Travaux pratiques/dirigés de *NFA 032* – Année *2023-2024*
Enseignant *: Pierre Sylvain ILOGA BIYIK*

Le pattern Builder

Objectif : Abstraire la construction d'objets complexes de leur implantation de sorte qu'un client puisse créer ces objets complexes sans devoir se préoccuper des différences d'implantation.

Prérequis : Langage UML, tableaux, classes abstraites, héritage, encapsulation, exceptions.

Cas pratique 1 : Considérez l'entreprise de vente de véhicules du TP portant sur le pattern *Abstract Factory*. Supposez maintenant qu'après avoir parcouru le catalogue de véhicules, un client souhaite commander un véhicule qui l'intéresse. Pour cela, le client doit transmettre à l'entreprise son *nom* (chaîne de caractère) et l'*identifiant du véhicule* concerné (entier). Dans cet exercice, le numéro de ligne du véhicule dans le catalogue est considéré comme identifiant. Un vendeur va donc prendre en charge la commande du client en ajoutant successivement les documents suivants à une *liasse de documents* relatifs à cette commande : un *bon de commande* et une *demande d'immatriculation*. Les documents de cette liasse peuvent être imprimés par le vendeur soit au format *HTML*, soit au format *PDF*, selon le choix du client.

Travail à faire : Vous supposerez que le catalogue contient exactement les 13 véhicules contenus dans le Tableau 3, du cas pratique 3, du TP portant sur le pattern *Abstract Factory*. Il vous est demandé d'utiliser le pattern ***Builder*** pour écrire un programme Java qui va implémenter ce système de gestion de commandes des clients. Votre programme doit permettre à chaque client saisir au clavier les informations relatives à sa commande tel que décrit ci-dessous, après quoi le programme va afficher à l'écran la liste des documents générés par le vendeur avec des messages adéquats :

```
Nom client : ___
Id véhicule : ___
Format des documents - (1) Html  (2) Pdf : ____
```

I- Compréhension de l'énoncé

Afin d'apporter une solution au cas pratique 1, vous devez dans un premier temps vous servir de l'énoncer pour répondre aux questions suivantes :

1- Quel produit générique est manipulé par le vendeur lors du traitement d'une commande ?
2- Quel est le contenu spécifique en nature et en quantité de ce produit générique ?
3- Quelles sont les méthodes de ce produit générique ?
4- Quels sont les produits dérivés de ce produit générique ?
5- Quels sont les formats possibles de ces produits dérivés ?
6- Quelles sont les informations transmises par le client lors du passage de sa commande ?

II- Programmation en Java

Considérez le diagramme UML général associé au Pattern ***Builder*** présenté à la Figure 1. Lorsque ce diagramme est adapté à l'énoncé du cas pratique 1, on obtient le diagramme de la Figure 2.

