Chapitre 4
Le design pattern Abstract Factory

1. Description

Le but du design pattern `Abstract Factory` est la création d'objets regroupés en familles sans avoir à connaître leurs classes concrètes.

2. Exemple

Le système de vente de véhicules gère des véhicules fonctionnant à l'essence et des véhicules fonctionnant à l'électricité. Cette gestion est confiée à l'objet `Catalogue`, à qui incombe la responsabilité de créer de tels objets.

Pour chaque produit, nous disposons d'une classe abstraite, d'une sous-classe concrète décrivant la version du produit fonctionnant à l'essence et d'une sous-classe concrète décrivant la version du produit fonctionnant à l'électricité. Par exemple, à la figure 4.1, pour l'objet scooter, il existe une classe abstraite `Scooter` et deux sous-classes concrètes : `ScooterElectricite` et `ScooterEssence`.

L'objet `Catalogue` peut utiliser ces sous-classes concrètes pour instancier les produits. Cependant, si par la suite de nouvelles familles de véhicules doivent être prises en compte (diesel ou hybride essence-électricité), les modifications à apporter à l'objet `Catalogue` peuvent s'avérer assez fastidieuses.

Le design pattern `Abstract Factory` résout ce problème en introduisant une interface `FabriqueVehiculeInterface` qui contient la signature des méthodes à utiliser pour créer chaque produit. Le type de retour de ces méthodes est constitué par l'une des classes abstraites de produit. Ainsi, l'objet `Catalogue` n'a pas besoin de connaître les sous-classes concrètes et reste parfaitement indépendant des familles de produits. En révélant l'interface de nos fabriques et non leur implémentation, nous découplons le code client des produits concrets : notre objet client `Catalogue` demande des produits à la fabrique qu'on lui passe en paramètre lors de sa construction sans avoir la moindre idée de qui elle est ni de ce qui se passe en coulisses pour qu'il obtienne le bon produit.

Une classe implémentant `FabriqueVehiculeInterface` est créée pour chaque famille de produits, à savoir les classes `FabriqueVehiculeElectricite` et `FabriqueVehiculeEssence`. Une telle classe a la responsabilité d'implémenter les opérations de création du véhicule appropriée pour la famille à laquelle elle est associée.

L'objet client `Catalogue` prend alors pour paramètre un objet se conformant à l'interface `FabriqueVehiculeInterface`, c'est-à-dire soit une instance de `FabriqueVehiculeElectricite`, soit une instance de `FabriqueVehiculeEssence`. Avec une telle instance, le catalogue peut créer et manipuler des véhicules sans devoir connaître les familles de véhicules et les classes concrètes correspondantes.

L'ensemble des classes du design pattern `Abstract Factory` pour cet exemple est détaillé à la figure 4.1.

