



ECOLE NORMALE SUPÉRIEURE DE L'ENSEIGNEMENT
TECHNIQUE DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

المدرسة العليا لأساتذة التعليم التقني المحمدية

Design Patterns

2^{ème} année Cycle Ingénieur

GLSID 2, ICCN 2 & IIBDCC 2

Pr. SARA RETAL



Identifier les aspects qui varient

Disons que vous possédez une boutique à Objectville et que vous vendez des pizzas. Pour rester à la pointe de la technologie, vous écrirez peut-être un programme comme celui-ci :

```
Pizza commanderPizza() {  
    Pizza pizza = new Pizza();  
  
    pizza.preparer();  
    pizza.cuire();  
    pizza.couper();  
    pizza.emballer();  
    return pizza;  
}}
```

Type de pizza

```
Pizza commanderPizza(String type) {  
    Pizza pizza;
```

Maintenant, nous transmettons le type de pizza à commanderPizza().

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

Selon le type de pizza, nous instancions la bonne classe concrète et nous l'affectons à chaque variable d'instance pizza. Notez que chaque pizza doit implémenter l'interface Pizza.

```
    pizza.preparer();  
    pizza.cuire();  
    pizza.couper();  
    pizza.emballer();  
    return pizza;  
}
```

Une fois que nous avons une Pizza, nous la préparons (vous savez, étaler la pâte, mettre la sauce et ajouter garnitures et fromage). Puis nous la faisons cuire, nous la coupons et nous la mettons dans une boîte

Chaque sous-type de Pizza (PizzaFromage, PizzaPoivrons, etc.) sait comment se préparer lui-même!

Ajouter d'autres types de pizza

Ce code n'est PAS fermé à la modification. Si vous changez votre carte, vous devrez reprendre ce code et le modifier.


```
Pizza commanderPizza(String type) {  
    Pizza pizza;  
  
    if (type.equals("fromage")) {  
        pizza = new PizzaFromage();  
    } else if (type.equals("grecque")) {  
        pizza = new PizzaGrecque();  
    } else if (type.equals("poivrons")) {  
        pizza = new PizzaPoivrons();  
    } else if (type.equals("fruitsDeMer")) {  
        pizza = new PizzaFruitsDeMer();  
    } else if (type.equals("vegetarienne")) {  
        pizza = new PizzaVegetarienne();  
    }  
  
    pizza.preparer();  
    pizza.cuire();  
    pizza.couper();  
    pizza.emballer();  
    return pizza;  
}
```

Voici ce qui varie. Comme le type de pizza change avec le temps, vous n'allez pas cesser de modifier ce code.

Voici ce qui ne devrait pas changer. En majeure partie, la préparation, la cuisson et l'emballage d'une pizza n'ont pas varié depuis des lustres. Ce n'est donc pas ce code qui changera, mais seulement les pizzas sur lesquelles il opère.

Encapsuler la création des objets

```
Pizza commanderPizza(String type) {  
    Pizza pizza;  
  
    pizza.preparer();  
    pizza.cuire();  
    pizza.couper();  
    pizza.emballer();  
    return pizza;  
}
```



Qu'allons-nous placer ici ?

Nous avons un nom pour ce nouvel objet :
nous l'appelons une Fabrique.

Une simple fabrique de pizzas

Voici notre nouvelle classe, la SimpleFabriqueDePizzas.
Elle n'a qu'une seule chose à faire dans la vie : créer des pizzas pour ses clients.

Nous définissons d'abord une méthode `creerPizza()` dans la fabrique. C'est la méthode que tous les clients utiliseront pour créer de nouvelles instances.

```
public class SimpleFabriqueDePizzas {  
    public Pizza creerPizza(String type) {  
        Pizza pizza = null;  
  
        if (type.equals("fromage")) {  
            pizza = new PizzaFromage();  
        } else if (type.equals("poivrons")) {  
            pizza = new PizzaPoivrons();  
        } else if (type.equals("fruitsDeMer")) {  
            pizza = new PizzaFruitsDeMer();  
        } else if (type.equals("vegetarienne")) {  
            pizza = new PizzaVegetarienne();  
        }  
        return pizza;  
    }  
}
```

Voici le code que nous avons extrait de la méthode `commanderPizza()`.

Ce code est toujours paramétré par le type de pizza, tout comme l'était notre méthode `commanderPizza()` d'origine.

Question

- Quel est l'avantage de procéder ainsi ? On dirait qu'on transfère simplement le problème à un autre objet.

Question

- J'ai vu une conception similaire dans laquelle une fabrique comme celle-ci est définie comme une méthode statique.

Quelle est la différence ?

