

# Design Pattern: Stratégie

---

Jonathan Petit

12 octobre 2018

ECAM - Architecture logicielle

# Design Pattern : Stratégie

---

- Design pattern

*Modèle de conception pour une solution reproductible à un problème récurrent*

- Accélère le processus de développement en fournissant un modèle testé et approuvé.

- 3 catégories

*Structurelle, Comportementale et de Création*

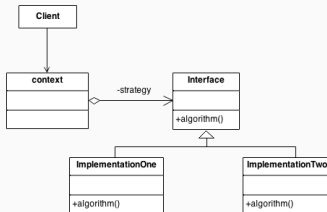
# Le design pattern stratégie

- Design pattern **Comportemental**  
*Encapsulation une famille d'algorithmes dans une classe.*
- **Indépendance** du client utilisateur  
*L'algorithme varie en fonction de l'utilisation d'un client*
- Cache les **détails d'implémentation**  
*Grâce à une abstraction réalisée dans une interface.*

- Plusieurs **choix** d'algorithmes  
*Quand une classe a différents choix d'algorithmes.*
- Plusieurs **procédures**  
*Quand une même tâche a plusieurs algorithmes.*

# Structure (1)

- Une interface
- Plusieurs implémentations de l'interface (les stratégies)
- Un contexte
- Un client



**Figure 1** – Structure : design pattern stratégie

## Structure (2)

- **Encapsulation** de l'interface  
*Les détails d'implémentation sont alors cachés dans les classes d'implémentation de l'interface.*
- **Aucun** impact sur le client  
*Si l'implémentation des stratégies change.*

```
1 class Strategy(metaclass=abc.ABCMeta):  
2     @abc.abstractmethod  
3     def algorithm_interface(self):  
4         pass
```

# L'interface

- L'interface avec les différentes **abstractions**  
*Les algorithmes représentant les différentes stratégies implémenteront les méthodes abstraites.*

```
1 class Strategy(metaclass=abc.ABCMeta):  
2     @abc.abstractmethod  
3     def algorithm_interface(self):  
4         pass
```



# Les stratégies

- Les différents algorithmes stratégies implémente l'interface  
*Les stratégies implémente les méthodes de l'interface afin de cacher les détails aux utilisateurs.*

```
1 class ConcreteStrategyA (Strategy):
2     """
3     Une implementation l'interface.
4     """
5
6     def algorithm_interface(self):
7         pass
8
9 class ConcreteStrategyB (Strategy):
10     """
11     Une autre implementation de l'interface.
12     """
13
14     def algorithm_interface(self):
15         pass
```

- Le contexte dirige les clients

*Redirection par l'interface vers les différentes stratégies.*

```
1 class Context:
2     def __init__(self, strategy):
3         self._strategy = strategy
4
5     def context_interface(self):
6         self._strategy.algorithm_interface()
```

## Exemple d'utilisation

- Application de **payements**

*Plusieurs moyens de payements, par carte, en cash, ...*

- Application de **trajet**

*Un client peut choisir de faire un trajet en bus, en train, en voiture, ...*

## Application (1) : Description

- "Tout le monde veut prendre ça place"  
*Application basique du jeu télévisé.*
- Plusieurs **procédures** de réponses  
*Un participant peut choisir entre plusieurs méthodes de réponse (DUO, CARRE ou CASH.)*
- Le score **dépend** de la méthode  
*En fonction de la méthode de réponse, le joueur aura plus ou moins de points*

## Application (2) : L'interface

- La **stratégie** de réponse

*Méthode pour le choix de la procédure de réponse (DUO, CARRE, CASH).*

- Le **score**

*Méthode pour le cumul des points en fonction de la stratégie de réponse.*

```
1 class QuestionStrategyAbstract(object, metaclass=ABCMeta):
2     @abstractmethod
3     def strategy_answer(self, incorrect_answers, correct_answer):
4         pass
5
6     @abstractmethod
7     def score(self):
8         pass
```

## Application (3) : Exemple d'une stratégie

- La **stratégie** DUO

*Nombres de réponses limitées à 2 pour un choix aléatoire dans les réponses possibles.*

- Le **score**

*Ajout du score de 1 point en cas de bonne réponse.*

```
1 class StrategyDuo(QuestionStrategyAbstract):
2     def strategy_answer(self, incorrect_answers, correct_answer):
3         proposals = []
4         choice = random.choice(incorrect_answers)
5         proposals.append(choice)
6         proposals.append(correct_answer)
7         return proposals
8
9     def score(self):
10        return 1
```

## Application (4) : Démo

- Choix de la méthode de réponses

```
What is the capital of Jamaica?  
1. DUO  
2. SQUARRE  
3. CASH  
>>> █
```

**Figure 2** – Exemple de choix de la méthode

- Nombre de réponses limitées pour DUO

```
What is the capital of Jamaica?  
- Rio de Janeiro  
- Kingston  
>>> █
```

**Figure 3** – Exemple de réponse

# Conclusion (1)

Le pattern stratégie est utilisé :

- Plusieurs choix possibles par **une classe utilisateur**  
*Différents choix possibles pour l'utilisateur. Exemple :  
procédure de réponses.*
- Une tâche se divise en **plusieurs algorithmes**  
*Exemple : Ajout de points en fonction d'un choix de réponse.*



## Conclusion (2)

Le pattern stratégie permet :

- Cacher **les détails** d'implémentation  
*Encapsulation des méthodes abstraites dans une interface.*
- Diviser une tâche en **plusieurs algorithmes**  
*Différentes stratégies en fonction de certains paramètres pour une tâche*


# Enjoy !

L'application est disponible sur GitHub :

- <https://github.com/JonathanPetit/Strategy-design-pattern>
- Pour lancer l'application README.md

# Bibliography i

 [https://sourcemaking.com/design\\_patterns/strategy](https://sourcemaking.com/design_patterns/strategy)

 [https://medium.com/@sheikhsajid/  
design-patterns-in-python-part-1-the-strategy-pattern-](https://medium.com/@sheikhsajid/design-patterns-in-python-part-1-the-strategy-pattern-)

 [https://medium.com/@iimam/  
design-patterns-with-python-strategy-pattern-f4c076b5dc](https://medium.com/@iimam/design-patterns-with-python-strategy-pattern-f4c076b5dc)

 Architecture logiciel slides - Mr. Combéfis

 <https://github.com/matze/mtheme>