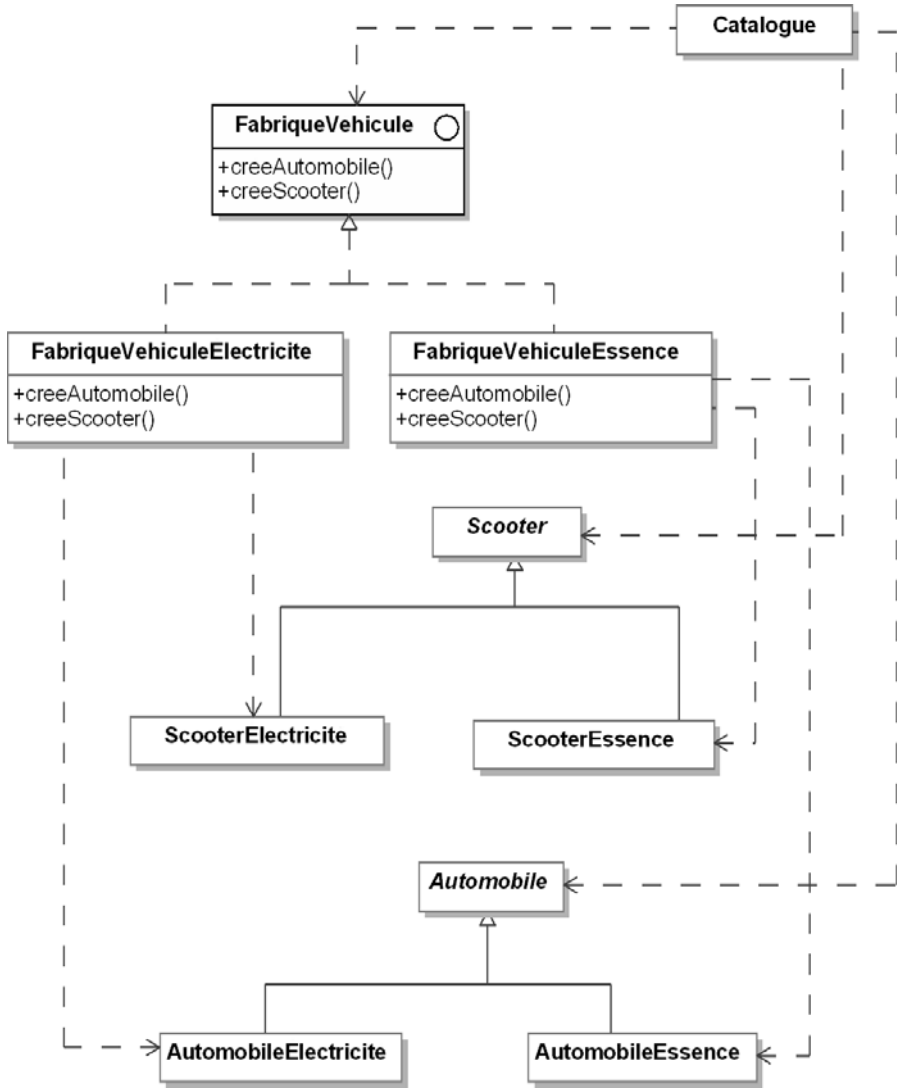

Figure 4.1 - Le design pattern `Abstract Factory` appliqué à des familles de véhicules

3. Structure

3.1 Diagramme de classes

La figure 4.2 détaille la structure générique du design pattern `Abstract Factory`.

