CPOA - TD

Code initial pour le TD3

Rappel du cours : 📱📱💻 http://bit.ly/jmb-cpoa

Informations générales

NOM

BRUEL

Prénom

Jean-Michel

Groupe #

- [x] Enseignants
- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] Innopolis

Pré-requis

Il vous faut :

- [x] Un compte GitHub
- [ ] Un terminal de type Git Bash (si vous utilisez Window$)

Essayez la commande suivante dans votre terminal pour vérifier votre environnement `git` :

```
git config --global -l
```

Tâche initiale

- [x] Clickez sur le lien Github Classroom fourni par votre enseignant (en fait c'est déjà fait si vous lisez ces lignes).
- [ ] Clonez sur votre machine le projet Github généré pour vous par Github Classroom.

- ☐ Modifez le `README` pour modifier Nom, Prénom et Groupe.
- ☐ Commit & push:

commit/push

```
fix #0 Initial task done
```

Dans la suite de ce document, à chaque fois que vous trouverez un énoncé commençant par `fix #···` vous devez vérifier que vos scripts/fichiers modifiés sont bien dans votre dépôt local en vue de committer et de pusher les modifications sur votre dépôt distant en utilisant comme message de commit cet énoncé.

- Si vous voulez vérifier que vous êtes prêt pour le `fix #0`, utilisez la commande : `make check`.
- Si vous voulez avoir la liste des ToDos de ce TP/TP, exécutez `make todos`.

Les exercices de ce TD sont tirés de l'excellent livre "Tête la première : Design Pattern". Bert Bates, Eric Freeman, Elisabeth Freeman, Kathy Sierra. Editions O'Reilly. 2005.

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…)!

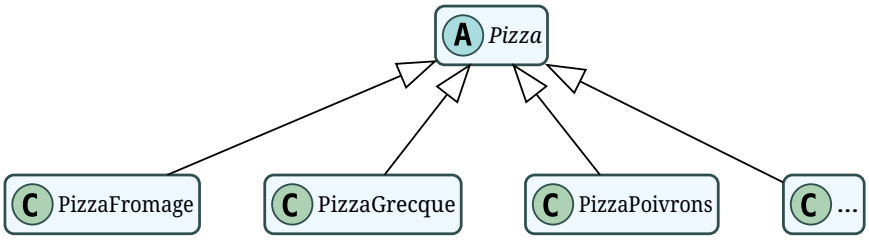
Le stagiaire de l'an dernier qui avait travaillé sur le code est parti avec la caisse (de Chianti). Vous n'avez à votre disposition que :

- Le code de départ suivant :

`Pizzeria.java` *code initial (source disponible dans* `src`*)*

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

        /**
         * @param type
         * @return a Pizza object according to the type
         */
        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;
        }
}
```

- L'ébauche de diagramme de classe des pizzas suivant :



- Le bout de code de test suivant :

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

TODO:

- ☐ 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.).
- ☐ Isolez dans une classe `SimpleFabriqueDePizzas` ce code.

- L'idée est de remplacer le `if` dans ce code par quelque chose comme `pizza = fabrique.creerPizza(type);`
- 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 éventuellement un `enum`.

- ☐ Réalisez le diagramme de classe obtenu.
- ☐ Quel est l'avantage de procéder ainsi ? Ne transfère-t'on pas simplement le problème à un autre objet ?

commit/push

```
fix #1.1 Simple Pizza Factory
```

2. On y est presque...

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.

commit/push

```
fix #2.1 Use of abstract factories
```

Nous sommes arrivés à une situation propre, qui s'apparente à un patron de conception.

Souvent appelé *Simple Factory*, ou *Factory Method*, il n'est pas complètement considéré comme un patron :

(almost) Design pattern : **Simple Fabrique**

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).

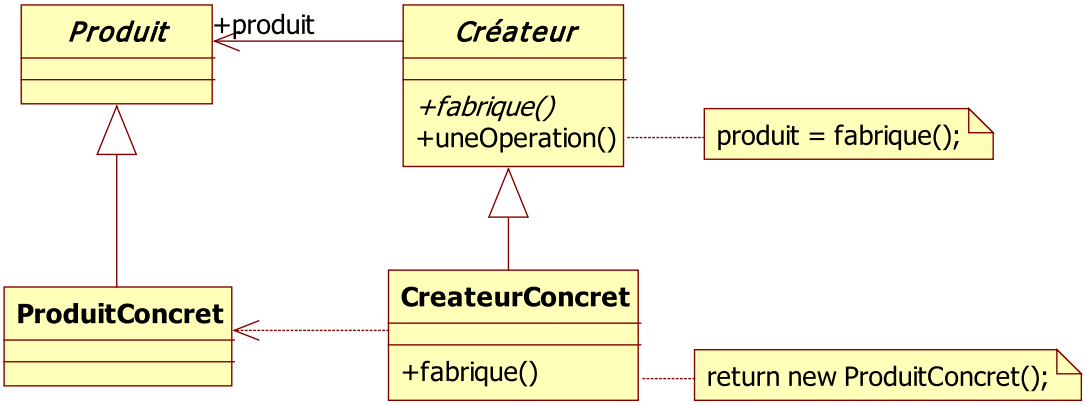


Figure 1. Modèle UML du patron Fabrique

Mais avant d'en arriver à la définition du patron lui-même, nous allons améliorer un peu les choses.

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);
}
```

