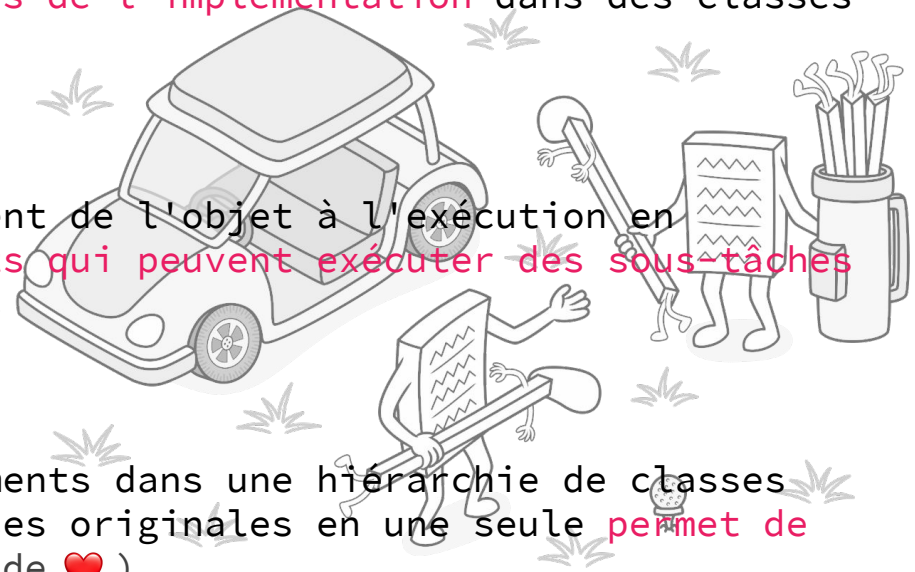
Design Pattern

Strategy

Elisa Gougerot & Victor Deyanovitch & Taj Singh

Strategy

- Design pattern comportemental permettant de cacher, dissimuler, d'enterrer (bon vous avez compris…) les détails de l'implémentation dans des classes dérivées

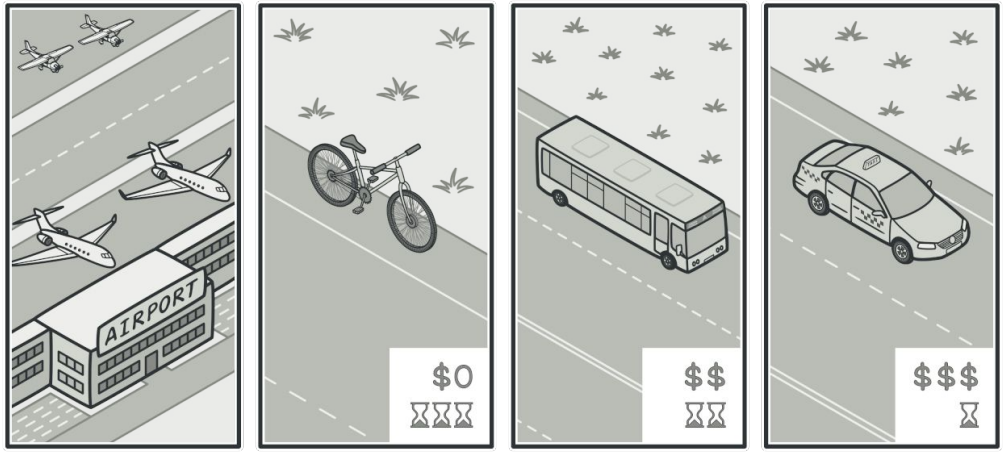


- Modifie indirectement le comportement de l'objet à l'exécution en l'associant à différents sous-objets qui peuvent exécuter des sous-tâches spécifiques de différentes manières

- Extraction des différents comportements dans une hiérarchie de classes séparée et la combinaison des classes originales en une seule permet de réduire les doublons (#duplicate code ❤️)

Analogie avec le monde réel

Imaginez que vous devez vous rendre à l'aéroport, vous pouvez prendre un bus, commander un taxi ou monter sur votre vélo



