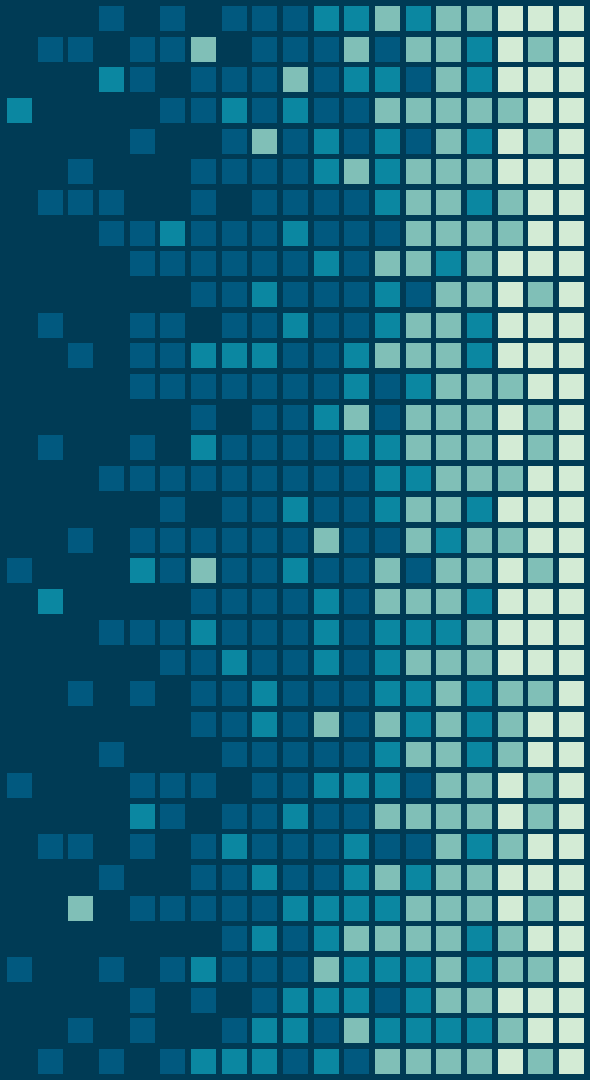
# Strategy – Design Pattern

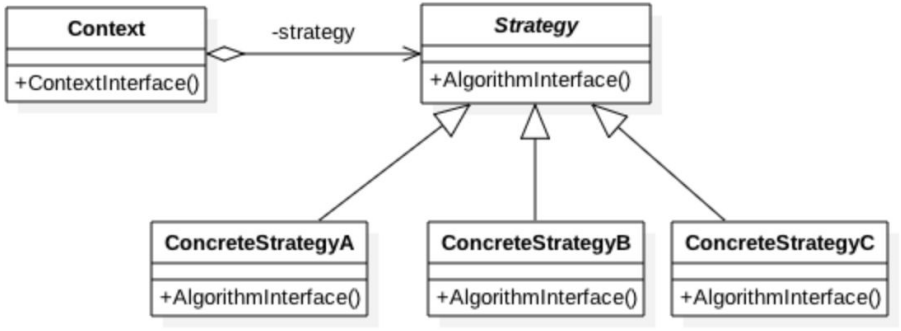


# Motivation

- Application de calcul
- Gère différentes opérations
- Opération spécifique = Algorithme précis

# Comment choisir dynamiquement un algorithme en particulier ?

Problème ☹




# Design Pattern Strategy

## Solution

# Intention

- Définir une famille d’algorithmes et les encapsuler
- Rendre interchangeables dynamiquement
- Changent indépendamment des clients qui l’emploient

# Implémentation

## 1. Créer une interface qui capture l'abstraction

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

# Implémentation

## 2. Créer les classes qui implémentent l'interface

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

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

# Implémentation

## 3. Créer un *context*

```
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);
   }
}
```

# Implémentation

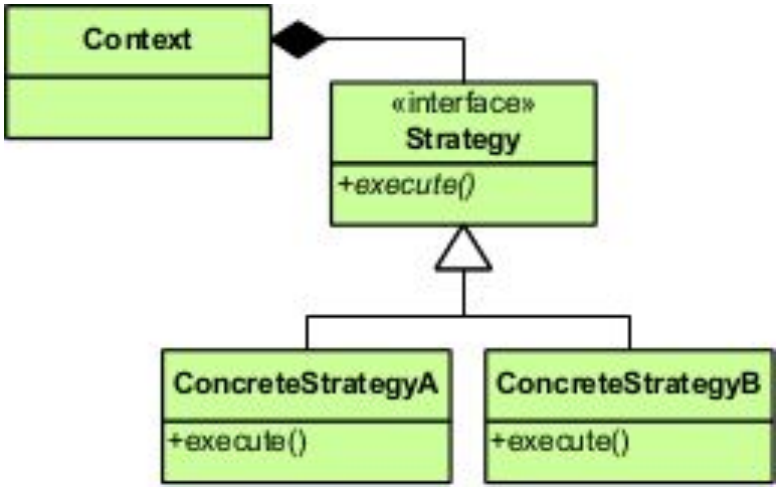
## 4. Utiliser le patron

```
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));
   }
}
```

# Diagramme UML



# Avantages

### Bonne pratique

Meilleure lisibilité et construction de code

### Extensibilité

Possibilité d'interchanger les algorithmes avec aisance

### Evolution

Code facile à maintenir en cas d'évolution

# Désavantages

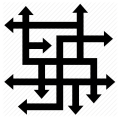
### Lourdeur

L'application doit maintenir 2 objets au lieu de un

(stratégie et contexte)

### Complexité

Augmente la complexité du programme avec l'ajout d'options



### Choix de stratégie

Le client doit connaître les différentes stratégies possibles pour faire son choix

# Sources

Tutorial point :
https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm

SourceMaking :
https://sourcemaking.com/design_patterns/strategy

Blog sur le design pattern :
https://softwareengineering.stackexchange.com/questions/302612/advantages-of-strategy-pattern?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa