TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

Tarea
Patrón strategy

Docente
José de Jesús Parra Galaviz

SEMESTRE agosto – diciembre 2021

INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES

Patrones de diseño

Sergio Alberto Garza Aguilar 15211700

24 de septiembre del 2021

Descripción

El patrón de diseño strategy permite establecer en tiempo de ejecución el rol de comportamiento de una clase. 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.

Funcionalidades

- Ayuda a eliminar sentencias condicionales.
- Ayuda a factorizar la funcionalidad común de los algoritmos.
- Hay mucha diversidad de estrategias para un comportamiento, por lo que el cliente debe comprender bien cómo funcionan cada una de ellas.
- Es una alternativa a las subclases. Encapsulando los algoritmos en subclases separadas hace que dichos algoritmos sean totalmente independientes del contexto, facilitando la comprensión, entendimiento y extensión.
- Se incrementa sustancialmente el número de objetos.
- Estimación "por lo alto" de las comunicaciones entre Estrategia y Contexto.

- **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.

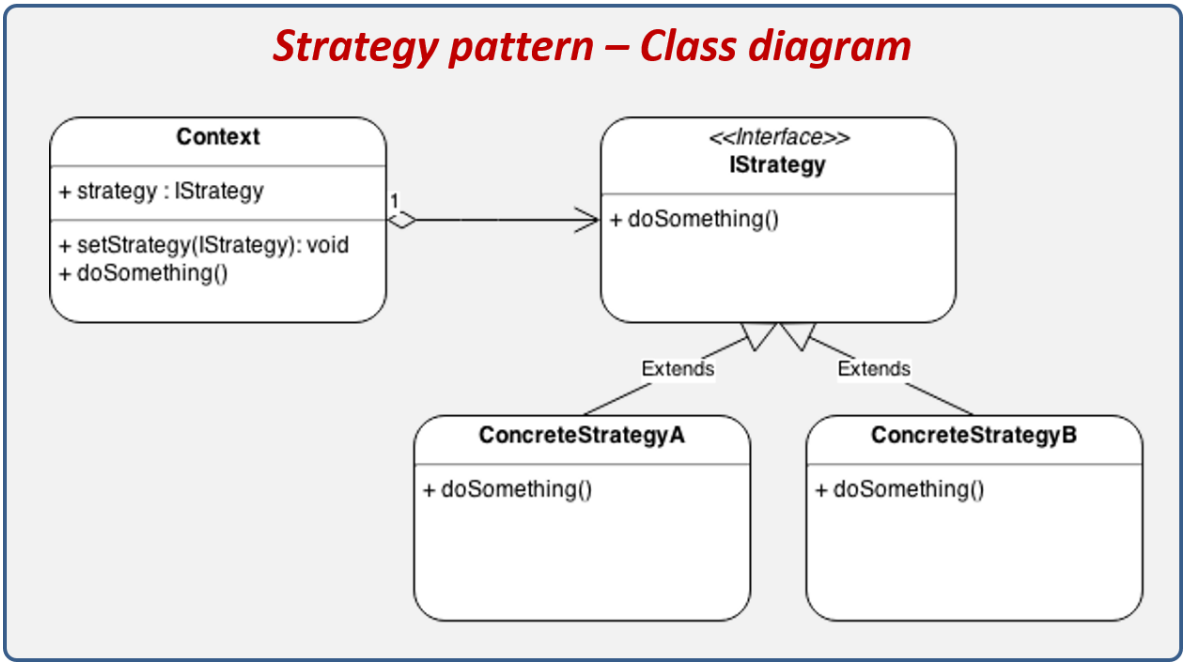
Componentes

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

- **IStrategy**
  Interfaz en común que todas las estrategias deberán implementar. En esta interfaz se definen las operaciones que las estrategias deberán implementar.

- **ConcreteStrategy**
  Representa las estrategias concretas, las cuales heredan de IStrategy.