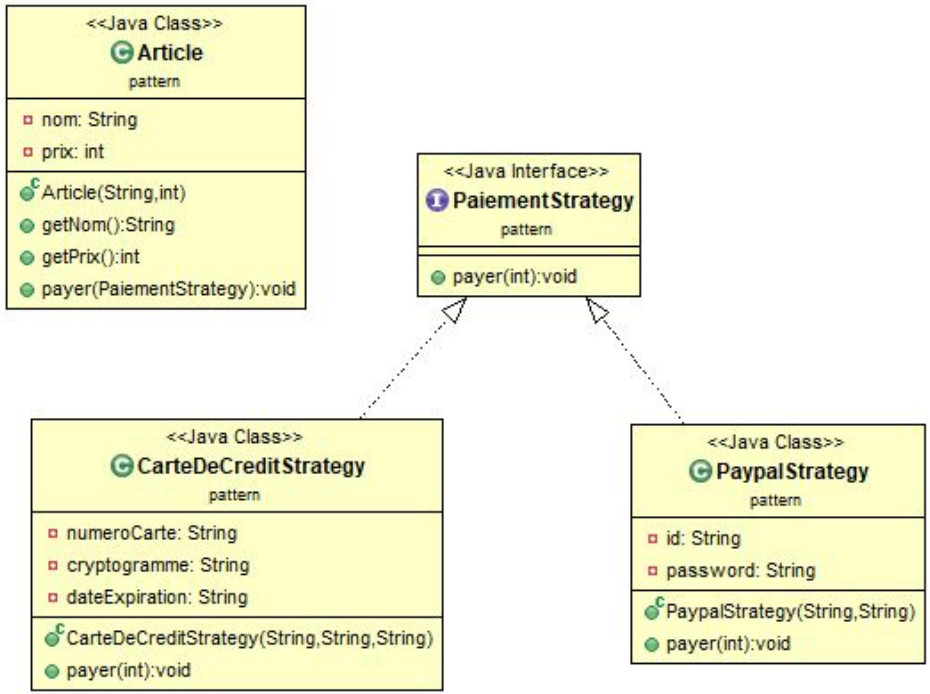
Ce sont vos stratégies de transport. Vous pouvez choisir l'une des stratégies en fonction de facteurs tels que le budget ou les contraintes de temps.

Exemple d'usage : Achat sur un site d'e-commerce

Un article vous intéresse, vous voulez l'acheter : vous l'ajoutez au panier. Vous ouvrez votre panier, et là, le site vous demande d'effectuer l'action abstraite de "payer".

Il est possible de payer de différentes manières : directement par carte, par Paypal...

La méthode "payer" peut être implémentée de différentes manières : c'est là l'intérêt du Pattern Strategy.



Checklist pour l'implémentation

- Identifier un algorithme qui est sujet à des changements fréquents
- Préciser l'interface de Strategy commune à toutes les variantes de l'algorithme
- Extraire un par un tous les algorithmes dans leurs propres classes
- Ajouter un champ pour stocker une référence à un objet de Strategy dans la classe de contexte

Exemple de code

Etape 1:

```java
//Strategy Interface
public interface CompressionStrategy {
  public void compressFiles(ArrayList<File> files);
}
```

Etape 2:

```java
public class ZipCompressionStrategy implements CompressionStrategy {
  public void compressFiles(ArrayList<File> files) {
    //using ZIP approach
  }
}
```

```java
public class RarCompressionStrategy implements CompressionStrategy {
  public void compressFiles(ArrayList<File> files) {
    //using RAR approach
  }
}
```

Exemple de code

Etape 3:

```
1 public class Client {
2   public static void main(String[] args) {
3     CompressionContext ctx = new CompressionContext();
4     //we could assume context is already set by preferences
5     ctx.setCompressionStrategy(new ZipCompressionStrategy());
6     //get a list of files...
7     ctx.createArchive(fileList);
8   }
9 }
```

Etape 4:

```
1  public class CompressionContext {
2    private CompressionStrategy strategy;
3    //this can be set at runtime by the application preferences
4    public void setCompressionStrategy(CompressionStrategy strategy) {
5      this.strategy = strategy;
6    }
7
8    //use the strategy
9    public void createArchive(ArrayList<File> files) {
10     strategy.compressFiles(files);
11   }
12 }
```

Les conséquences de son utilisation

- Isoler les détails de la mise en œuvre d'un algorithme → meilleure lisibilité du code
- Introduction de nouvelles stratégies sans avoir à changer le contexte → principe d'ouverture/fermeture
- Définir plusieurs algorithmes interchangeables dynamiquement

- Nécessite d’ajouter une classe
- Attention à ne pas surcompliquer le programme avec les nouvelles classes et interfaces qui accompagnent le modèle (dans le cas ou vous avez peu d’algorithmes ou si ils changent peu)

Strategy ≃ Command

- Strategy et Command peuvent se ressembler car vous pouvez utiliser les deux pour paramétrer un objet avec une certaine action

- L’objectif de Command est de convertir n'importe quelle opération en un objet. La conversion vous permet de différer l'exécution de l'opération, de la mettre en file d'attente, de stocker l'historique des commandes, d'envoyer des commandes à des services distants...

- L’objectif de Strategy est de décrire les différentes façons de faire la même chose, vous permettant d'échanger ces algorithmes dans une seule classe de contexte

Sources

- https://refactoring.guru/design-patterns/strategy.guru/design-patterns/strategy
- https://sourcemaking.com/design_patterns/strategy
- https://www.codingame.com/playgrounds/10741/design-pattern-strategy/presentation
- https://www.codingame.com/playgrounds/10741/design-pattern-strategy/exemple
- https://medium.com/elp-2018/strategy-design-pattern-76ee08bdb644