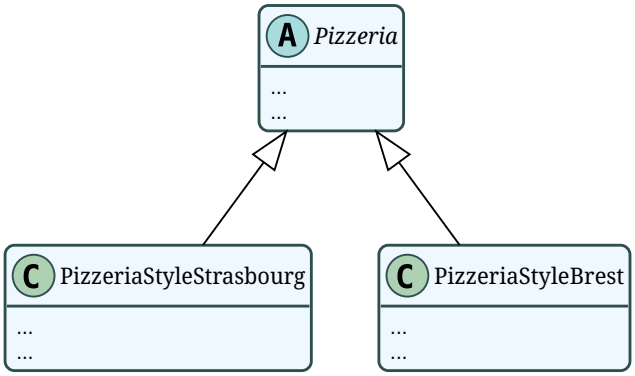
QUESTION

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".



TIP

Chaque sous-classe redéfinit la méthode `creerPizza()`, tandis que toutes les sous-classes utilisent la méthode `commanderPizza()` définie dans `Pizzeria`.

commit/push

```
fix #2.3 Simple Pizza Factory
```

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.

commit/push

```
fix #2.5 Simple Pizza Factory
```

3. Le patron Fabrique

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

Design pattern : **Fabrique**

Fabrique (abstraite) fournit une interface pour la création de familles d'objets apparentés ou interdépendants, sans qu'il soit nécessaire de spécifier leurs classes concrètes.

