Centro Universitário UniBH
Ciência da Computação
Práticas de Programação
Professor: Lucas Schmidt

Aula Prática - 21 Padrões de Comportamento - Strategy

Intenção

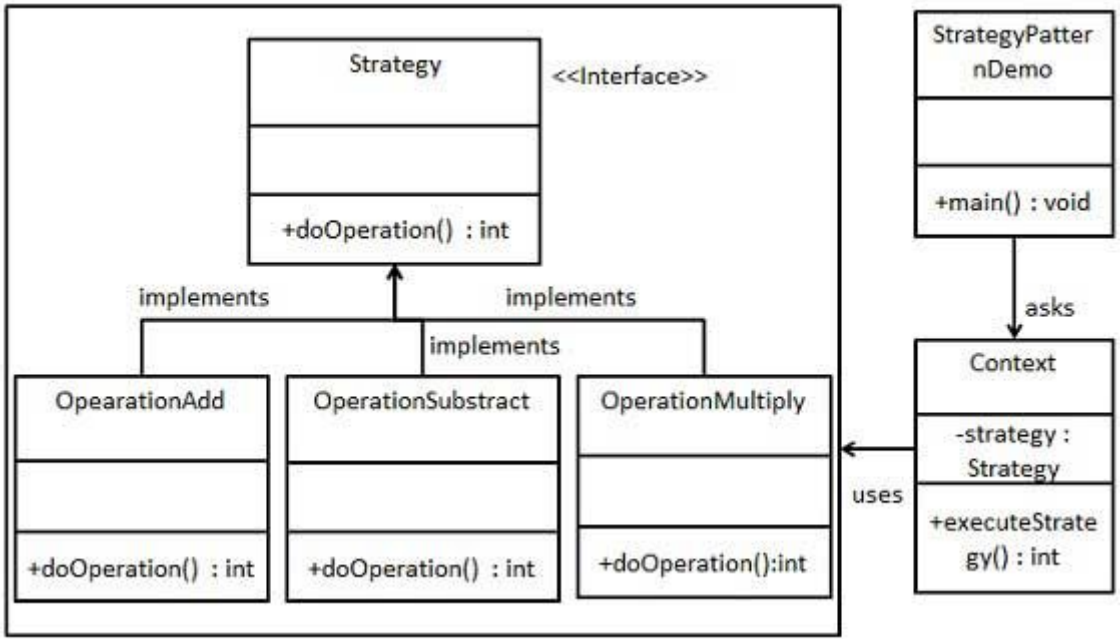
Definir uma família de algorítmos e permitir que um objeto possa escolher qual algorítmo da família utilizar em cada situação. Também conhecido como: Policy.

Usar este padrão quando…

- Várias classes diferentes diferem-se somente no comportamento;
- Você precisa de variantes de um mesmo algorítmo;
- Um algoritmo utiliza dados que o cliente não deve conhecer;
- Uma classe define múltiplos comportamentos, escolhidos num grande condicional.

Vantagens e desvantagens

- Famílias de algorítmos:
  - Beneficiam-se de herança e polimorfismo.
- Alternativa para herança do cliente:
  - Comportamento é a única coisa que varia.
- Eliminam os grandes condicionais:
  - Evita código monolítico.
- Escolha de implementações:
  - Pode alterar a estratégia em runtime.
- Clientes devem conhecer as estratégias:
  - Eles que escolhem qual usar a cada momento.
- Parâmetros diferentes para algoritmos diferentes:
  - Há possibilidade de duas estratégias diferentes terem interfaces distintas.
- Aumenta o número de objetos:
  - Este padrão aumenta a quantidade de objetos pequenos presentes na aplicação.



Passo 1
Crie uma interface.

Strategy.java

```
public interface Strategy {
   public int doOperation(int num1, int num2);
}
```

Passo 2
Crie classes concretas implementando a mesma interface.

OperationAdd.java

```
public class OperationAdd implements Strategy{
   @Override
   public int doOperation(int num1, int num2) {
      return num1 + num2;
   }
}
```

OperationSubstract.java

```
public class OperationSubstract implements Strategy{
   @Override
   public int doOperation(int num1, int num2) {
      return num1 - num2;
   }
}
```

OperationMultiply.java

```
public class OperationMultiply implements Strategy{
   @Override
   public int doOperation(int num1, int num2) {
      return num1 * num2;
   }
}
```

Passo 3
Criar classe de contexto.

Context.java

```
public class Context {
   private Strategy strategy;

   public Context(Strategy strategy){
      this.strategy = strategy;
   }

   public int executeStrategy(int num1, int num2){
      return strategy.doOperation(num1, num2);
   }
}
```

Passo 4
Use o contexto para ver a mudança de comportamento quando muda sua estratégia.

StrategyPatternDemo.java

```
public class StrategyPatternDemo {
   public static void main(String[] args) {
      Context context = new Context(new OperationAdd());
      System.out.println("10 + 5 = " + context.executeStrategy(10, 5));

      context = new Context(new OperationSubstract());
      System.out.println("10 - 5 = " + context.executeStrategy(10, 5));

      context = new Context(new OperationMultiply());
      System.out.println("10 * 5 = " + context.executeStrategy(10, 5));
   }
}
```

Passo 5

Teste sua implementação!