Une simple fabrique de pizzas

```
public class Pizzeria {  
    SimpleFabriqueDePizzas fabrique;  
  
    public Pizzeria(SimpleFabriqueDePizzas fabrique) {  
        this.fabrique = fabrique;  
    }  
  
    public Pizza commanderPizza(String type) {  
        Pizza pizza;  
  
        pizza = fabrique.creerPizza(type);  
  
        pizza.preparer();  
        pizza.cuire();  
        pizza.couper();  
        pizza.emballer();  
        return pizza;  
    }  
    // autres méthodes  
}
```

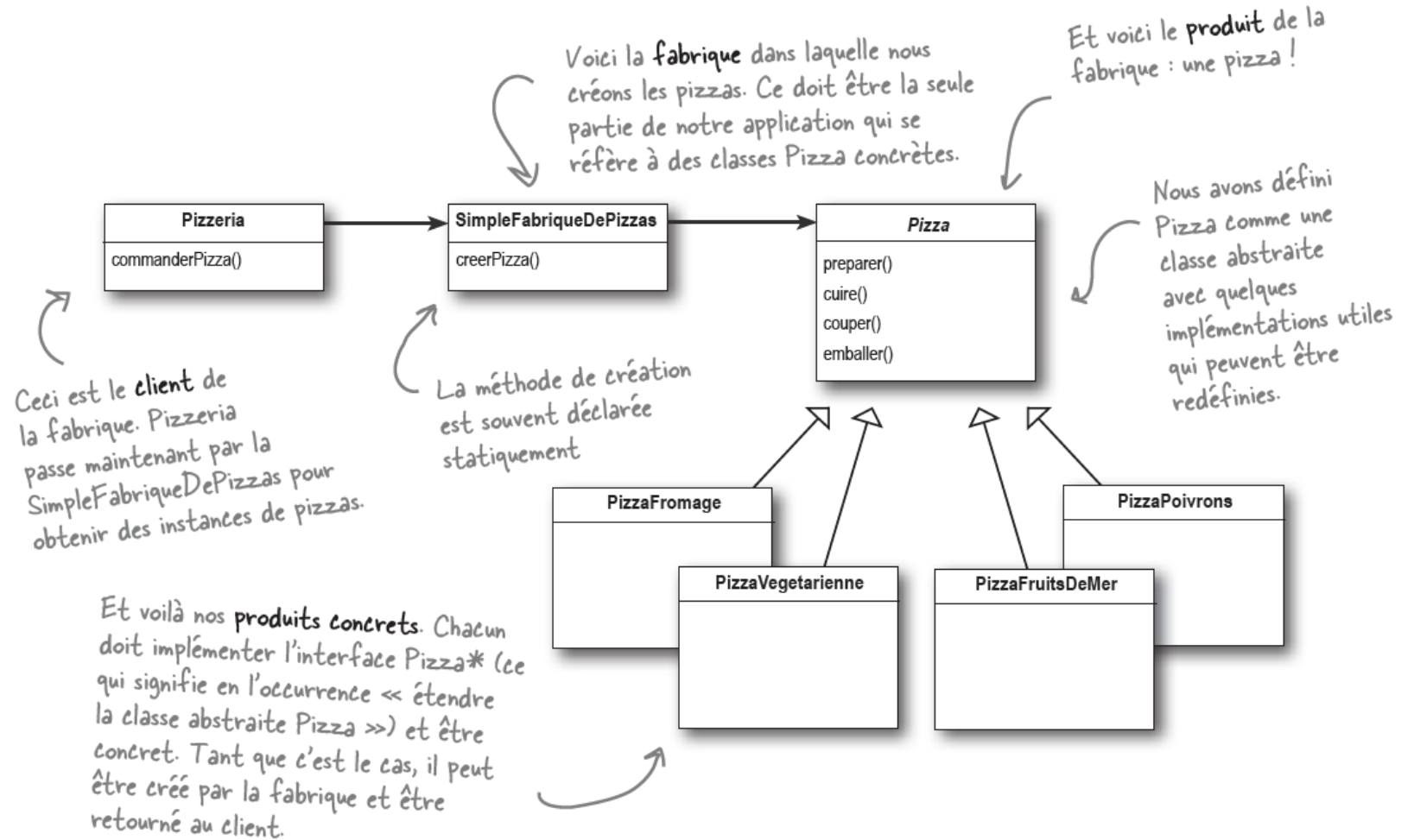
Nous donnons maintenant à Pizzeria une référence à une SimpleFabriqueDePizzas.

Nous transmettons la fabrique à Pizzeria dans le constructeur.

Et la méthode commanderPizza() utilise la fabrique pour créer ses pizzas en transmettant simplement le type de la commande.

