ISIMA
Institut Supérieur d'Informatique de
Modélisation et de leurs Applications

24, Avenue des Landais
BP 10 125
63 173 AUBIERE cedex.

Compte-rendu

Java et objet avancé

Filière 3 : " Systèmes d'Information et Aide à la décision "

Design Patterns & JEE

Tome 2

Auteur : **Mathieu BRUNOT**

Responsable ISIMA : Guillaume DEMONSABLON

Date : 13/02/2012

Titre du document :	Type du document
JAVA ET OBJET AVANCÉ Design Patterns & JEE	Compte-rendu
	Date du document : 13/02/2012
Origine du document : Mathieu BRUNOT / ISIMA	Pagination : 13 pages

Objet du document

Ce document présente la seconde partie du rapport dans le cadre du cours de Java et objet avancé.

Sommaire

Introduction

La seconde partie du rapport de Java et objet avancé se concentre sur les *Design Patterns* du GOF (*Gang Of Four*) et le fonctionnement de Java EE.

Dans un premier temps, on présentera les *Design Patterns* suivants :

1. Stratégie (*Strategy*) : *Design Patterns* de comportement ;
2. Fabrique (Factory Method) : *Design Patterns* de création ;
3. Stratégie & Fabrique : association des *Design Patterns* précédant ;
4. Observateur (*Observer*) : *Design Patterns* de comportement.

Chacun des *Design Patterns* présentera le principe et l'implémentation réalisée en TP.

Par la suite, on présentera les intéractions entre les différentes couches de Java EE : BDD, JDBC, EJB entité, ...

I. Design Patterns

A. Stratégie (Strategy)

Le but de ce *Design Pattern* de comportement est de rendre interchangeables des algorithmes d'une même famille

Exemple :

- *Parcours d'un arbre :*
  - *Parcours en profondeur ;*
  - *Parcours en largeur*

Cette interchangeabilité est obtenue via l'utilisation d'un classe abstraite définissant un algorithme générique. Les algorithmes spécifiques dérivent de l'algorithme générique. De cette façon, le contexte s'éxécute indépendamment de l'algorithme utilisé.

Diagramme de classes de l'implémentation du *Design Pattern* Stratégie :

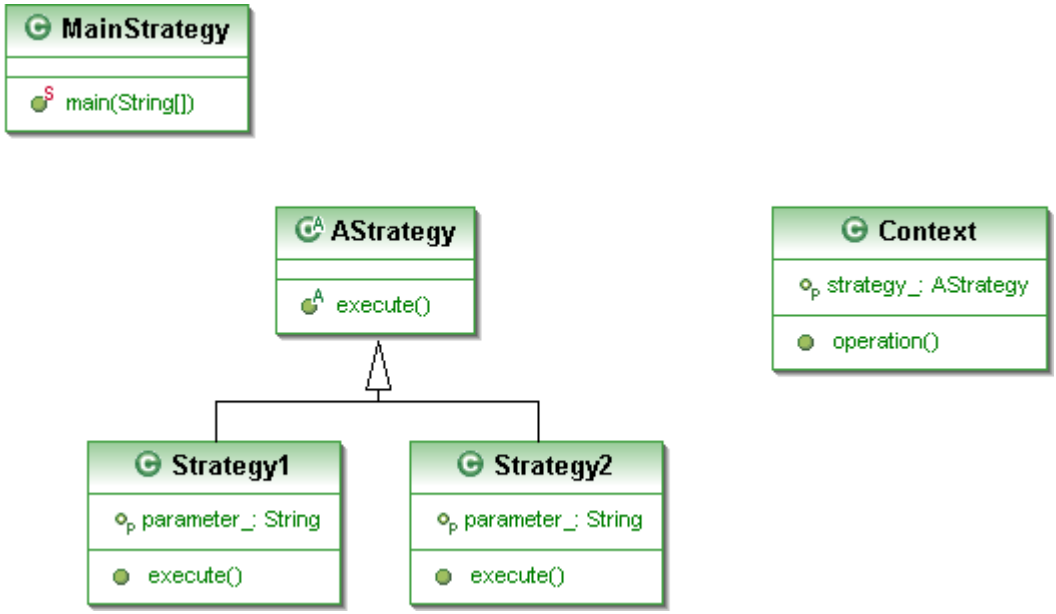

Illustration 1: Stratégie (Strategy) : Design Patterns de Comportement

Dans l'implémentation du *Design Pattern* Stratégie effectuée en TP, la stratégie a pour simple but d'afficher l'objet courant avec plus ou moins de détails :

- la straégie 1 affiche uniquement le paramètre (quelconque) de la stratégie ;
- la stratégie 2 affiche l'instance et le paramètre de cette dernière.

