Diseño de Software

Patrones de Diseño

Patron I

STRATEGY

Autor
Ekaitz Arriola Garcia

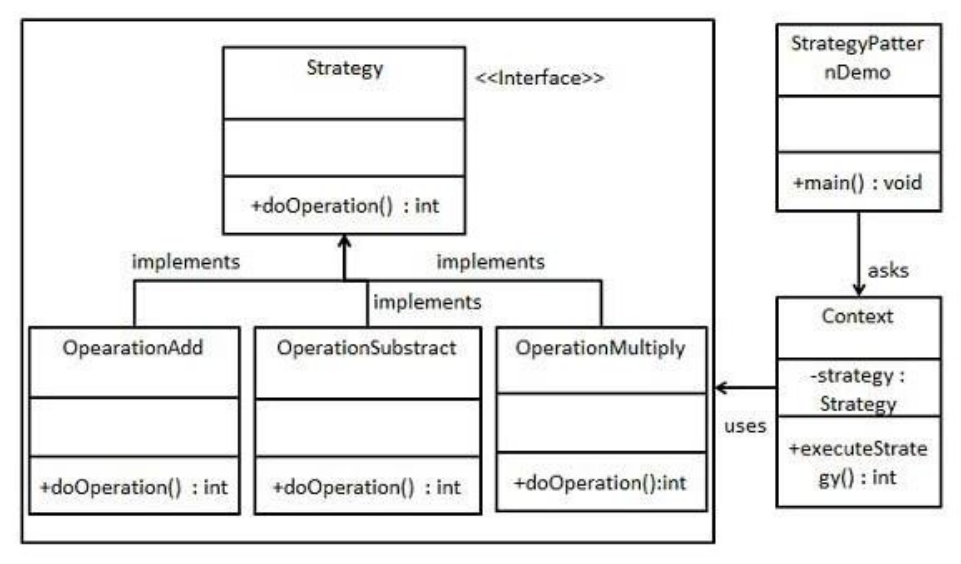
Profesor
Miguel Angel Mesas Uzal

March 7, 2021

Índice

Descripción general de la solución - Patron Strategy

(a) Contexto, restricciones, alcance

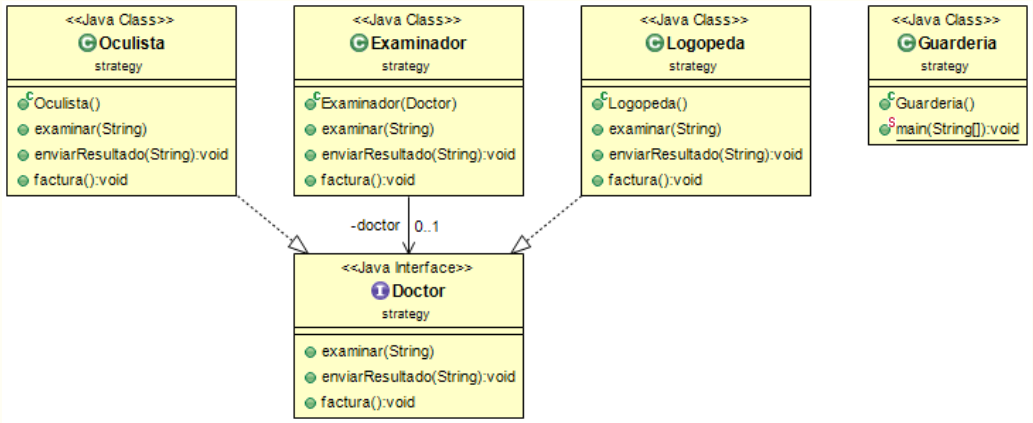


Este patrón permite el mantenimiento de un conjunto de algoritmos, entre lo cuales se puede elegir usar uno de ellos e intercambiarlos dinámicamente según el contexto y necesidades. Su homónimo mal, de forma simplificada, sería un switch infinitamente elongado. Esto es, todos los casos de estrategias a utilizar se alojarían en un solo código, provocando que estos sean demasiados grandes y complicados me mantener y escalar.

Usar este patrón limita el uso de la herencia, ya que se accede a estas soluciones (o estrategias) mediante otra clase no relacionada por herencia sino a la que se le delega una estrategia para hacer de intermediario.
A cambio, este patrón nos permite añadir nuevas estrategias sin alterar nada del código anterior, dándonos la libertad de cambiar las estrategias como y cuando queramos, claro esta, siempre respetando el formato en uso de dichas estrategias.

Componentes del patrón utilizado

(b) Descripción UML - Uso de composición/agregación y Interface/clase abstracta



Este ejemplo minimalista del patrón strategy esta compuesto por una interfaz que implementan otras dos clases, las cuales se delegan mediante composición a otra, la cual es usada por Guardería (la clase Test).