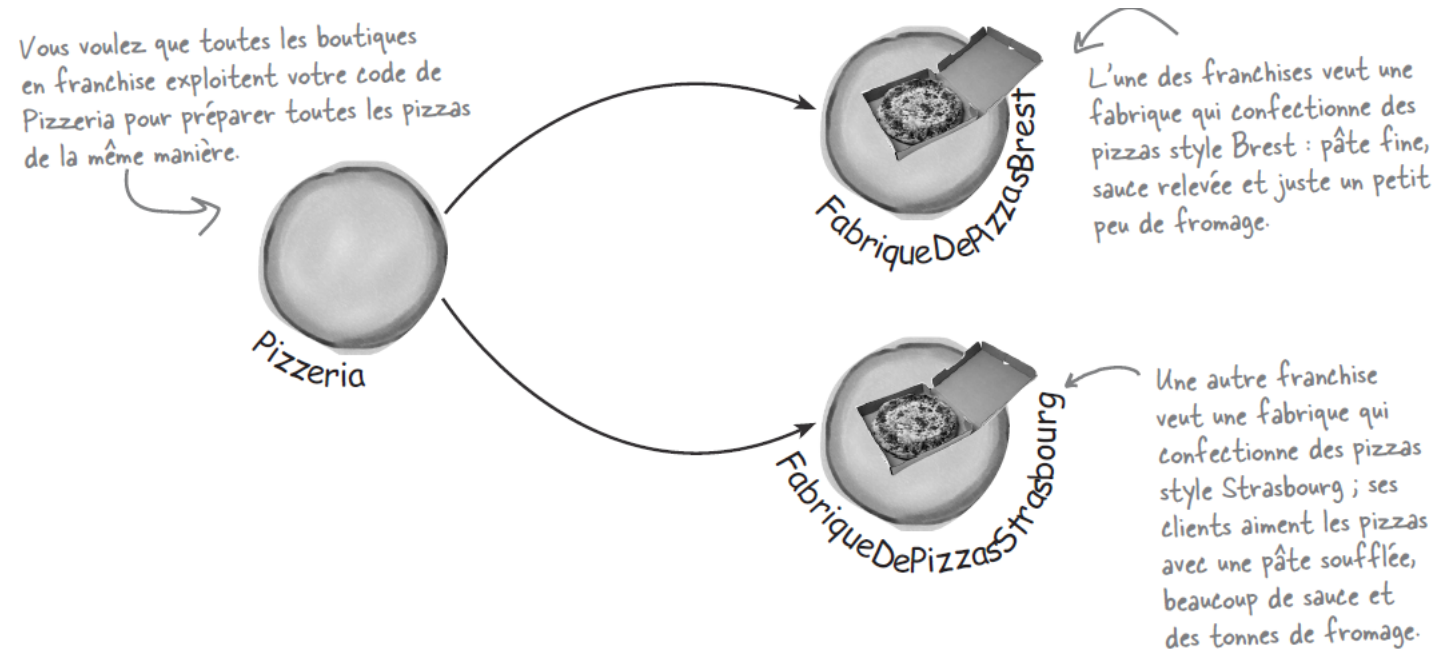
Remarquez que nous avons remplacé l'opérateur new par une méthode de création de l'objet fabrique. Nous n'avons plus d'instanciations concrètes !

Diagramme de classes de notre Fabrique simple



Franchiser les pizzerias

- Votre Pizzeria d'Objectville a connu une telle. Maintenant, chacun veut une Pizzeria à proximité de chez lui. En tant que franchiseur, vous voulez vous assurer de la qualité de la production des franchises et vous voulez qu'ils utilisent votre code.
- Mais qu'en est-il des différences régionales ? Chaque franchise peut souhaiter proposer différents styles de pizzas (Brest, Strasbourg, et Marseille par exemple), selon son emplacement géographique et les goûts des amateurs de pizza locaux.



Franchiser les Pizzerias

```
FabriqueDePizzasBrest fabriqueBrest = new FabriqueDePizzasBrest();  
Pizzeria boutiqueBrest = new Pizzeria(fabriqueBrest);  
boutiqueBrest.commander("Végétarienne");
```

← Ici, nous créons une fabrique pour faire des pizzas style Brest.

← Puis nous créons un Pizzeria et nous lui transmettons une référence à la fabrique Brest.

← ... et quand nous faisons des pizzas, nous obtenons des pizzas brestoises.

```
FabriqueDePizzasStrasbourg fabriqueStrasbourg = new FabriqueDePizzasStrasbourg();  
Pizzeria boutiqueStrasbourg = new Pizzeria(fabriqueStrasbourg);  
boutiqueStrasbourg.commander("Végétarienne");
```

↑ Pareil pour Strasbourg : nous créons une fabrique de pizzas style Strasbourg et nous créons une Pizzeria composée avec une fabrique Strasbourg. Quand nous créons des pizzas, elles ont le goût de celles de Strasbourg.

Une structure pour la Pizzeria

```
public abstract class Pizzeria {
```

```
    public Pizza commanderPizza(String type) {  
        Pizza pizza;
```

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

```
        return pizza;
```

```
    }
```

```
    abstract creerPizza(String type);
```

```
}
```