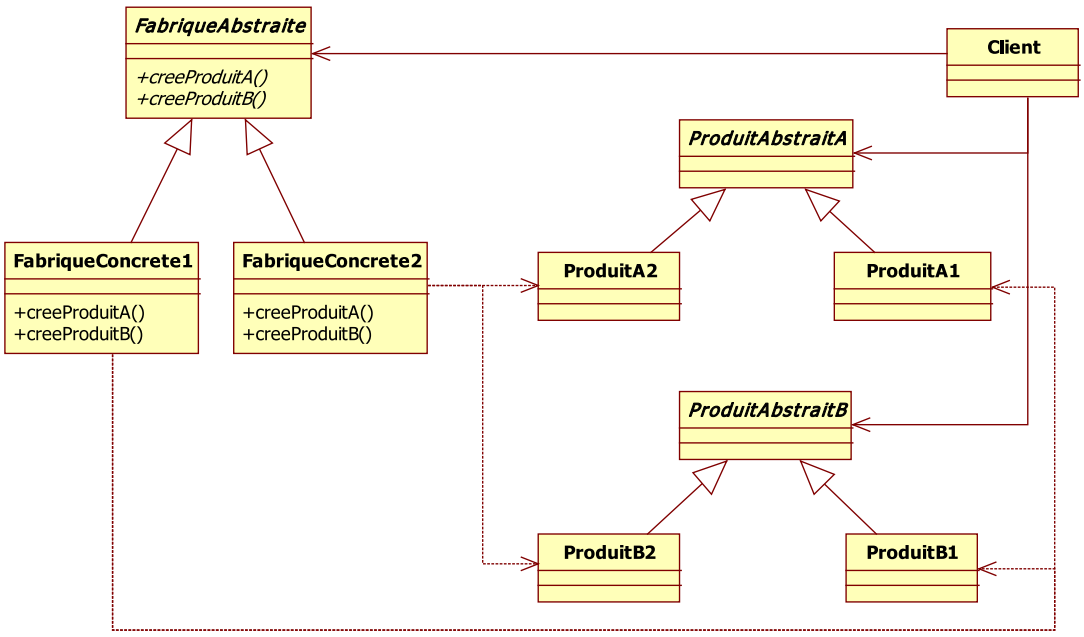


Figure 2. Modèle UML du patron Fabrique Abstraite

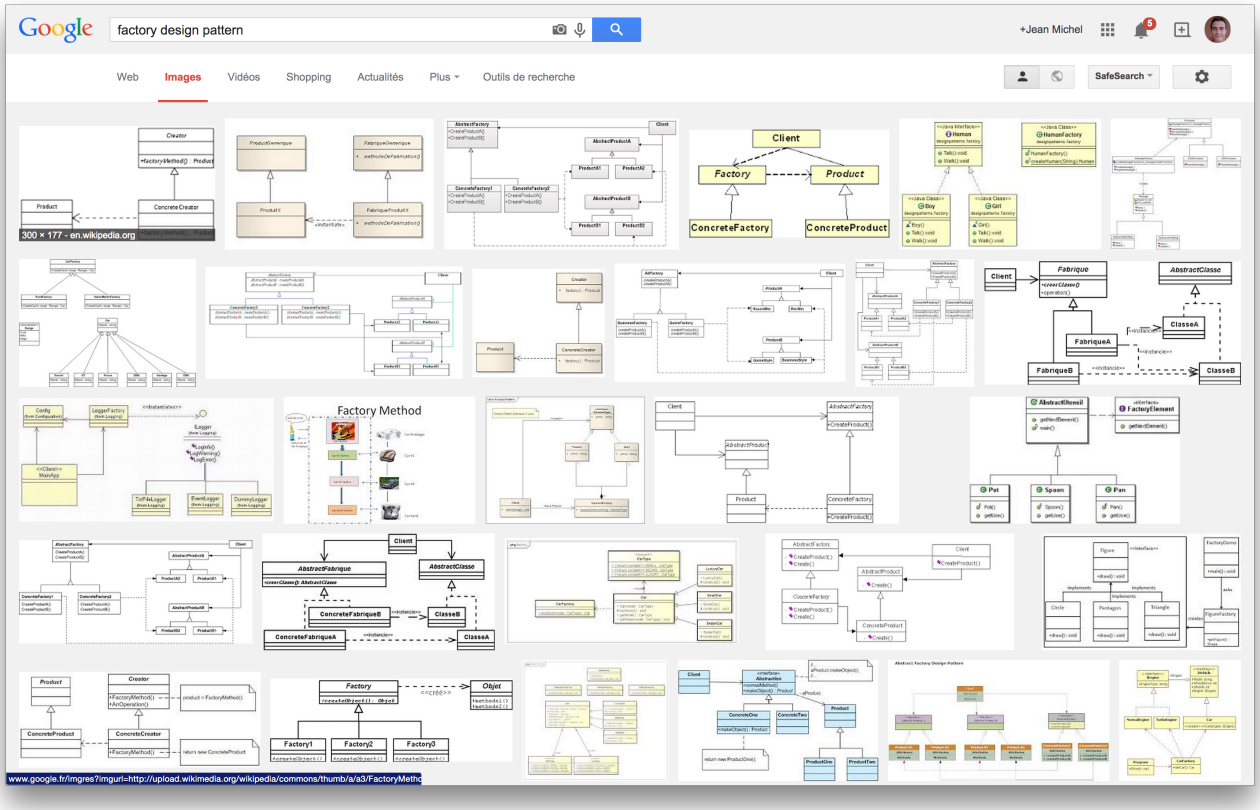


Figure 3. Several examples of Factory pattern definitions

4. Ressources

Mise en place de l'environnement Eclipse:

- Créez un nouveau répertoire (disons `TD3`)
- Dans ce dossier ouvrez un git bash
- git clone http://urlduprojet
- Ouvrez eclipse, changez de workspace, sélectionnez le dossier `TD3`
- Une fois le workspace ouvert, cliquez sur **File › Import › General › Projects from Folder or Achive** puis `Next`
- **Select root directory › Browse**, sélectionnez ce repo (`urlduprojet` dans cet example)
- Cliquez sur **Finish**

Installation de Maven sous Windows:

- Télécharger: https://apache.mediamirrors.org/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.zip
- Décompresser le contenu dans `C:\Programmes\Java`
- Dans Menu Démarrer, taper `Environnement` puis cliquer sur **Modifier les variables d'environnement système**
- Cliquer sur `Variables d environnement`
- Dans la section `Variables système`, sélectionner `Path`, cliquer sur `Modifier`
- Cliquez sur `Nouveau`, puis saisissez `C:\Program Files\Java\apache-maven-3.6.3\bin` (vérifiez que le dossier existe)
- Cliquez successivement 3 fois sur "OK".
- La commande MVM devrait être disponible dans un cmd windows (fermer les existants)

5. Traductions

Table 1. Translations French/English/Russian of the code vocabulary

FR	US	RU
Grecque	Greek	??
Fromage	Cheese	??
Poivron	Pepper	??
preparer	prepare	??
cuire	bake	??
couper	cut	??
emballer	wrap	??

FR	US	RU
creer	create	??
commander	order	??

6. Contributeurs

- Jean-Michel Bruel

7. À propos...