

---

# Programmation orientée objet

## TD 4 : Design Pattern

---

Nous allons ici utiliser des *design pattern*. Ce sont des façons d'organiser ses classes en vue de résoudre des problèmes courants en développement.

### 1 Observer

Le pattern *observer* permet de faire fonctionner la programmation événementielle. Il est utile lorsque plusieurs sources sont dépendantes d'une donnée et veulent être informé lorsque sa valeur change.

#### 1.1 Score à jour

Supposons que l'on possède une classe `MatchDeFootball`, chaque match réel étant un objet de cette classe. On voudrait être tenu au courant à chaque but. Proposez donc :

1. un modèle minimal de la classe `MatchDeFoot` pour notre problème
2. un moyen pour des objets extérieurs d'être informés d'un but en temps réel
3. codez ces classes en python

#### 1.2 Observer et UI

Comment mettre en place le mécanisme précédent pour la gestion d'interfaces ?

### 2 Memento

Le pattern *memento* permet de sauvegarder une information et de la redonner via une méthode `restore` plus tard dans le temps. Ce pattern est souvent la première brique d'une méthode d'UNDO/REDO.

Proposez une modélisation UML de ce pattern. Déclinez-le pour stocker/replacer la valeur d'un dé.

#### 2.1 Undo

Proposez un modèle UML et une façon de faire algorithmique du UNDO.

## 2.2 Redo

Que faudrait-il ajouter pour faire également un REDO ?

## 3 Iterator

Ce pattern permet d'accéder itérativement à des objets. Il est très présent en python et on peut facilement en créer pour nos objets.

### 3.1 Modèle UML

En utilisant le code suivant (tiré de <https://www.programiz.com/python-programming/iterator>) essayez de construire un modèle UML d'un itérateur.

```
# define a list
my_list = [4, 7, 0, 3]
my_iter = iter(my_list) # get an iterator using iter()

## iterate through it using next()
print(next(my_iter)) #prints 4
print(next(my_iter)) #prints 7

## next(obj) is same as obj.__next__()
print(my_iter.__next__()) #prints 0
print(my_iter.__next__()) #prints 3

## This will raise error, no items left
next(my_iter) # StopIteration exception
```

### 3.2 Itération pour les dés

Créer une classe itérateur prenant un dé et un entier  $n$  comme paramètres et permettant de lancer le dé  $n$  fois.

Vous proposerez le modèle UML, le code python de la classe et une façon de l'implémenter dans la classe Dé.