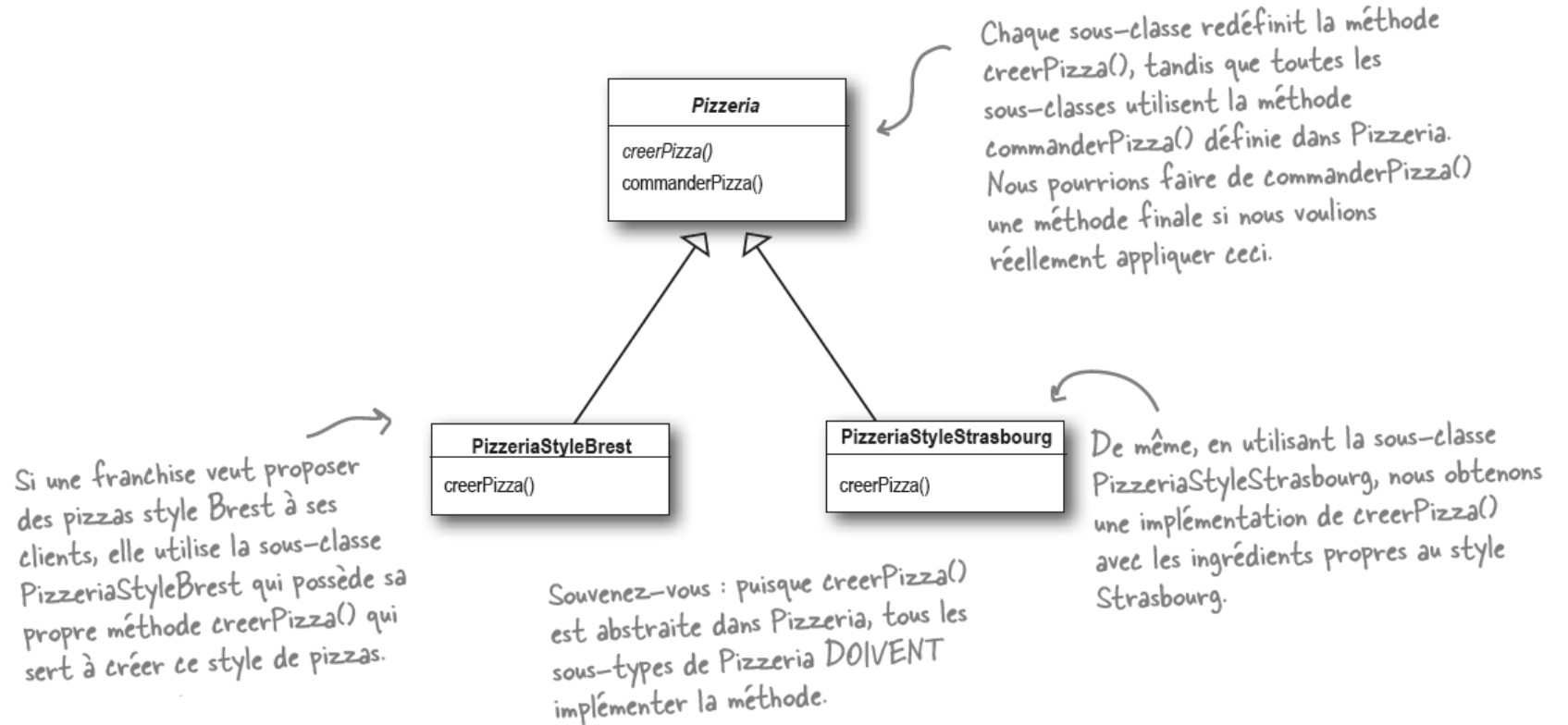
Maintenant, `creerPizza()` est de nouveau un appel à une méthode de `Pizzeria` et non à un objet fabrique.

Ceci ne change pas...

Et nous avons transféré notre objet fabrique à cette méthode.

Notre « méthode de fabrication » est maintenant abstraite dans `Pizzeria`.

Une structure pour la Pizzeria



Une structure pour la pizzeria de Brest

Voici le style Brest :

creerPizza() retourne une Pizza, et la sous-classe a l'entière responsabilité de déterminer la Pizza concrète qu'elle instancie

Comme PizzeriaBrest étend Pizzeria, elle hérite (entre autres) de la méthode commanderPizza().

```
public class PizzeriaBrest extends Pizzeria {  
    Pizza creerPizza(String item) {  
        if (choix.equals("fromage")) {  
            return new PizzaFromageStyleBrest();  
        } else if (choix.equals("vegetarienne")) {  
            return new PizzaVegetarienneStyleBrest();  
        } else if (choix.equals("fruitsDeMer")) {  
            return new PizzaFruitsDeMerStyleBrest();  
        } else if (choix.equals("poivrons")) {  
            return new PizzaPoivronsStyleBrest();  
        } else return null;  
    }  
}
```

← Nous devons implémenter creerPizza() puisqu'elle est abstraite dans Pizzeria.

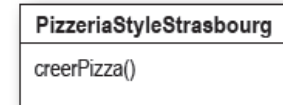
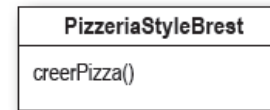
← Voici l'endroit où nous créons nos classes concrètes. Pour chaque type de Pizza nous créons le style Brest.

** Notez que la méthode commanderPizza() de la superclasse ne dispose d'aucun indice sur la Pizza que nous créons : elle sait simplement qu'elle peut la préparer, la faire cuire la découper et l'emballer !*

Déclarer une méthode de fabrique

```
public abstract class Pizzeria {  
  
    public Pizza commanderPizza(String type) {  
        Pizza pizza;  
  
        pizza = creerPizza(type);  
  
        pizza.preparer();  
        pizza.cuire();  
        pizza.couper();  
        pizza.emballer();  
  
        return pizza;  
    }  
  
    protected abstract Pizza creerPizza(String type);  
  
    // autres méthodes  
}
```

Les sous-classes de Pizzeria gèrent l'instanciation des objets à notre place dans la méthode `creerPizza()`.



Toute la responsabilité de l'instanciation des Pizzas a été transférée à une méthode qui se comporte comme une fabrique.