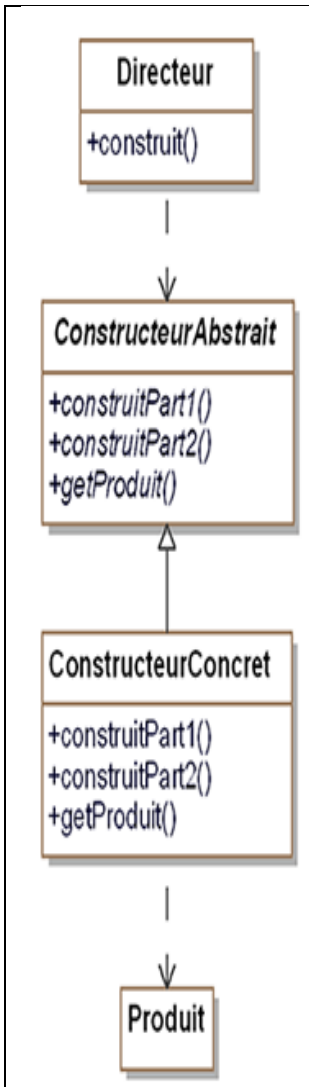

Figure 1

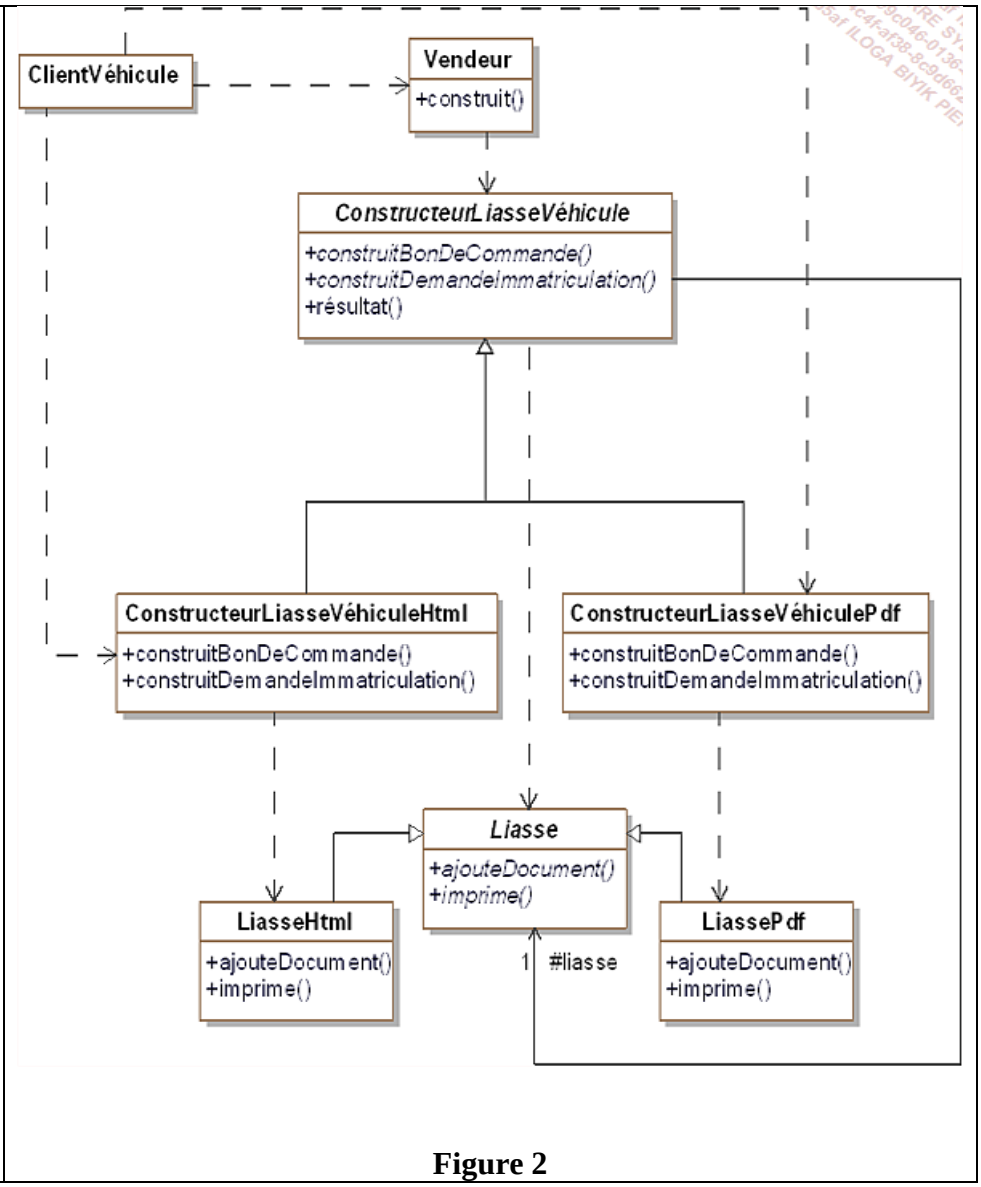

Figure 2

En vous appuyant sur les diagrammes UML des Figures 1 et 2, ainsi que sur les réponses aux questions de la Section I, répondez aux questions suivantes :

1- Créez une classe abstraite correspondant au produit générique identifié à la Question I-1. Cette classe doit avoir des attributs qui reflètent le contenu en nature et en quantité de ce produit générique identifié à la Question I-2. Elle doit aussi disposer d'un constructeur adéquat pour initialiser convenablement ces attributs. Enfin, cette classe doit disposer des méthodes (abstraites) identifiées à la Question I-3.