L'opération du contexte consiste donc simplement à afficher la stratégie courante.

Main :

```
public static void main(String[] args) {
	System.out.println("GOF Design Patterns: Strategy");

	Context context = new Context();
	context.operation();

	context.setStrategy(new Strategy2());
	context.operation();
}
```

Sortie Console :

```
GOF Design Patterns: Strategy
Strategy1
com.isima.zz3.advjava.tp1.strategy.Strategy2@4fe5e2c3:Strategy2
```

B. Fabrique (Factory Method)

Le *Design Pattern* Fabrique a pour objectif de déléguer la création d'un objet.

Exemple :

- *Création d'un document:*
  - *Document XML ;*
  - *Document texte.*

Pour cela, on définit un créateur abstrait qui définit une méthode de fabrication d'une interface de produits. La fabriquation d'un produit particulier peut ensuite être spécifiée par héritage.

Le client ne communique qu'avec un créateur abstrait des interfaces de produits. Seul le créateur instancié connait la classe réelle de l'objet.

Diagramme de classes de l'implémentation du *Design Pattern* Fabrique :

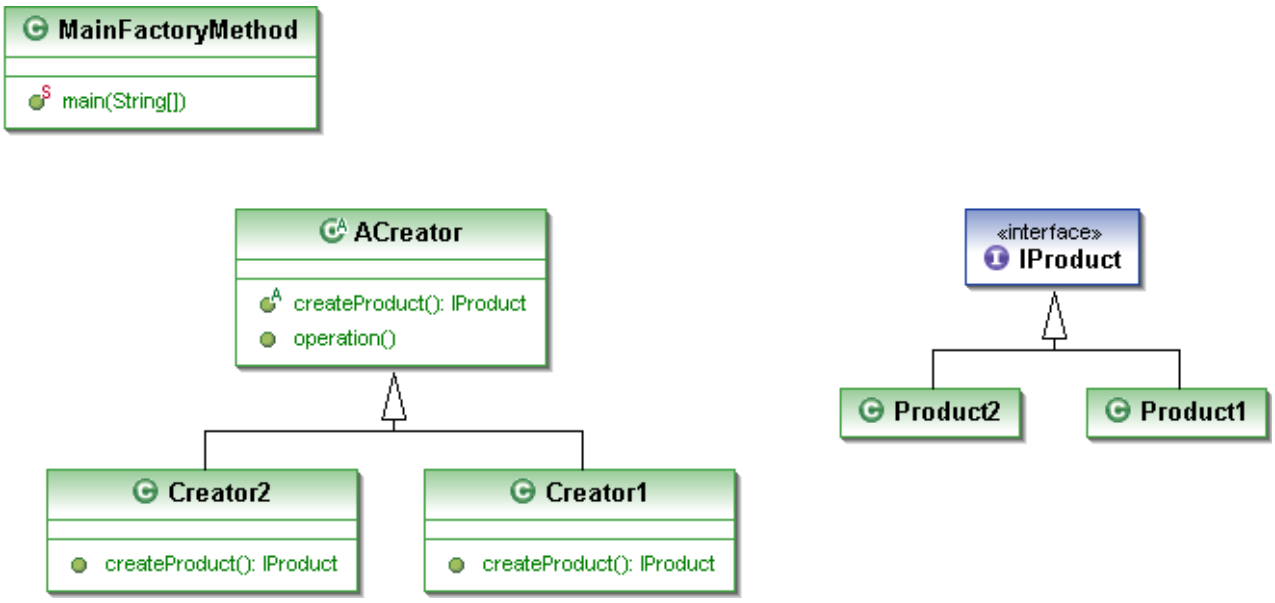

Illustration 2: Fabrique (Factory Method) : Design Pattern de Création

Là encore l'implémentation reste très simple. Chacune des fabriques est chargée de fabriquer un seul et unique produit et l'opération généique d'une fabrique consiste à fabriquer un produit et l'afficher.