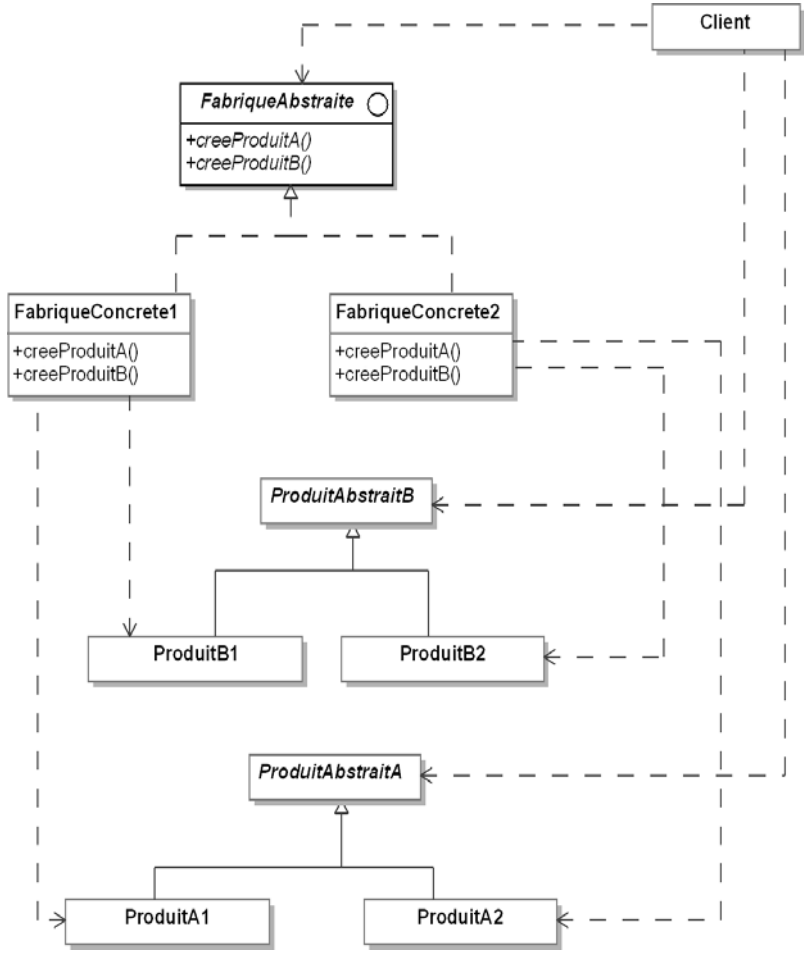

Figure 4.2 - Structure du design pattern `Abstract Factory`

3.2 Participants

Les participants au design pattern `Abstract Factory` sont les suivants :

- `FabriqueAbstraite` (`FabriqueVehiculeInterface`) est une interface spécifiant les signatures des méthodes créant les différents produits.
- `FabriqueConcrete1`, `FabriqueConcrete2` (`FabriqueVehicule-Electricite`, `FabriqueVehiculeEssence`) sont les classes concrètes implémentant les méthodes créant les produits pour chaque famille de produits. Connaissant la famille et le produit, elles sont capables de créer une instance du produit pour cette famille.
- `ProduitAbstraitA` et `ProduitAbstraitB` (`AbstractScooter` et `AbstractAutomobile`) sont les classes abstraites des produits indépendamment de leur famille. Les familles sont introduites dans leurs sous-classes concrètes.
- `Client` (`Catalogue`) est la classe qui utilise l'interface `FabriqueAbstraite`.

3.3 Collaborations

La classe `Catalogue` utilise une instance de l'une des fabriques concrètes pour créer ses produits au travers de l'interface exposée par `FabriqueAbstraiteInterface`.

Remarque

Il est recommandé de ne créer qu'une seule instance des fabriques concrètes, celle-ci pouvant être partagée par plusieurs clients. Nous verrons plus tard un design pattern capable de garantir qu'une seule instance d'une classe est disponible à l'exécution : `Singleton`.

4. Domaines d'utilisation

Le design pattern `Abstract Factory` est utilisé dans les domaines suivants :

– Un système utilisant des produits a besoin d'être indépendant de la façon dont ces produits sont créés et regroupés.

– Un système est paramétré par plusieurs familles de produits qui peuvent évoluer.

5. Exemple en PHP

Voici maintenant un exemple d'utilisation du design pattern écrit en PHP. Le code PHP correspondant à la classe abstraite `AbstractAutomobile` et ses sous-classes est donné à la suite. Il est très simple, il décrit les quatre propriétés des automobiles ainsi que la méthode `afficheCaracteristiques` qui permet de les afficher.

```
<?php

declare(strict_types=1);

namespace ENI\DesignPatterns\AbstractFactory;

abstract class AbstractAutomobile
{
   protected string $marque;

   protected string $couleur;

   protected int $puissance;

   protected float $espace;

   public function __construct(string $marque, string $couleur,
int $puissance, float $espace)
   {
       $this->marque = $marque;
       $this->couleur = $couleur;
       $this->puissance = $puissance;
```

```
        $this->espace = $espace;
    }

    abstract public function afficheCaracteristiques(): void;
}


<?php

declare(strict_types=1);

namespace ENI\DesignPatterns\AbstractFactory;

class AutomobileElectricite extends AbstractAutomobile
{
    public function afficheCaracteristiques(): void
    {
        echo "Automobile électrique - marque: $this->marque"
            . ", couleur: $this->couleur"
            . ", puissance: $this->puissance"
            . ", espace: $this->espace" . PHP_EOL;
    }
}


<?php

declare(strict_types=1);

namespace ENI\DesignPatterns\AbstractFactory;

class AutomobileEssence extends AbstractAutomobile
{
    public function afficheCaracteristiques(): void
    {
        echo "Automobile à essence - marque: $this->marque"
            . ", couleur: $this->couleur"
            . ", puissance: $this->puissance"
            . ", espace: $this->espace" . PHP_EOL;
    }
}
```

Le code PHP correspondant à la classe abstraite `AbstractScooter` et ses sous-classes est donné à la suite. Il est similaire à celui des automobiles, à