2- Pour chacun des produits dérivés identifiés à la Question I-4, vous devez créer une classe fille concrète qui va hériter de la classe abstraite créée à la Question II-1. Chaque classe fille a le même constructeur que sa classe mère, mais elle doit implémenter de manière spécifique chaque méthode abstraite conformément à l'énoncé.

3- Créez une classe abstraite dont l'objectif est de permettre la fabrication de chaque produit dérivé. Cette classe doit donc avoir un unique attribut protégé de type produit générique pour lequel elle disposera aussi de méthodes (abstraites) pour la fabrication de chaque contenu spécifique en fonction des informations transmises par le client et identifiées à la Question I-6. Enfin, cette classe doit avoir une méthode concrète permettant de retourner son unique attribut protégé.
4- Pour chacun des formats possibles du produit générique identifiés à la Question I-5, créez une classe fille concrète qui va hériter de la classe abstraite créée à la Question II-3 et qui va effectivement ajouter du contenu spécifique au produit dérivé concerné. Le constructeur de chaque classe fille concrète va simplement instancier (créer) l'unique attribut protéger comme étant du format adéquat.
5- Créez une classe correspondant au Vendeur qui va disposer d'un unique attribut protégé lui permettant de fabriquer le produit concret commandé par chaque client. Cette classe va aussi disposer d'une méthode permettant de fabriquer et de retourner le produit concret associé à la commande d'un client dont les informations sont prises en paramètres. Le constructeur de cette classe va simplement adapter son unique attribut au mode de fabrication choisi par le client pris en paramètre.
6- Enfin, vous allez créer la classe associée au programme principal qui correspond ici au *Client*. Cette classe va se servir de toutes les classes créées depuis le début afin de simuler le passage d'une commande de véhicule par le client conformément aux consignes données dans l'énoncé.

Cas pratique 2: Quelques années plus tard, l'entreprise va évoluer en permettant à chaque vendeur de manipuler une liste de documents au format *XML*, en plus des formats *HTML* et *PDF*. Dans les mêmes conditions que dans le cas pratique 1, il vous est demandé d'utiliser le pattern ***Builder*** pour écrire un programme Java qui va prendre en compte cette évolution en permettant à chaque client saisir au clavier les informations relatives à sa commande tel que décrit ci-dessous, après quoi le programme va afficher à l'écran la liste des documents générés par le vendeur avec des messages adéquats :

```
Nom client : ____
Id véhicule : ____
Format des documents - (1) Html  (2) Pdf  (3) XML : _____
```

Cas pratique 3 : Par la suite, une nouvelle équipe dirigeante de l'entreprise a décidé que l'entreprise va aussi fournir au client une *facture*, en plus du bon de commande et de la demande d'immatriculation. La facture contient le nom du client, l'identifiant du véhicule commandé et le prix du véhicule en euros (entier). Dans les mêmes conditions que dans le cas pratique 2, il vous est demandé d'utiliser le pattern ***Builder*** pour écrire un programme java qui va prendre en compte cette autre évolution en permettant à chaque client saisir au clavier les informations relatives à sa commande tel que décrit ci-dessous, après quoi le programme va afficher à l'écran la liste des documents générés par le vendeur avec des messages adéquats :

```
Nom client : ____
Id véhicule : ____
Prix véhicule : ____
Format des documents - (1) Html (2) Pdf  (3) XML :
```

Problème : Il vous est demandé d'écrire un programme Java qui va fusionner le cas pratique 3 du TP portant sur le pattern **Abstract Factory** et le cas pratique 3 du présent TP portant sur le pattern **Builder**. