Politécnico Colombiano Jaime Isaza Cadavid

Facultad de Ingeniería

Strategy pattern

Diseño de software

Autor: Claudia
Apellidos: Gil Sánchez

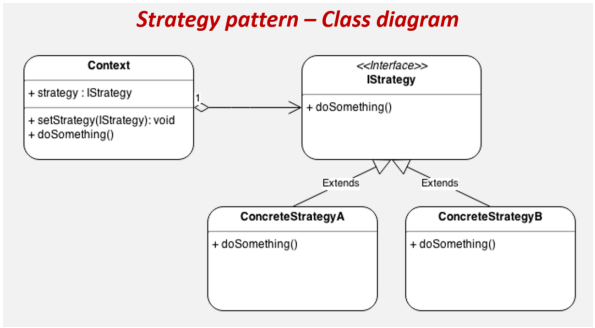
3 de febrero de 2022

Introducción

El patrón Estrategia es un patrón de diseño para el desarrollo de software. Se clasifica como patrón de comportamiento porque determina cómo se debe realizar el intercambio de mensajes entre diferentes objetos para resolver una tarea.

Strategy

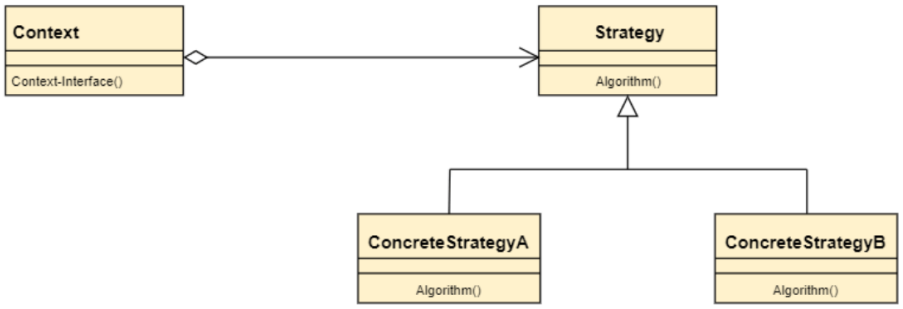
Strategy se basa en el polimorfismo para implementar una serie de comportamientos que podrán ser intercambiados durante la ejecución del programa, logrando con esto que un objeto se pueda comportar de forma distinta según la estrategia establecida.



1. Los componentes del patrón son los siguientes:

    - Context: Componente que encapsula la estrategia a utilizar, tiene como característica que se puede establecer la estrategia a utilizar en tiempo de ejecución

- IStrategy: Interface en común que todas las estrategias deberán implementar. En esta interface se definen las operaciones que las estrategias deberán implementar.
- ConcreteStrategy: Representa las estrategias concretas, las cuales heredan de IStrategy.



Consecuencias:

Ventajas: Permite cambiar el comportamiento de la clase de manera dinámica sin poner condiciones dentro de la clase contexto. Permite encapsular comportamiento para reutilizarlo y no repetir código. Permite modificar comportamiento sin tener que modificar las clases de contexto.

Desventajas: La aplicación debe estar al tanto de todas las posibles estrategias para seleccionar la adecuada para cada situación. Todas las estrategias deben implementar la misma interfaz, esto hace que algunas estrategias tengan que implementar métodos que no necesitan para su comportamiento.

La interfaz Strategy encapsula las variantes de cálculo y puede ser implementada simultáneamente por todos los algoritmos. Para la interacción con Context, la interfaz genérica proporciona solo un método para activar los algoritmos de ConcreteStrategy. Utilizando una referencia interna, el contexto puede acceder a las variables de computación externalizadas (ConcreteStrategyA, ConcreteStrategyB, etc.) si lo necesita. No interactúa directamente con los algoritmos, sino con una interfaz. Estas clases separadas contienen los algoritmos llamados ConcreteStrategies.

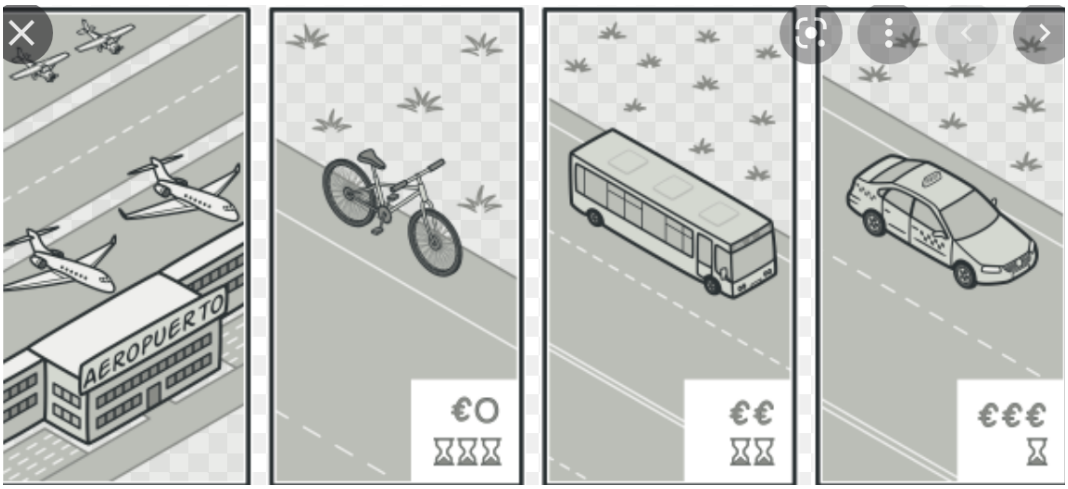
Similitud con otros patrones

Es posible que este patrón nos recuerde a otros que ya hemos comentado en anteriores posts, como el Patrón Command, el Patrón Adapter o el Patrón Template Method.

- Command Se parecen porque ambos parametrizan un objeto con una acción definida, sin embargo tienen fines diferentes:
  Con Command conviertes una operación en un objeto, esto te permite aplazar la ejecución del mismo, encolarla, guardar un histórico de ejecución o incluso enviarlos a servicios remotos como en sistemas RPC.
  Strategy en cambio te permite ejecutar distintas formas de una misma acción sobre un mismo contexto.

- Adapter
  Tiene un parecido estructural, ya que ambos se basan en distintas implementaciones de una misma interfaz. Sin embargo, solucionan problemas diferentes.

- Template Method
  La diferencia con Template Method es que este e basa en herencia, a diferencia de Strategy que se basa en composición. Por otra parte Template Method funciona a nivel clase, por lo que es estático, mientras que Strategy lo hace a nivel objeto y nos permite cambiar su comportamiento en tiempo de ejecución.



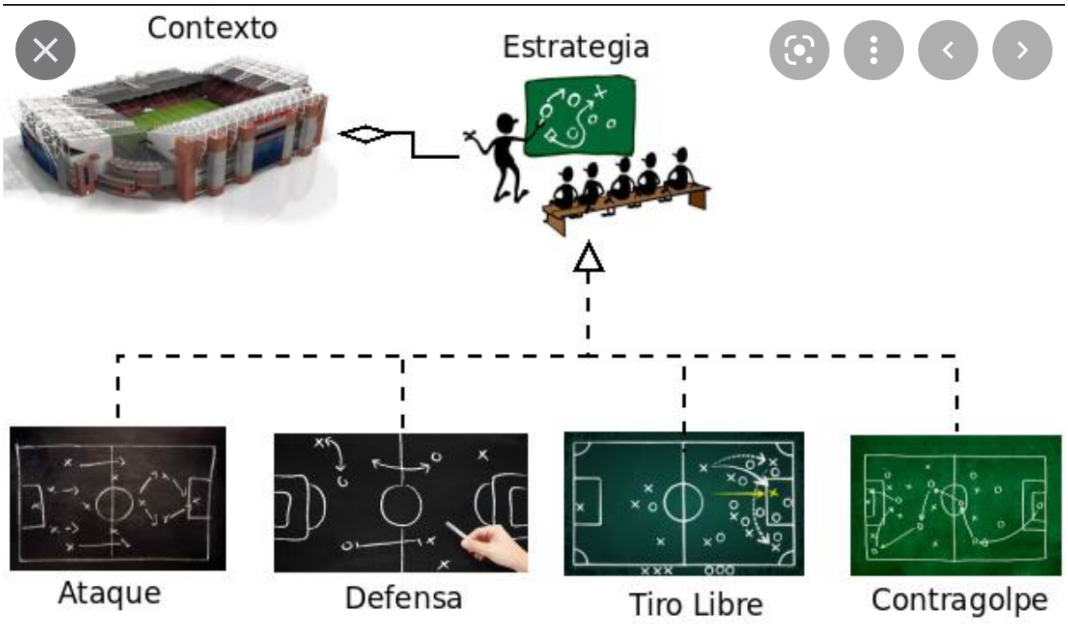
Cómo implementarlo

Se pueden encontrar aplicaciones más específicas de los strategy patterns en la biblioteca estándar de Java (Java API) y en los conjuntos de herramientas de la interfaz gráfica de Java (por ejemplo, AWT, Swing y SWT), que utilizan un gestor de diseño en el desarrollo y la generación de interfaces gráficas de usuario. Esto permite, por ejemplo, convertir de forma versátil los vídeos a un formato de archivo que ahorre espacio o restaurar archivos comprimidos (por ejemplo, archivos ZIP o RAR) a su estado original mediante estrategias especiales de descompresión.

