

# CPOA - Sujet TD 3

PreReq	<ol style="list-style-type: none"><li>1. Je sais programmer en <a href="#">Java</a>.</li><li>2. J'ai conscience qu'il faut réfléchir avant de se lancer dans le codage.</li><li>3. Je maîtrise les concepts objet de base (héritage, polymorphisme, ...).</li><li>4. J'ai compris ce qu'est un patron et j'ai grand soif d'en apprendre d'autres que <i>Strategy</i> et Singleton</li></ol>
ObjTD	Aborder le patron <b>fabrique</b> .
Durée	1 TD et 2 TP de 1,5h (sur 2 semaines).

## 1. La pizzeria O'Reilly

Vous êtes embauché dans une pizzeria pour faire ... de l'informatique (il y en a bien qui font leur PTUT pour une boulangerie...)!

Le stagiaire de l'an dernier qui avait travaillé sur le code est parti avec la caisse (de Chianti). Vous n'avez à votre disposition que :

1. Le code de départ suivant :

```

/**
 * @author bruel (from O'Reilly Head-First series)
 */
public class Pizzeria {

    public Pizza commanderPizza(String type) {

        Pizza pizza;

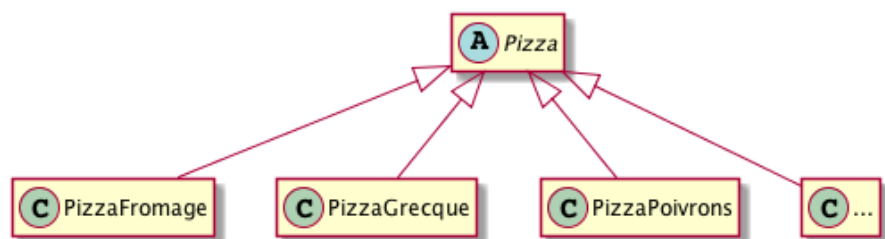
        if (type.equals("fromage")) {
            pizza = new PizzaFromage();
        } else if (type.equals("grecque")) {
            pizza = new PizzaGrecque();
        } else {
            pizza = new PizzaPoivrons();
        }

        pizza.preparer();
        pizza.cuire();
        pizza.couper();
        pizza.emballer();

        return pizza;
    }
}

```

2. L'ébauche de diagramme de classe des pizzas suivant :



3. Le bout de code de test suivant :

```

Pizzeria boutiqueBrest = new Pizzeria();
boutiqueBrest.commanderPizza("fromage");
...
Pizzeria boutiqueStrasbourg = new Pizzeria();
boutiqueStrasbourg.commanderPizza("grecque");

```



#### QUESTION

1. Identifiez ce qui varie dans ce code (si la pression du marché fait ajouter des pizzas à la carte ou si une pizza n'a plus de succès et doit disparaître, etc.).
2. Isolez dans une classe `SimpleFabriqueDePizzas` ce code.
3. Réalisez le diagramme de classe obtenu.
4. Quel est l'avantage de procéder ainsi ? Ne transfère-t'on pas simplement le problème à un autre objet ?



Bien sûr vous héritez de cet horrible "if then else" et dans votre implémentation en TP vous remplacerez ce code avantageusement par un "switch case" et utiliserez une `enum` comme link: [vu en cours](#).

## 2. On y est presque...

Nous sommes arrivés à une situation propre, qui s'apparente à un patron de conception. Mais avant d'en arriver à la définition du patron lui-même, nous allons améliorer un peu les choses.

### 2.1. Succès des pizzerias O'Reilly : les franchises

Plusieurs villes veulent ouvrir des pizzerias comme la vôtre. Votre patron, très content de vos programmes souhaite imposer à toutes les futures pizzerias d'utiliser vos codes.

Le problème : les pizzas au fromage de Strasbourg sont différentes des pizzas aux fromages de Corse!



#### QUESTION

Proposez une solution où `SimpleFabriqueDePizzas` serait une classe abstraite.

### 2.2. La dérive : chacun travaille comme il l'entend!

Les pizzerias utilisent bien vos fabriques mais ont changé leurs procédures : certaines ne coupent pas les pizzas, changent les temps de cuissons, et les pizzerias O'Reilly perdent leur identité. Il nous faut donc **restructurer** les pizzerias.

Un consultant italien payé fort cher (heureusement en pizzas!) propose de revenir à la structure suivante :

```

public abstract class Pizzeria {
    public final Pizza commanderPizza(String type) {
        Pizza pizza;

        pizza = creerPizza(type);
        pizza.preparer();
        pizza.cuire();
        pizza.couper();
        pizza.emballer();

        return pizza;
    }

    ..... Pizza creerPizza(String type);
}

```



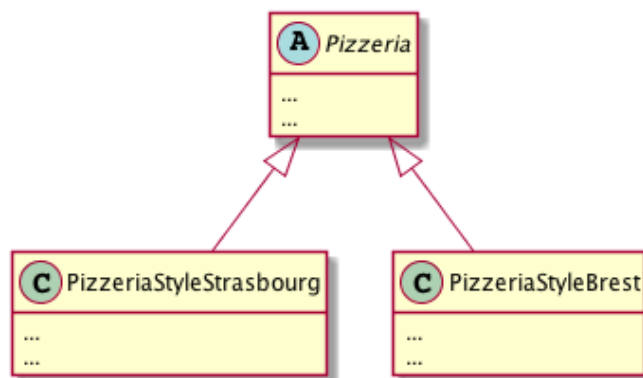
#### QUESTION

Quelles sont les différences avec notre conception actuelle?

## 2.3. Laisser les sous-classes décider

#### QUESTION

Dans le schéma suivant, placez les méthodes au bon endroit de façon à ce que les procédures soient respectées tout en ayant des pizzas à variantes "régionales".



## 2.4. Déclarer une méthode de fabrique

Rien qu'en apportant une ou deux transformations à `Pizzeria`, nous sommes passés d'un objet gérant l'instanciation de nos classes concrètes à un ensemble de sous-classes qui assument maintenant cette responsabilité.



#### QUESTION

Quelle est la déclaration exacte de la méthode `creerPizza()` de la classe `Pizzeria` ?

## 2.5. Récapitulons



### QUESTION

Donnez le diagramme de séquence d'une "commande de pizza au fromage de type Strasbourg".



Vous implémenterez les classes manquantes en TP.

## 3. Le patron Fabrique (simple)

Nous y sommes, vous venez de décortiquer le patron Fabrique Simple

### Design pattern : Fabrique (simple)

**Fabrique** (simple) définit une interface pour la création d'un objet, mais en laissant à des sous-classes le choix des classes à instancier (voir aussi [Fabrique abstraite](#)).

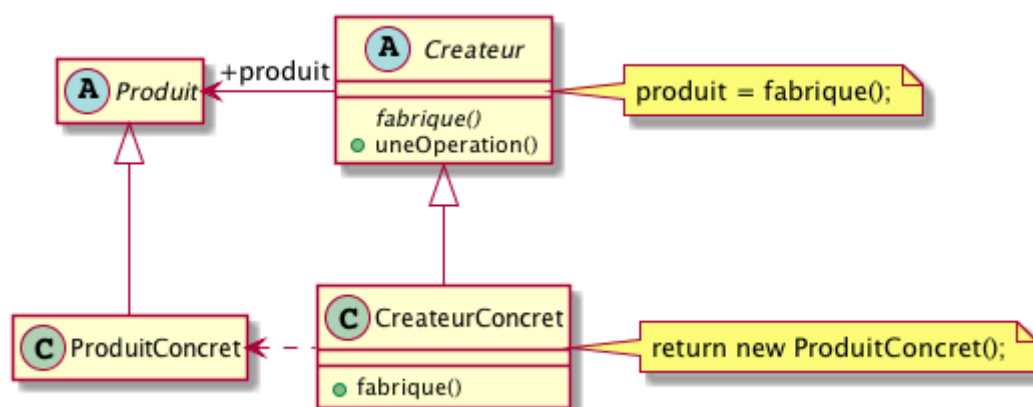


Figure 1. Modèle UML du patron Fabrique

## Pour aller plus loin

### Et les pizzas dans tout ça !?



### QUESTION

Proposez un diagramme de classe UML pour les pizzas (classes, attributs et méthodes).

## Et sans patron, ça donne quoi ?

Un stagiaire de 2013 (les patrons n'étaient pas au programme du PPN!) a réalisé le programme suivant :

```

public class PizzeriaDependante {
    public Pizza creerPizza(String style, String type) {
        Pizza pizza = null;

        if (style.equals("Brest")) {
            if (type.equals("fromage")) {
                pizza = new PizzaFromageStyleBrest();
            } else if (type.equals("vegetarienne")) {
                pizza = new PizzaVegetarienneStyleBrest();
            } else if (type.equals("fruitsDeMer")) {
                pizza = new PizzaFruitsDeMerStyleBrest();
            } else if (type.equals("poivrons")) {
                pizza = new PizzaPoivronsStyleBrest();
            }
        } else if (style.equals("Strasbourg")) {
            if (type.equals("fromage")) {
                pizza = new PizzaFromageStyleStrasbourg();
            } else if (type.equals("vegetarienne")) {
                pizza = new PizzaVegetarienneStyleStrasbourg();
            } else if (type.equals("fruitsDeMer")) {
                pizza = new PizzaFruitsDeMerStyleStrasbourg();
            } else if (type.equals("poivrons")) {
                pizza = new PizzaPoivronsStyleStrasbourg();
            }
        } else {
            System.out.println("Erreur : type de pizza invalide");
            return null;
        }
        pizza.preparer(); pizza.cuire(); pizza.couper(); pizza.emballer();
        return pizza;
    }
}

```



#### QUESTION

1. Faites le compte du nombre de classes concrètes dont cette classe dépend.
2. Et si vous ajoutez des pizzas de style Marseille à cette Pizzeria ?

## Problème du main de test du jeu d'aventure

Vous avez sûrement dans votre `main` de l'application de jeu d'aventure une partie du code ressemblant à ceci :

```
if (choix.equals("Epee")) {  
    perso.setArme(new ComportementEpee());  
}  
else if (choix.equals("Arc")) {  
    perso.setArme(new ComportementArc());  
    else if ...  
    ...  
}
```

Ce code est peu adaptatif et va souffrir des évolutions, par exemple :

- changement de la liste des armes possibles
- rajouter des `if then else` à chaque nouvelle arme
- suppression de certaines armes
- ...

#### QUESTION

1. Isoler ce code dans une classe `SimpleFabriqueArme` qui possèdera une méthode `creerComportementArme(String type)` qui retourne le comportement adapté en fonction du paramètre reçu.
2. Donnez le diagramme de classe `UML™` de la nouvelle organisation.
3. Quelles différences a-t-on avec le patron Fabrique ? Donner quelques exemples d'extension du jeu d'aventure qui correspondraient à l'usage d'une Fabrique.
4. Donnez le diagramme de séquence du main. Par exemple avec le code de test suivant :



```
Chevalier perso = new Chevalier("JMI");  
  
SimpleFabriqueArme fabrique = new SimpleFabriqueArme();  
ComportementArme c = fabrique.creerComportementArme("Epee");  
  
perso.setArme(c);  
perso.frapper();
```