La interfaz Doctor es la que dictamina la estructura de las estrategias. De ella es de la que la heredan aquellas clases que modelan las estrategias, las cuales obviamente lo implementan. Estas últimas —Oculista y Logopeda—, a su vez, se delegan mediante composición a la clase Examinador, y no mediante agregación ya que para la clase Examinador es imprescindible tener una estrategia definida.
Por último,la clase Guardería sera en este caso la clase Test, la cual ejecuta y prueba el programa.

(c) Uso de Interface/clase abstracta

Se usa una interfaz ya que se quiere determinar que métodos deben implementar las estrategias. En el caso de necesitarse, se podría usar una clase abstracta intermedia tal y como ocurre en el ejemplo del final, el cual esta más elaborado.

(d) Motivo del diseño utilizado

El principal motivo para usar este diseño es evitar usar un switch o cadena de else-ifs, al haber distintas formas en las que se implementa Doctor.

Resultado de la ejecución mediante una clase Test

(e) Descripción del test realizado

En esta solución, Guardería es la clase Test, esto es, tendrá el main. Esta usará un Examinador con una estrategia definida (como no puede ser de otra forma) para ejecutar los métodos exigidos por Doctor.

(f) Resultado del Test

Resultado del test

```
Enviando resultado:Paco examinado por el oculista
Enviando factura
Enviando resultado:Yeray examinado por el logopeda
Enviando factura
```

El resultado del test, al ser una solución muy minimalista, los métodos solo operan con Strings.

Conclusiones

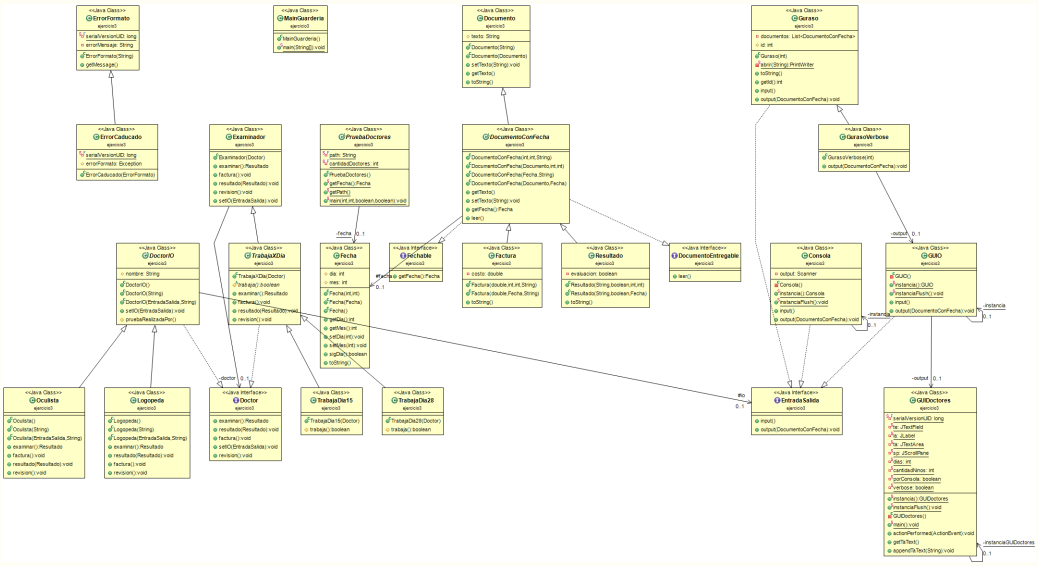
(g) Adaptación al cambio

Cumpliendo con su propósito, al aplicar este patrón se pueden crear, editar y obliterar las estrategias como y cuando se quiera siempre que se respete la estructura intrínseca

(h) Posibles mejoras - Otras soluciones

Esta solución es bastante simple, lo justo necesario para que se manifieste el patrón. Ahora bien, se podrían añadir bastantes mejoras y extras (cambiar los Strings por clases, añadir excepciones...), incluso añadiendo otros patrones, como puede ser el patrón decorador, el patrón fachada...

Problema continua en la siguiente página...



Pulsa en la imagen o busca abajo en la bibliografia para poder acceder a la imagen. Al ser png, se le puede hacer zoom y se ve bastante bien.
Este sería un ejemplo del uso del patrón strategy más elaborado que el anterior, aunque no es que sea muy complejo lograr eso. El codigo esta subido en github. Se puede encontrar sacando el link que hace media frase, o en la bibliografía.

Otra posible solución, sería alterar el contexto. Esto es, que haya distintos tipos de contextos, o que al contexto se le delegen más cosas depende de las cuales actue de una forma u otra.

Bibliografía

[1] Código y UML
[2] Tutorials Point - Strategy Pattern
[3] Y claro está, las clases