Por ejemplo, los programas que ofrecen diferentes formatos de almacenamiento de archivos o varias funciones de clasificación y búsqueda utilizan el strategy pattern.

El patrón de diseño strategy también se utiliza en el desarrollo e implementación de software de juegos, que, por ejemplo, tienen que reaccionar con flexibilidad a las situaciones cambiantes del juego durante el tiempo de ejecución. También los programas que traducen los datos a diferentes formatos

gráficos (por ejemplo, como gráficos de líneas, circulares o de barras) utilizan strategy patterns. El patrón strategy es ideal para software que ha de resolver tareas y problemas variables, opciones de comportamiento y cambios. Como patrón de diseño básico en el desarrollo de software, el strategy patternno está limitado a un solo ámbito de aplicación. El strategy pattern se utiliza también en sistemas de base de datos, controladores de dispositivos y programas de servidores.



Resumen del patron

Los strategy patterns permiten el desarrollo eficiente y económico de software en la programación orientada a objetos con soluciones de problemas a medida. El sistema, que está orientado a la variabilidad y al dinamismo, puede así, en general, controlarse y vigilarse mejor. Los componentes reutilizables e intercambiables ahorran costes de desarrollo, especialmente en proyectos complejos con una perspectiva a largo plazo.

- Está orientado al comportamiento (los comportamientos y los cambios son más fáciles de programar e implementar, los cambios también son posibles durante el tiempo de ejecución de un programa).
- Está orientado a la eficiencia (la externalización simplifica y optimiza el código y su mantenimiento).
- Está orientado al futuro (los cambios y optimizaciones también son fáciles de realizar a medio y largo plazo).
- Tiene como objetivo la capacidad de expansión (favorecida por el diseño modular y la independencia de los objetos y las clases).
- Tiene como objetivo la reutilización (por ejemplo, el uso recurrente de estrategias).
- Tiene como objetivo optimizar la usabilidad, controlabilidad y configurabilidad del software.

- Requiere consideraciones conceptuales minuciosas (¿qué se puede externalizar a las clases Strategy, dónde y cómo?).
- Identificación: El patrón Builder se puede reconocer por una clase, que tiene un único método de creación y varios métodos para configurar el objeto resultante. A menudo, los métodos del Builder soportan el encadenamiento (por ejemplo, algúnBuilder-¿establecerValorA(1)-¿establecerValorB(2)-¿crear()).

Ejemplo del patrón de diseño

```
public class Context {
    // Valor por defecto (comportamiento por defecto): ConcreteStrategyA
    private Strategy strategy = new ConcreteStrategyA();
    public void execute() {
        //delega el comportamiento a un objeto de estrategia
        strategy.executeAlgorithm();
    }
    public void setStrategy(Strategy strategy) {
        strategy = strategy;
    }
    public Strategy getStrategy() {
        return strategy;
    }
}
```

```
interface Strategy {
    public void executeAlgorithm();
}
class ConcreteStrategyA implements Strategy {
    public void executeAlgorithm() {
        System.out.println("Concrete Strategy A");
    }
}
class ConcreteStrategyB implements Strategy {
    public void executeAlgorithm() {
        System.out.println("Concrete Strategy B");
    }
}
```

```
public class Client {
    public static void main(String[] args) {
        //Comportamiento por defecto
        Context context = new Context();
        context.execute();
        //Cambiar el comportamiento
        context.setStrategy(new ConcreteStrategyB());
        context.execute();
    }
}
```

Referencias

Strategy pattern: un patrón de diseño de software para estrategias variables de comportamiento. (n.d.-a). IONOS Digitalguide. Retrieved January 27, 2022, from https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/strategy-pattern/.

Ortega Mateo, E. (2020, January 2). Patrones de comportamiento - Strategy. Somos PNT - Desarrollamos Software https://somospnt.com/blog/136-patrones-de-comportamiento-strategy.

Strategy. (n.d.). Reactiveprogramming.Io. Retrieved January 27, 2022, from https://reactiveprogramming.io/blog/es/patrones-de-diseno/strategy.

(N.d.). Medium.Com. Retrieved January 27, 2022, from https://medium.com/all-you-need-is-clean-code/patr\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{o\global\mathchardef\accent@spacefactor\spacefactor}\let\begingroup\let\typeout\protect\begingroup\def\MessageBreak{Ω(Font)}\let\protect\immediate\write\m@ne{LaTeXFontInfo:oninputline160.}\endgroup\endgroup\relax\let\ignorespaces\relax\accent19o\egroup\spacefactor\accent@spacefactor\futurelet\@let@token\protect\penalty\@M\hskip\z@skipn-estrategia-strategy-pattern-654c6e9d2abe.