Main :

```
public static void main(String[] args) {
	System.out.println("GOF Design Patterns: Factory Method");

	ACreator creator = new Creator1();
	creator.operation();

	creator = new Creator2();
	creator.operation();
}
```

Sortie Console :

```
GOF Design Patterns: Factory Method
com.isima.zz3.advjava.tp1.factorymethod.Product1@7d8a992f
com.isima.zz3.advjava.tp1.factorymethod.Product2@23fc4bec
```

C. Stratégie & Fabrique

Ici, on couple les *Design Patterns* Stragégie et Fabrique vus précédemment.

Exemple :

- *Création d'un document:*
  - *Document XML ;*
  - *Document texte ;*
- *Algorithme de lecture du document (en fonction du type de document créé) :*
  - *Lecture d'un fichier XML ;*
  - *Lecture d'un fichier plat.*

Diagramme de classes de l'implémentation des *Design Patterns* Stratégie & Fabrique :

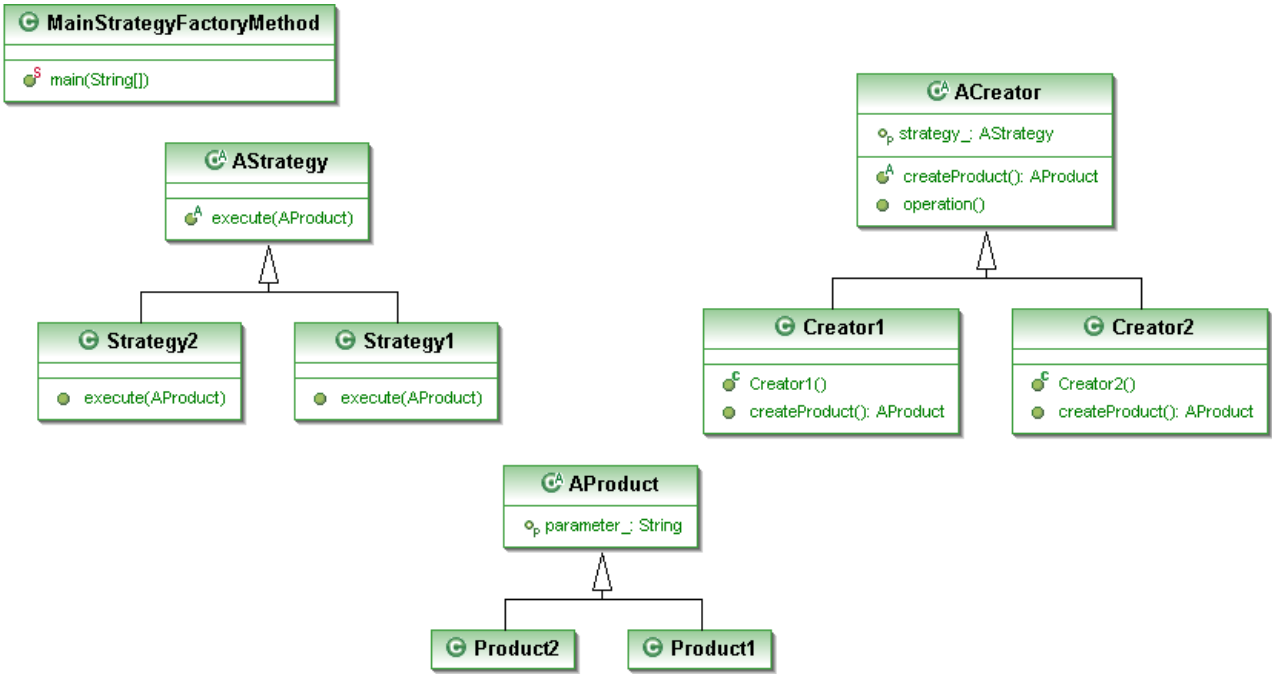

Illustration 3: Stratégie & Fabrique utilisés simultanément

L'aspect Fabrique de cette implémentation est la même que dans l'exemple précédant. La Stratégie intervient dans l'algorithme qu'utilise l'opération de la fabrique :

- Produit 1 → Stratégie 1 : affichage du paramètre du produit ;
- Produit 2 → Stratégie 2 : affichage du produit et de son paramètre.

Main :

```
public static void main(String[] args) {
	System.out.println("GOF Design Patterns: Strategy + Factory Method");

	ACreator creator = new Creator1();
	creator.operation();

	creator = new Creator2();
	creator.operation();
}
```

Sortie Console :

```
GOF Design Patterns: Strategy + Factory Method
This is a product!
com.isima.zz3.advjava.tp1.strategyandfactorymethod.Product2@45bab50a: This is a product!
```

D. Observateur (Observer)

Le *Design Pattern* Observateur a pour objectif de synchroniser plusieurs objets sur l'état d'un autre objet. Lorsque l'objetsujet change d'état, il avertit les observateurs et ceux-ci se mettent à jour en conséquence.

Remarque :

- *Il est implémenté de façon native en Java au travers des Listenner.*

Diagramme de classes de l'implémentation du *Design Pattern* Observateur :

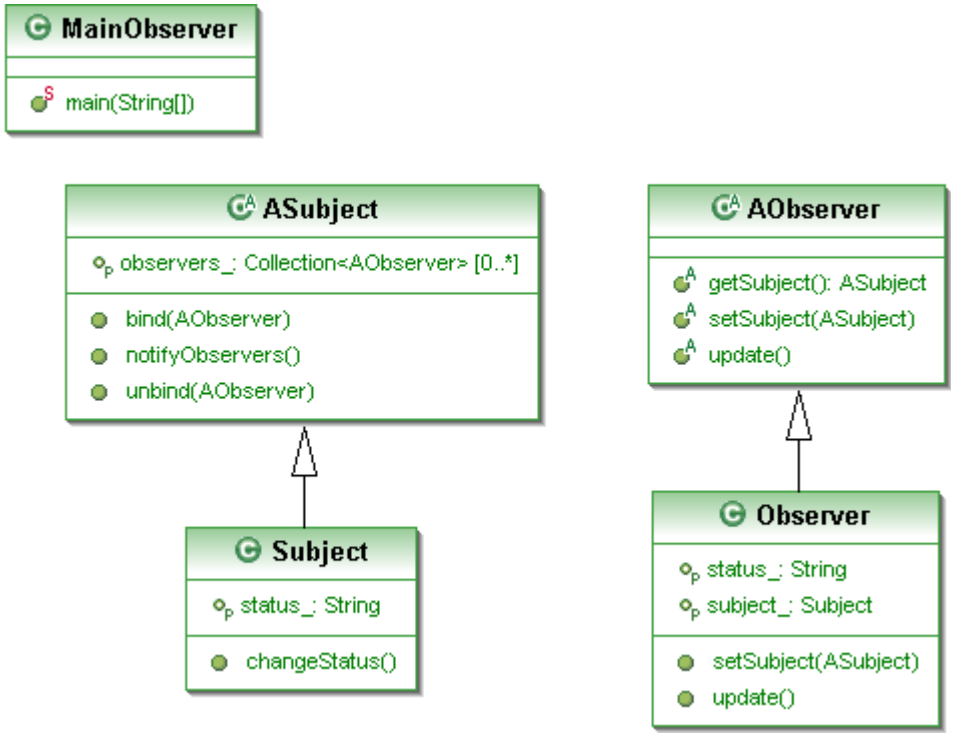

Illustration 4: Observateur (Observer) : Design Patterns de Comportement

L'implémentation du *Design Pattern* Observateur prend la forme d'un système de log des modifications d'un sujet. Le sujet a un paramètre d'état qui a pour valeur initiale `"default"`., tandis que l'observateur à état initial à `null`.

Lors du changement d'état du sujet, l'observateur log le changement du sujet : instance, date & heure et paramètre du sujet.

Main !

```
public static void main(String[] args) {
	System.out.println("GOF Design Patterns: Observer");

	Subject subject = new Subject();
	Observer observer = new Observer();

	subject.bind(observer);

	System.out.println(subject.getStatus());
	System.out.println(observer.getStatus());

	subject.changeStatus();

	System.out.println(subject.getStatus());
	System.out.println(observer.getStatus());

}
```

Sortie Console :

```
GOF Design Patterns: Observer
default
null
changed
com.isima.zz3.advjava.tp1.observer.Subject@6ac2a132 changed at 2012/02/12 00:41:25:
changed
```

II. Java Enterprise Edition (JEE)