Partie 5
Application des patterns

Chapitre 5-1
Compositions et variations de patterns

1. Préliminaire

Les vingt-trois patterns de conception introduits dans cet ouvrage ne constituent bien évidemment pas une liste exhaustive. Il est possible de créer de nouveaux patterns soit ex nihilo, soit en composant ou adaptant des patterns existants. Ces nouveaux patterns peuvent avoir une portée générale, à l'image de ceux introduits dans les chapitres précédents ou être spécifiques à un environnement de développement particulier. Nous pouvons citer ainsi le pattern de conception des JavaBeans ou les design patterns des EJB.

Dans ce chapitre, nous allons vous montrer trois nouveaux patterns obtenus par composition et variation de patterns existants.

2. Le pattern Pluggable Factory

2.1 Introduction

Nous avons introduit dans un précédent chapitre le pattern `Abstract Factory` pour abstraire la création (instanciation) de produits de leurs différentes familles. Une fabrique est alors associée à chaque famille de produits. Sur le diagramme de la figure 5-1.1, deux produits sont exposés : les automobiles et les scooters, décrits chacun par une classe abstraite. Ces produits sont organisés en deux familles : traction essence et traction à l'électricité. Chacune de ces deux familles engendre une sous-classe concrète de chaque classe de produit.

Il existe donc deux fabriques pour les familles `FabriqueVéhicule-Essence` et `FabriqueVéhiculeÉlectricité`. Chaque fabrique permet de créer l'un des deux produits à l'aide des méthodes appropriées.

Ce pattern organise de façon très structurée la création d'objets. Chaque nouvelle famille de produits oblige à ajouter une nouvelle fabrique et donc une nouvelle classe.

À l'opposé, le pattern `Prototype` introduit dans le chapitre du même nom offre la possibilité de créer des nouveaux objets de façon très souple.

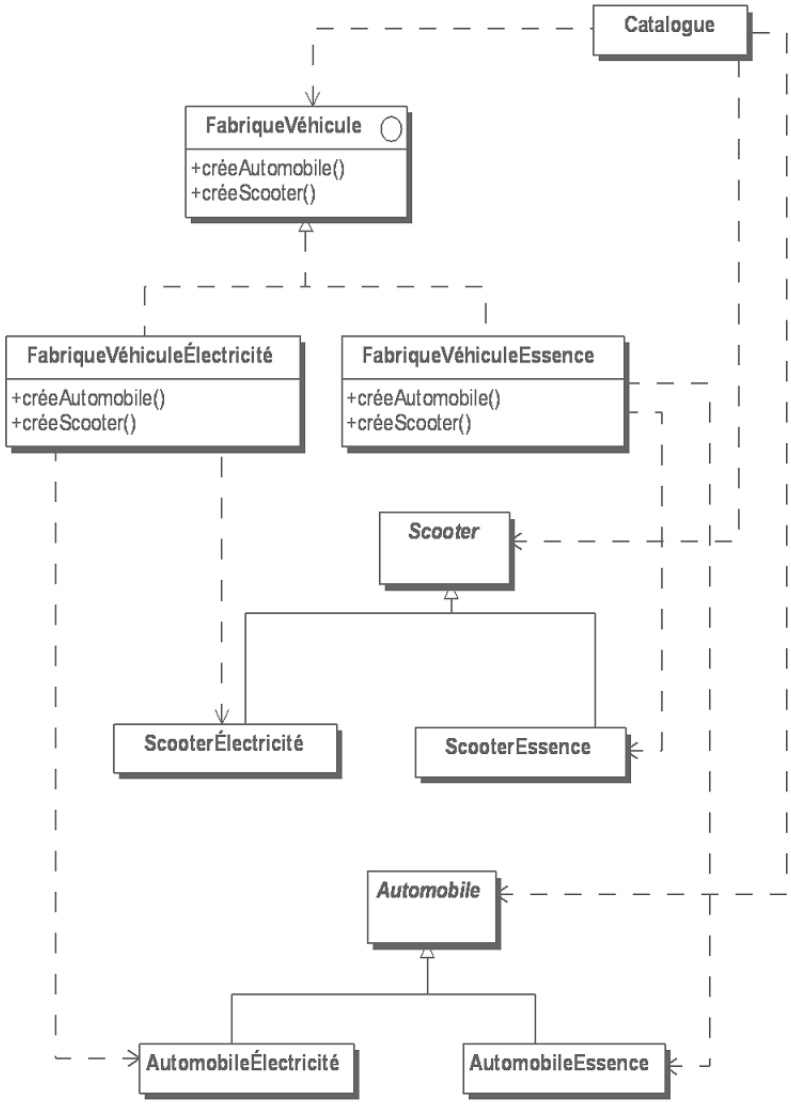

Figure 5-1.1 - Exemple d'utilisation du pattern `Abstract Factory`

La structure du pattern `Prototype` est décrite à la figure 5-1.2. Un objet initialisé afin d'être prêt à l'emploi et détenant la capacité de se dupliquer est appelé un prototype.

Le client dispose d'une liste de prototypes qu'il peut dupliquer lorsqu'il le désire. Cette liste est construite dynamiquement et peut être modifiée à tout moment lors de l'exécution. Le client peut ainsi construire de nouveaux objets sans connaître la hiérarchie de classes dont ils proviennent.

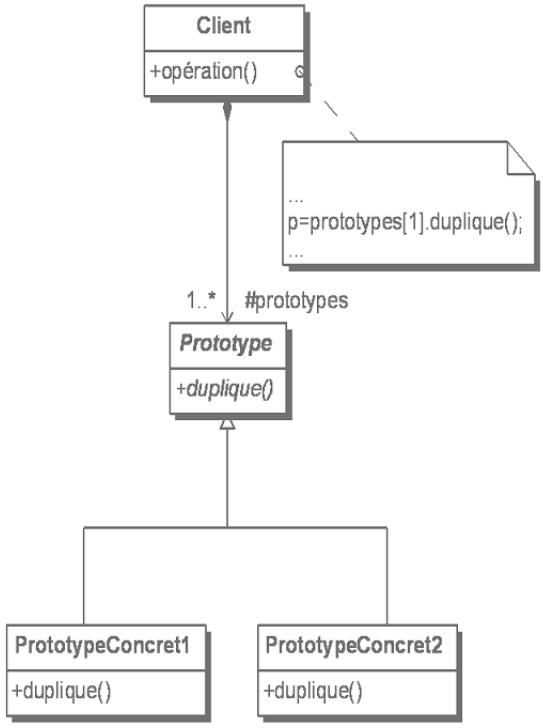

Figure 5-1.2 - Exemple d'utilisation du pattern `Prototype`

L'idée du pattern `Pluggable Factory` est de composer ces deux patterns pour conserver l'idée de création d'un produit par l'invocation d'une méthode de la fabrique et d'autre part, de pouvoir changer dynamiquement la famille à créer. Ainsi, la fabrique ne doit plus connaître les familles d'objets, le nombre de familles peut être différent pour chaque produit et enfin, il est possible de faire varier le produit à créer non plus uniquement par sa sous-classe (sa famille) mais aussi par des valeurs différentes de certains attributs. Nous reviendrons sur ce dernier point dans l'exemple Java.

La figure 5-1.3 reprend l'exemple du chapitre Le pattern Abstract Factory, structuré cette fois à l'aide du pattern `Pluggable Factory`. La classe de fabriques d'objets `FabriqueVéhicule` n'est plus une interface comme à la figure 5-1.1 mais une classe concrète qui permet la création des objets et qui n'a donc plus besoin de sous-classes. Chaque fabrique possède un lien vers un prototype de chaque produit. De façon plus précise, il s'agit d'un lien vers une instance de l'une des sous-classes de la classe `Automobile` et d'un lien vers une instance de l'une des sous-classes de la classe `Scooter`.

C'est ici qu'intervient le pattern `Prototype`. Chaque produit devient un prototype. La classe abstraite qui introduit et décrit chaque famille de produits leur confère la capacité de clonage. Elle joue ainsi le rôle de la classe abstraite `Prototype` de la figure 5-1.2.

Les deux liens présents dans `FabriqueVéhicule` vers chaque prototype peuvent être modifiés dynamiquement à l'aide des méthodes `setPrototype Automobile` et `setPrototypeScooter`. La fabrique nécessite d'ailleurs que ces liens soient initialisés soit à l'aide de ces deux méthodes, soit par un autre moyen (comme le constructeur de la classe) pour pouvoir fonctionner. Le fonctionnement de la fabrique est réalisé par les deux méthodes `créeAutomobile` et `créeScooter` qui s'appuient sur la capacité de clonage des deux prototypes.

La classe `Test` représente le client de la fabrique et des classes de produits. Nous verrons son rôle dans l'exemple Java.

Remarque

Le nom du pattern provient d'une part du fonctionnement de la fabrique de produits qui est dépendant des prototypes fournis et d'autre part de la possibilité de changer (« plugger ») dynamiquement ces prototypes.