Classe pizza

```
public abstract class Pizza {  
    String nom;  
    String pate;  
    String sauce;  
    ArrayList garnitures = new ArrayList();  
  
    void preparer() {  
        System.out.println("Préparation de " + nom);  
        System.out.println("Étalage de la pâte...");  
        System.out.println("Ajout de la sauce...");  
        System.out.println("Ajout des garnitures: ");  
        for (int i = 0; i < garnitures.size(); i++) {  
            System.out.println(" " + garnitures.get(i));  
        }  
    }  
  
    void cuire() {  
        System.out.println("Cuisson 25 minutes à 180°");  
    }  
  
    void couper() {  
        System.out.println("Découpage en parts triangulaires");  
    }  
  
    void emballer() {  
        System.out.println("Emballage dans une boîte officielle");  
    }  
  
    public String getNom() {  
        return nom;  
    }  
}
```

La classe abstraite fournit des comportements par défaut pour la cuisson, le découpage et l'emballage.

La préparation suit un certain nombre d'étapes ordonnées selon une suite particulière.

Classe pizza

```
public class PizzaFromageStyleBrest extends Pizza {  
    public PizzaFromageStyleBrest() {  
        nom = "Pizza sauce style brest et fromage";  
        pate = "Pâte fine"; sauce = "Sauce Marinara";  
        garnitures.add("Parmigiano reggiano râpé");  
    }  
}
```

La Pizza brestoise a sa propre sauce marinara et une pâte fine.



Et une garniture, du parmigiano reggiano !



La Pizza Strasbourg a une sauce aux tomates cerise et une pâte très épaisse.

```
public class PizzaFromageStyleStrasbourg extends Pizza {  
  
    public PizzaFromageStyleStrasbourg() {  
        nom = "Pizza pâte style Strasbourg et fromage";  
        pate = "Extra épaisse";  
        sauce = "Sauce aux tomates cerise";  
        garnitures.add("Lamelles de mozzarella");  
    }  
  
    void couper() {  
        System.out.println("Découpage en parts carrées");  
    }  
}
```



La Pizza Strasbourg a des tonnes de mozzarella !



La pizza Strasbourg redéfinit également la méthode couper() afin de découper des parts carrées.

Test

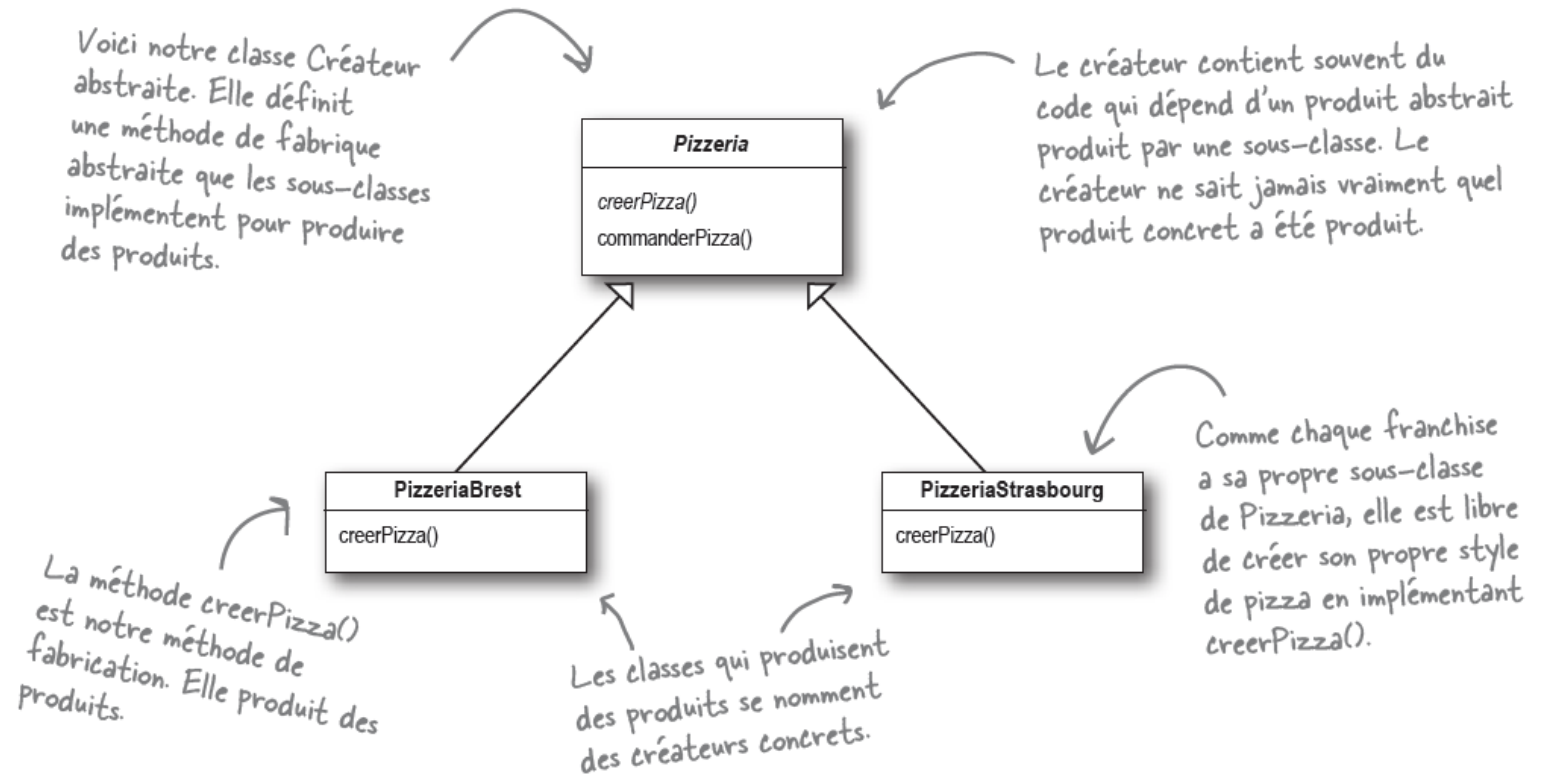
```
public class PizzaTestDrive {  
    public static void main(String[] args) {  
        Pizzeria boutiqueBrest = new PizzeriaBrest();  
        Pizzeria boutiqueStrasbourg = new PizzeriaStrasbourg();  
  
        Pizza pizza = boutiqueBrest.commanderPizza("fromage");  
        System.out.println("Luc a commandé une " + pizza.getNom() + "\n");  
  
        pizza = boutiqueStrasbourg.commanderPizza("fromage");  
        System.out.println("Michel a commandé une " + pizza.getNom() + "\n");  
    }  
}
```

