lo usan. El patrón Estrategia se compone de los siguientes elementos:

- **Contexto:** es la clase que usa una estrategia para realizar una operación. Tiene una referencia a una estrategia abstracta y delega la ejecución del algoritmo en ella.
- **Estrategia:** es la clase abstracta que define la interfaz común para todos los algoritmos. Declara un método abstracto que debe ser implementado por las estrategias concretas.
- **Estrategia Concreta:** es la clase que implementa una estrategia específica. Define el algoritmo que se ejecuta cuando el contexto invoca el método de la estrategia. La motivación para usar el patrón Estrategia es poder cambiar el comportamiento de un objeto en tiempo de ejecución, sin tener que modificar el código del objeto o usar sentencias condicionales. El patrón Estrategia permite encapsular los algoritmos en clases separadas y hacer que el contexto pueda elegir la estrategia adecuada según las circunstancias.

3. Aplicación del patrón en el proyecto

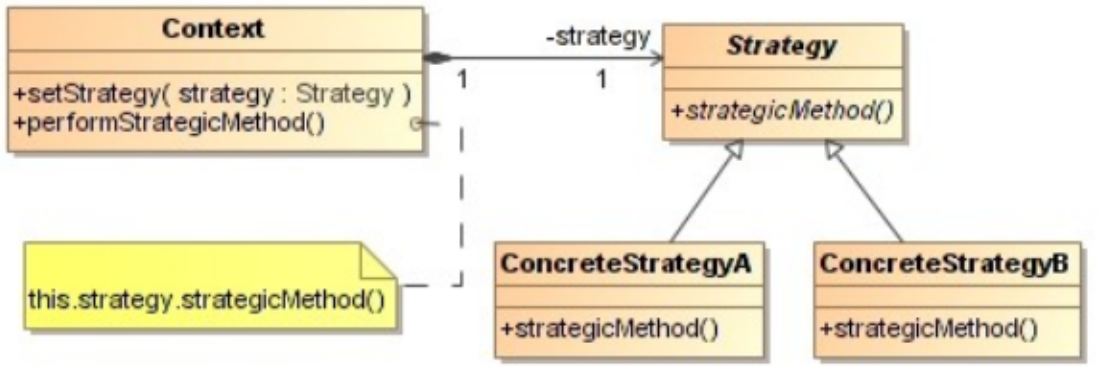
El patrón Estrategia se utiliza en el proyecto “Patrones” para implementar el ejemplo del juego de piedra, papel o tijera. El contexto es la clase Juego, que representa el juego entre dos jugadores. La estrategia es la clase abstracta Jugada, que define la interfaz común para todas las jugadas posibles. Las estrategias concretas son las clases Piedra, Papel y Tijera, que implementan las jugadas específicas. El contexto tiene una referencia a una estrategia abstracta y delega la ejecución de la jugada en

ella. El contexto puede cambiar la estrategia en tiempo de ejecución, según la elección del jugador. El patrón Estrategia permite que el juego pueda tener diferentes comportamientos según la jugada elegida, sin tener que modificar el código del juego o usar sentencias condicionales.

4. Aparición del patrón

La información y estructura del fragmento del proyecto donde aparece el patrón Estrategia son las siguientes:

- El archivo que contiene el código fuente del patrón Estrategia se llama Estrategia.java y se encuentra en la carpeta src/main/java/com/patrones/estrategia del repositorio.
- El archivo contiene las declaraciones de las clases Juego, Jugada, Piedra, Papel y Tijera, y el método main para probar el ejemplo.
- El archivo muestra los elementos que definen el patrón Estrategia, como se puede ver en el siguiente diagrama de clases:



5. Ventajas

Las ventajas de usar el patrón Estrategia en ese punto del proyecto son las siguientes:

- El patrón Estrategia permite separar las responsabilidades entre el juego y las jugadas, haciendo que el código sea más modular y fácil de mantener.
- El patrón Estrategia permite añadir nuevas jugadas sin tener que modificar el código del juego o usar sentencias condicionales, haciendo que el código sea más flexible y extensible.
- El patrón Estrategia permite cambiar el comportamiento del juego en tiempo de ejecución, según la elección del jugador, haciendo que el juego sea más dinámico e interactivo.

6. Desventajas

Las desventajas o limitaciones de usar el patrón Estrategia en ese punto del proyecto son las siguientes:

- El patrón Estrategia puede aumentar la complejidad del diseño, al introducir más clases y más niveles de abstracción, lo que puede dificultar la comprensión y el seguimiento del código.
- El patrón Estrategia puede generar una sobrecarga de comunicación entre el contexto y la estrategia, al tener que pasar los datos necesarios para la ejecución del algoritmo, lo que puede afectar al rendimiento y la eficiencia del código.
- El patrón Estrategia puede requerir la creación de múltiples objetos de estrategia, lo que puede consumir más recursos de memoria y generar más basura, lo que puede afectar al rendimiento y la eficiencia del código.

7. Reconstrucción

Para reconstruir el diseño a partir de los artefactos concretos (código fuente), se puede seguir el siguiente pseudocódigo:

- Crear una clase abstracta Jugada que define la interfaz común para todas las jugadas posibles. Declara un método abstracto jugar que reciba una jugada como parámetro y devuelva un resultado como cadena.
- Crear una clase concreta Piedra que herede de Jugada e implemente el método jugar. Define el algoritmo para jugar con piedra, comparando la jugada recibida y devolviendo el resultado según las reglas del juego.
- Crear una clase concreta Papel que herede de Jugada e implemente el método jugar. Define el algoritmo para jugar con papel, comparando la jugada recibida y devolviendo el resultado según las reglas del juego.
- Crear una clase concreta Tijera que herede de Jugada e implemente el método jugar. Define el algoritmo para jugar con tijera, comparando la jugada recibida y devolviendo el resultado según las reglas del juego.
- Crear una clase Juego que represente el juego entre dos jugadores. Tiene una referencia a una jugada abstracta y un método elegirJugada que recibe una cadena como parámetro y asigna la jugada correspondiente a la referencia. Tiene un método jugar que recibe una jugada como parámetro y delega la ejecución de la jugada en la referencia. Tiene un método main que crea un objeto de juego y prueba el ejemplo con diferentes jugadas.