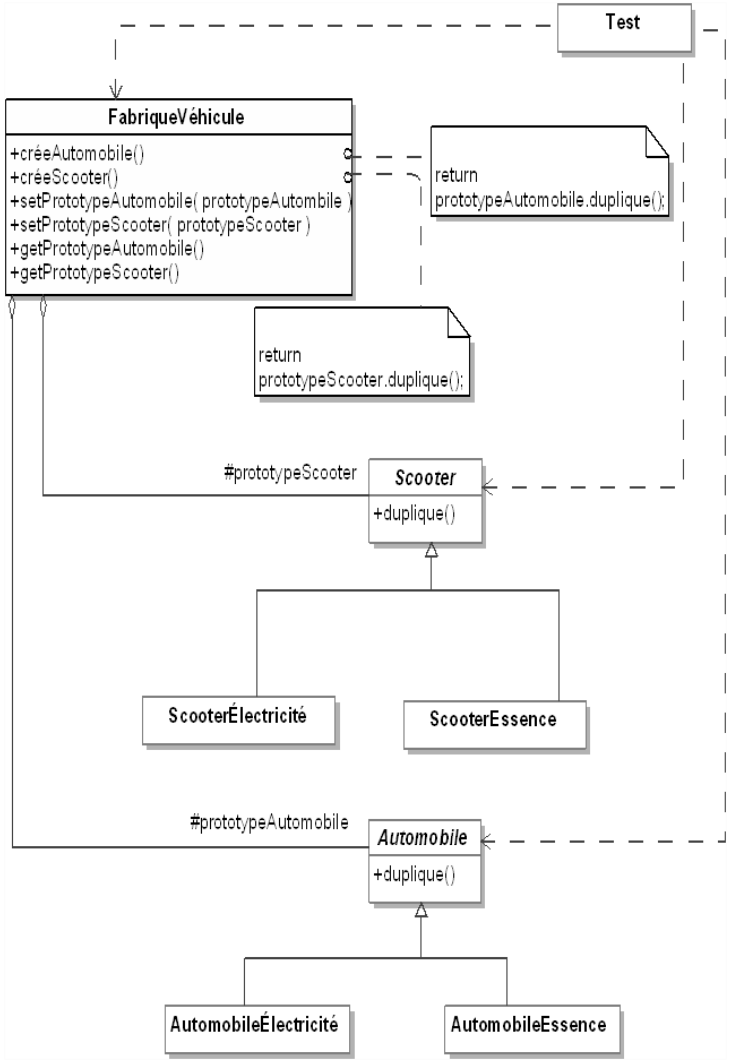

Figure 5-1.3 - Exemple d'utilisation du pattern `Pluggable Factory`

2.2 Structure

La figure 5-1.4 illustre la structure générique du pattern `Pluggable Factory`.

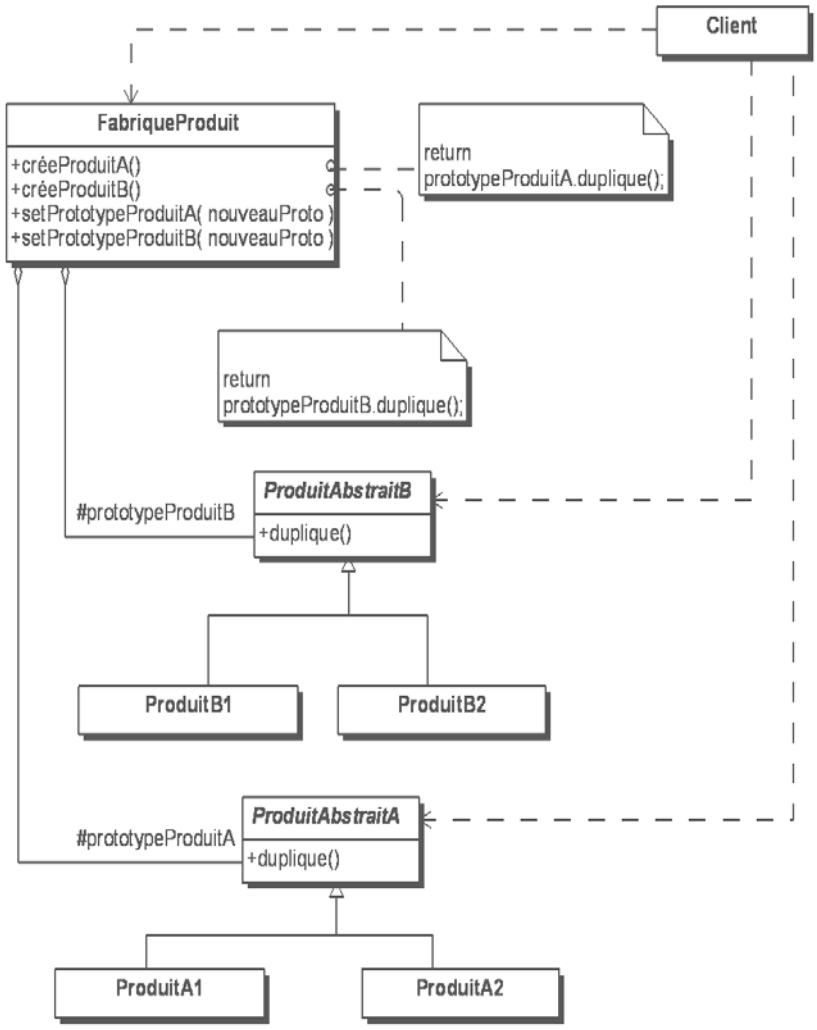

Figure 5-1.4 - Structure du pattern `Pluggable Factory`