D'abord, nous créons deux boutiques différentes.

Puis nous en utilisons une pour la commande de Luc.

Le pattern Fabrication

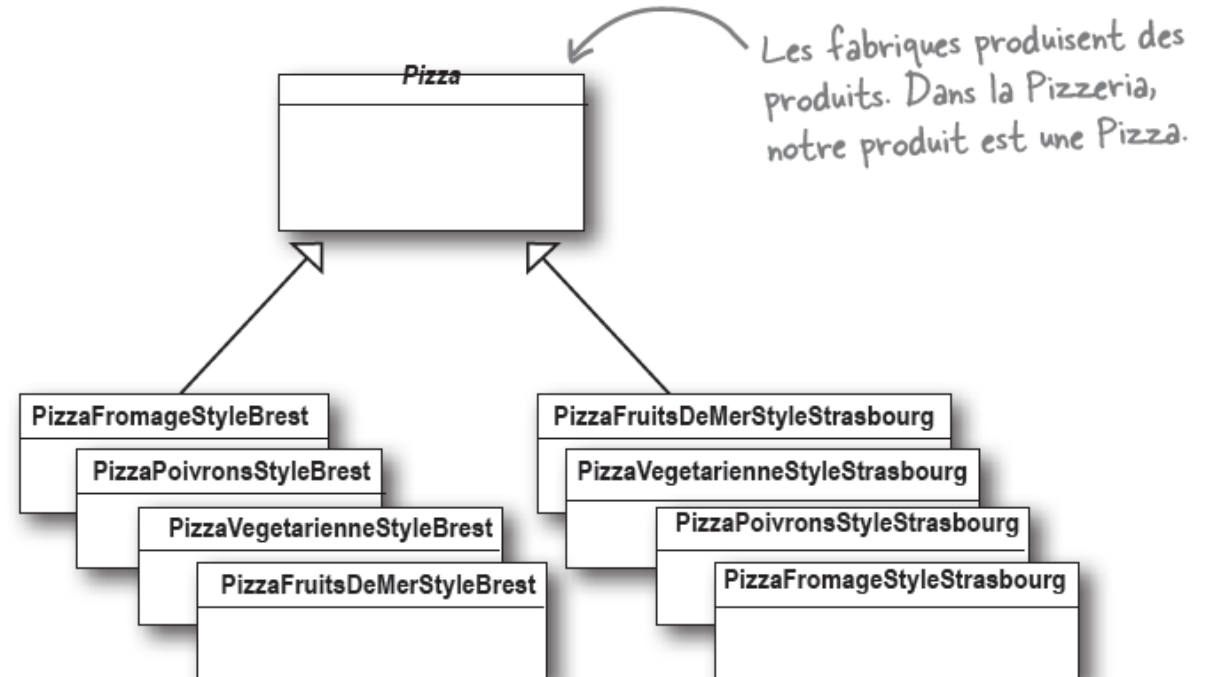
Les classes Créateur (ou Facteur)