Diagrama



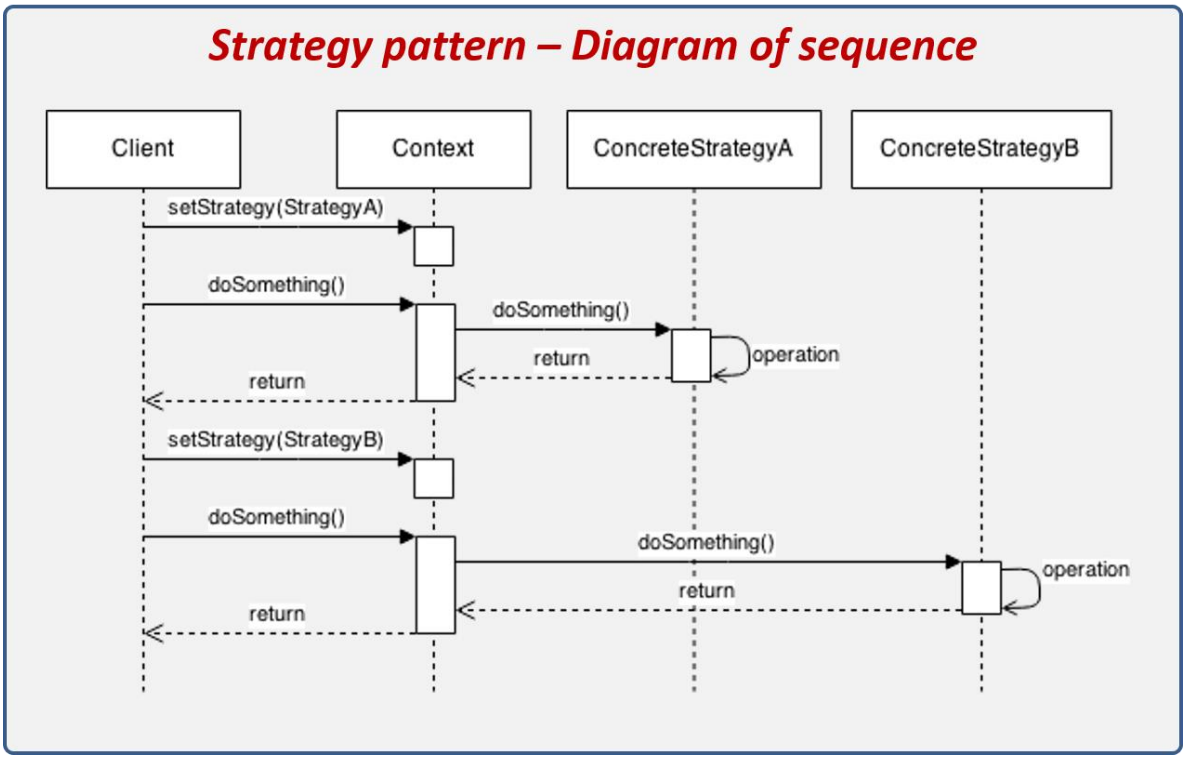
Implementación

1. Definición de la comunicación entre Contexto y Estrategia.

2. Configurar Contexto con una Estrategia.

3. Definir el comportamiento por defecto del Contexto en el caso de que no exista una Estrategia.

Diagrama de secuencia



1. El cliente crea un nuevo contexto y establece la estrategia A.
2. El cliente ejecuta la operación doSomething.
3. Context a su vez delega esta responsabilidad a ConcreteStrategyA.
4. ConcreteStrategyA realiza la operación y regresa el resultado.
5. Context toma el resultado y lo regresa al cliente.
6. El cliente le cambia la estrategia al Context en tiempo de ejecución.
7. El cliente ejecuta nuevamente la operación doSomething.
8. Context a su vez delega esta responsabilidad a ConcreteStrategyB.
9. ConcreteStategyB realiza la operación y regresa el resultado.
10. Context toma el resultado y lo regresa al cliente.

Ejemplo

Declaración de contexto OperacionAritmetica y de las estrategias a utilizar

- Suma
- Resta
- Multiplicación

Main.java:

```
01.  package Strategy;
02.
03.  public class Main
04.  {
05.      public static void main(String[] args)
06.      {
07.          OperacionAritmetica context;
08.
09.          context = new OperacionAritmetica( new Sumar() );
10.          int suma = context.procesar(3,4);
11.
12.          context = new OperacionAritmetica( new Restar() );
13.          int resta = context.procesar(3,4);
14.
15.          context = new OperacionAritmetica( new Multiplicar() );
16.          int multip = context.procesar(3,4);
17.
18.          System.out.println("Suma: " + suma );
19.          System.out.println("Resta: " + resta );
20.          System.out.println("Multiplicación: " + multip );
21.      }
22.  }
```

Establecer la interface que manejara las variables en las diferentes estrategias.

IAritmetica.java (una interface **IAritmetica** en el diagrama anterior):

```
01.  package Strategy;
02.
03.  public interface IAritmetica
04.  {
05.      public int realizarOperacion(int a, int b);
06.  }
```

Collapse Code

Definir estrategias concretas

Sumar.java (una **EstrategiaConcreta** en el diagrama anterior):

```
01.  package Strategy;
02.
03.  public class Sumar implements IAritmetica
04.  {
05.      public Sumar() {
06.      }
07.
08.      // ------------------------------
09.
10.      @Override
11.      public int realizarOperacion(int a, int b)
12.      {
13.          return a + b;
14.      }
15.  }
```

Collapse Code

Restar.java (una **EstrategiaConcreta** en el diagrama anterior):

text pop-up

```
01.  package Strategy;
02.
03.  public class Restar implements IAritmetica
04.  {
05.      public Restar() {
06.      }
07.
08.      // --------------------------------
09.
10.      @Override
11.      public int realizarOperacion(int a, int b)
12.      {
13.          return a - b;
14.      }
15.  }
```

Collapse Code

Multiplicar.java (una **EstrategiaConcreta** en el diagrama anterior):

```
01.  package Strategy;
02.
03.  public class Multiplicar implements IAritmetica
04.  {
05.      public Multiplicar() {
06.      }
07.
08.      // --------------------------------
09.
10.      @Override
11.      public int realizarOperacion(int a, int b)
12.      {
13.          return a * b;
14.      }
15.  }
```

Collapse Code

OperacionAritmetica.java (un **Contexto** en el diagrama anterior):

```
01.  package Strategy;
02.
03.  public class OperacionAritmetica
04.  {
05.      private IAritmetica strategy;
06.
07.      // ---------------------------------
08.
09.      public OperacionAritmetica(IAritmetica strategy) {
10.          this.strategy = strategy;
11.      }
12.
13.      // ---------------------------------
14.
15.      public int procesar(int a, int b) {
16.          return strategy.realizarOperacion(a, b);
17.      }
18.  }
```

text pop-up

Collapse Code

Resultado

```
Output - PatronesJava (run)
run:
Suma: 7
Resta: -1
Multiplicación: 12
BUILD SUCCESSFUL (total time: 0 seconds)
```

Bibliografía

- InformaticaPC (2013), Patrones de diseño, Strategy (https://informaticapc.com/patrones-de-diseno/strategy.php)

- Ortega Mateo Ezequiel (02 de enero del 2020), SOMOSPTN, Patrones de comportamiento, Strategy. (https://somospnt.com/blog/136-patrones-de-comportamiento-strategy)

- Blancarte Iturralde Oscar, (diciembre 2016) Introducción a los patrones de diseño, Strategy (https://reactiveprogramming.io/blog/es/patrones-de-diseno/strategy)