



Le génie pour l'industrie

LOG121

Conception orientée objet

Patron Méthode Fabrique

Enseignante: Souad Hadjres

Retour sur les itérateurs

2

- L'interface Collection de Java définit une méthode

```
Iterator iterator();
```

- Les classes concrètes de collection (ArrayList, LinkedList, etc.) implémentent cette méthode
 - ▣ Chaque classe l'implémente de façon différente pour retourner un itérateur adéquat selon le type de la collection

```
Collection myList = new LinkedList();  
Iterator iter = myList.iterator();
```

Retour sur les itérateurs

3

- Pourquoi ne pas utiliser les constructeurs des itérateurs tout simplement comme suit?

```
Collection myList = new LinkedList();  
Iterator iter = new LinkedListIterator(myList);
```

- ▣ Nous avons besoin de connaître quelle collection nous manipulons et quel est l'itérateur associé.

```
Collection coll = ???  
Iterator iter = new ???(coll);
```

Retour sur les itérateurs

4

- La méthode `iterator()`, appelée *méthode fabrique*, évite ce problème
 - ▣ La méthode `iterator()` crée un objet itérateur implémentant l'interface `Iterator`
 - ▣ Nous n'avons pas besoin de connaître quel sous-type de l'interface `Iterator` on doit construire
 - ▣ Grâce au polymorphisme, peu importe le type concret de la collection, je peux accéder à son itérateur comme suit

```
Collection myList = ...
Iterator iter = myList.iterator()
```
- Une méthode fabrique est plus flexible qu'un constructeur
 - ▣ On peut construire des objets des sous-classes

Un autre exemple

5

- Dans une compagnie de fabrication d'ustensiles en métal
 - ▣ Pour chaque ustensile, il y a un moule
 - ▣ On injecte le métal fondu dans le moule pour fabriquer l'ustensile
- Dans une application qui gère ce processus de fabrication
 - ▣ Une classe cliente peut manipuler directement les moules et les ustensiles

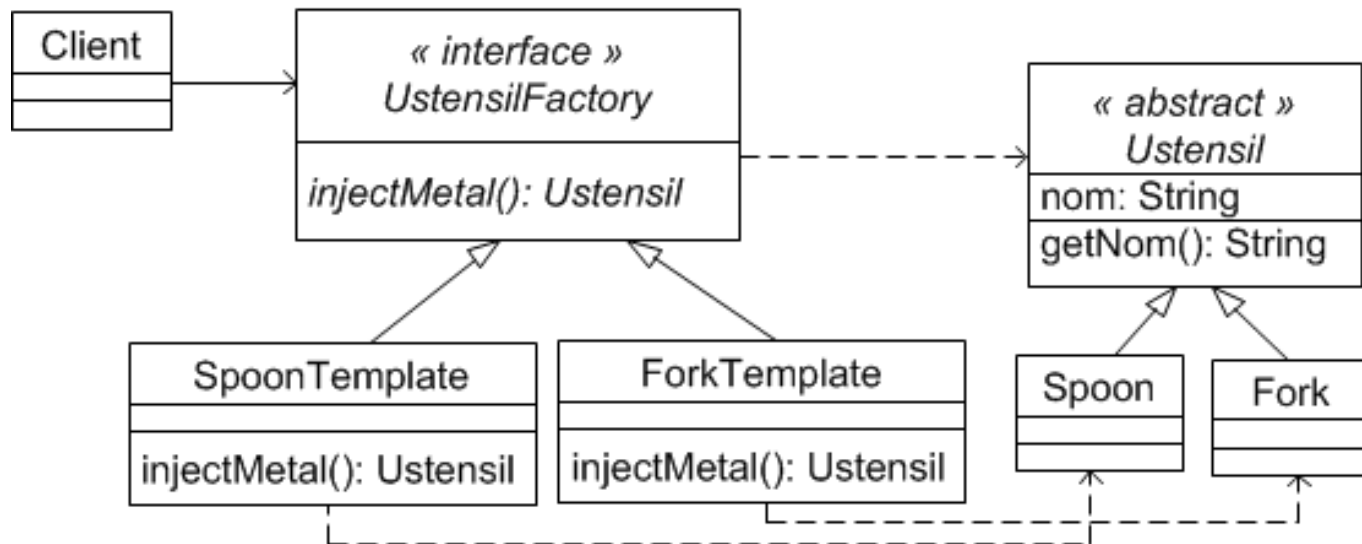
```
SpoonTemplate mySpoonTemplate = new SpoonTemplate();  
Spoon mySpoon = new Spoon(mySpoonTemplate);
```

- ▣ La classe cliente a besoin de connaître quel moule elle manipule et quel ustensile lui est associé
 - ▣ Plus de « if... else » dans le code
 - ▣ Plus de couplage pour la classe cliente qui doit connaître tous les moules et tous les ustensiles

Un autre exemple

6

- Une meilleure solution:
 - ▣ Définir une interface commune pour la fabrication d'ustensiles
 - ▣ Chaque moule implémente cette interface
 - ▣ Chaque moule sait quel ustensile il produit



- ▣ **injectMetal()** est une méthode fabrique

```
UstensilFactory myFactory = ...
```

```
Ustensil myUstensil = myFactory.injectMetal();
```

Le patron Méthode Fabrique

7

□ Contexte

- ▣ Un objet d'une classe (Creator) crée des objets d'une autre classe (Product)
- ▣ Les sous-classes de Creator créent différents types d'objets
- ▣ Une classe cliente n'a pas besoin de connaître le type des objets produits

Le patron Méthode Fabrique

8

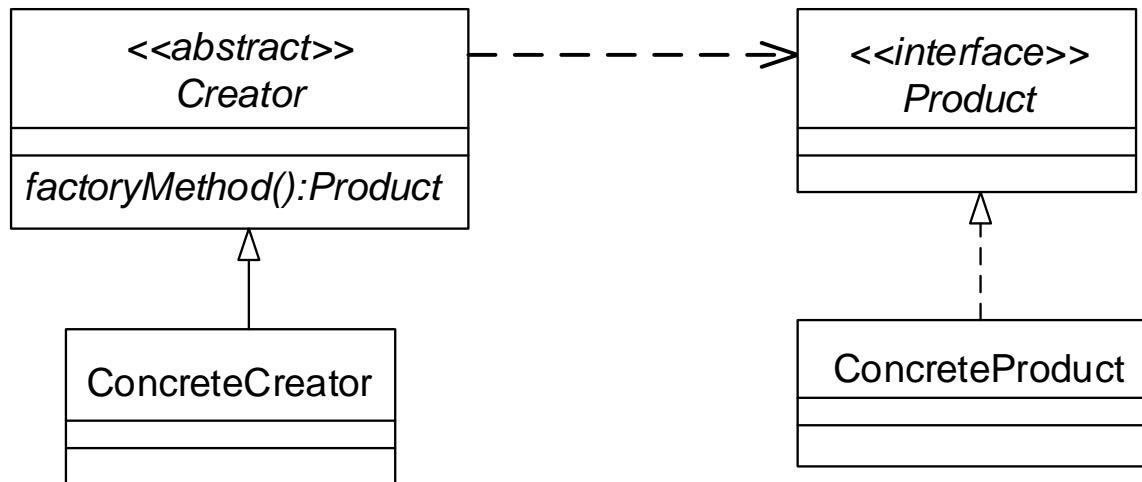
□ Solution

- ▣ Définir un type Creator qui regroupe les parties communes à toutes les classes qui créent
- ▣ Définir un type Product qui regroupe les parties communes à toutes les classes à créer
- ▣ Définir une méthode, appelée la *méthode fabrique*, dans le type Creator; cette méthode retourne une instance d'un objet de type Product
- ▣ Chaque classe concrète de création (ConcreteCreator) implémente la méthode fabrique pour retourner un objet d'une classe concrète à produire (ConcreteProduct)

Le patron Méthode Fabrique

9

□ La structure du patron



Nom dans le patron de conception

Creator

ConcreteCreator

FactoryMethod()

Product

ConcretProduct

Nom dans l'exemple avec les itérateurs

Collection

Une classe qui implémente Collection
iterator()

Iterator

Une classe qui implémente Iterator

Le patron Méthode Fabrique

10

- Attention! Les méthodes qui créent de nouveaux objets ne sont pas toutes des méthodes fabriques.

- Par exemple:

```
DateFormat formatter = DateFormat.getDateInstance();  
Date now = new Date();  
String formattedDate = formatter.format(now);
```

- ▣ `getDateInstance` n'est pas une méthode fabrique
 - ▣ C'est une méthode statique qui retourne le formatteur par défaut `SimpleDateFormat`
 - ▣ Il n'y a pas de création polymorphique