Le pattern Fabrication

Les classes Produit

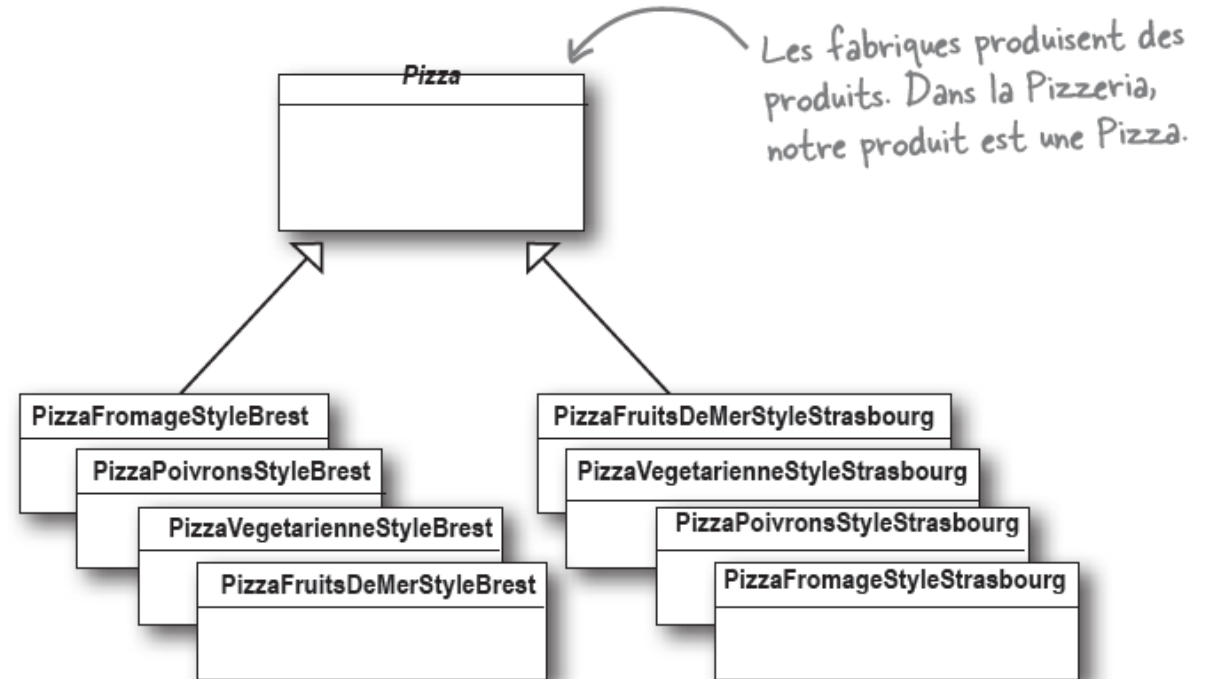
Voici les produits concrets
– toutes les pizzas qui sont
produites par nos boutiques.



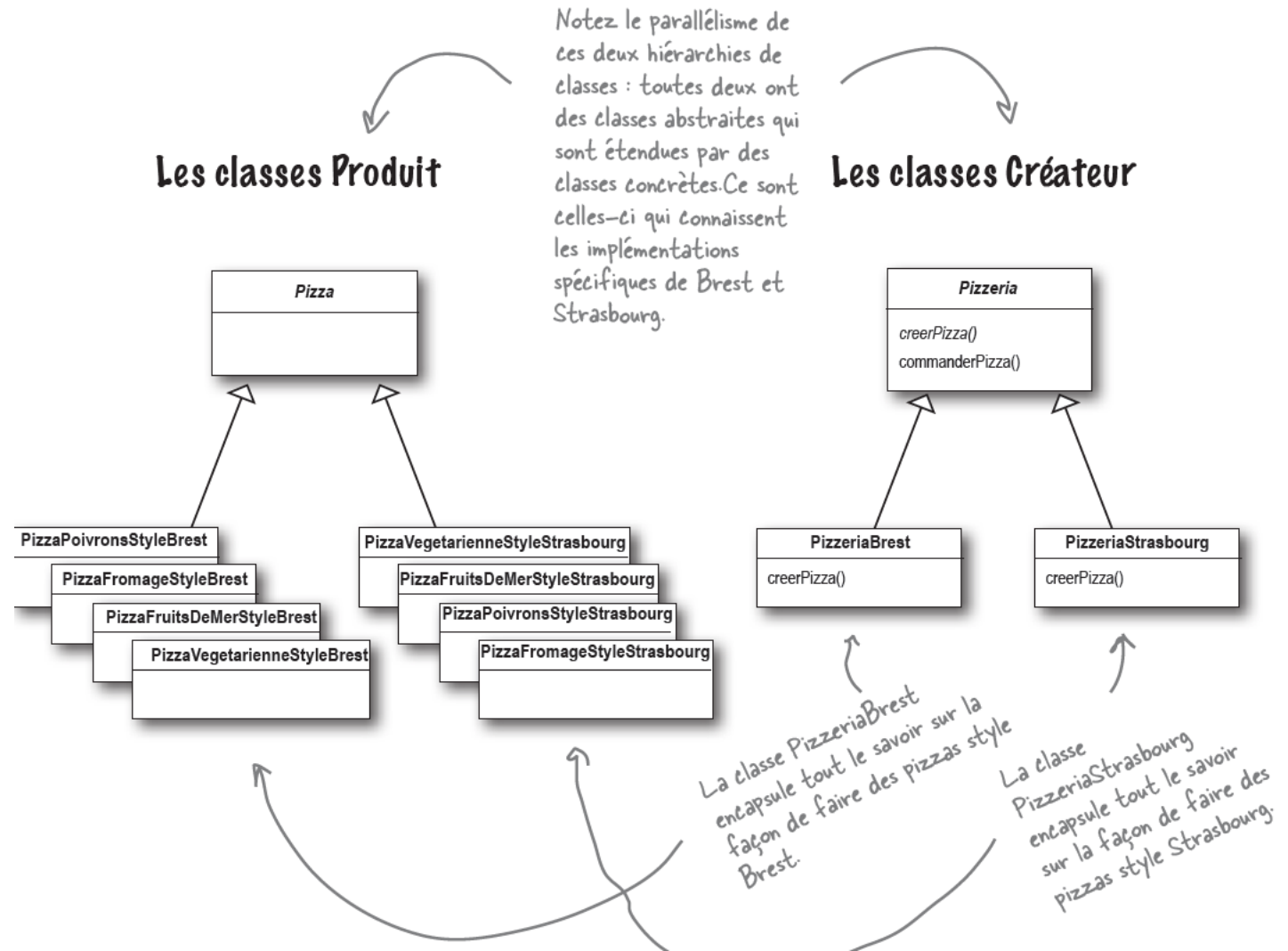
Le pattern Fabrication

Les classes Produit

Voici les produits concrets
– toutes les pizzas qui sont
produites par nos boutiques.



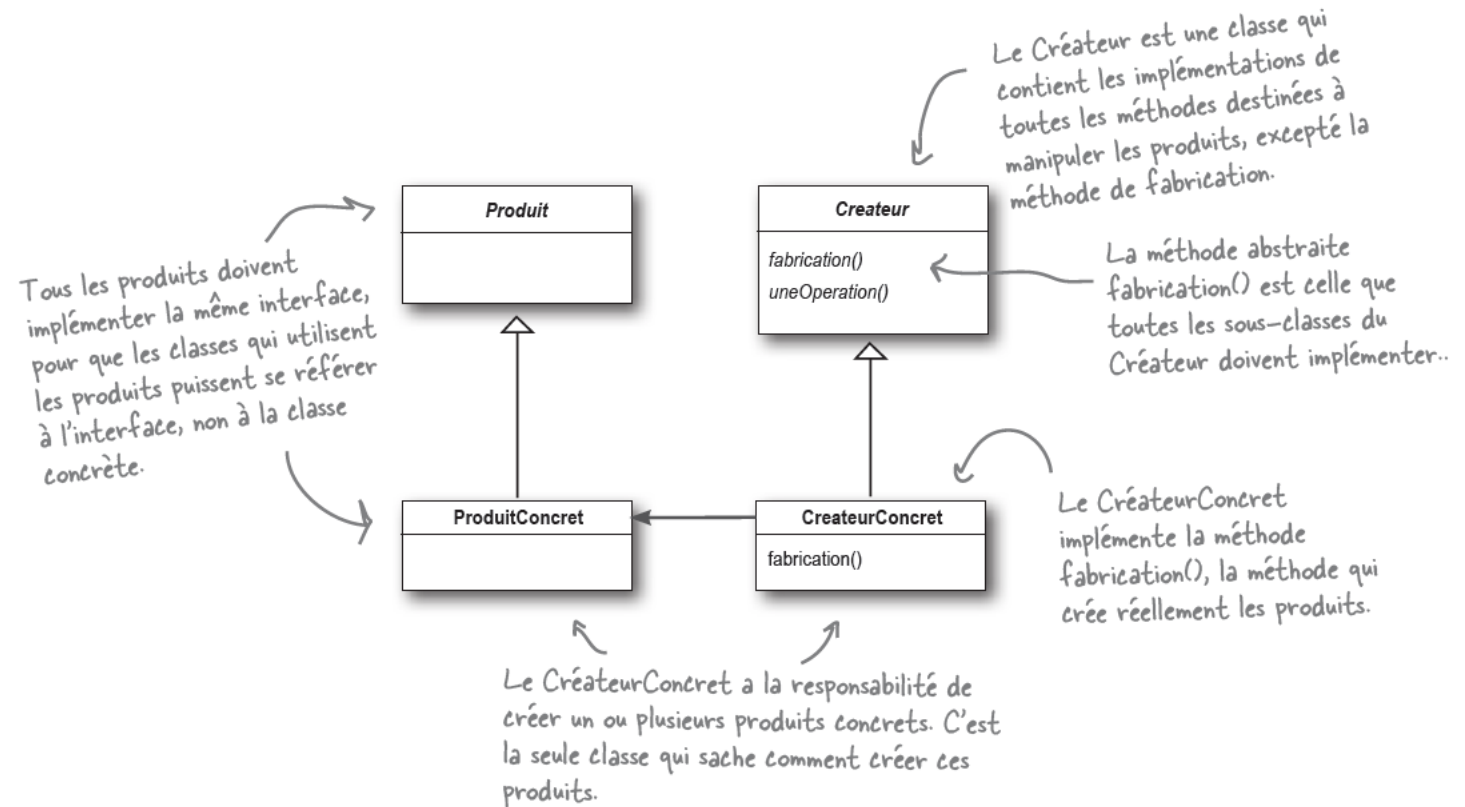
Le Pattern Fabrication



Le Pattern Fabrication : définition

- **Le pattern Fabrication** définit une interface pour la création d'un objet, mais en laissant aux sous-classes le choix des classes à instancier. Fabrication permet à une classe de déléguer l'instanciation à des sous-classes.

Le Pattern Fabrication



Question

Quelle est la différence entre Fabrique Simple et Fabrication. Ils ont l'air très similaires, excepté que, dans Fabrication, la classe qui retourne la pizza est une sous-classe. Pouvez-vous expliquer ?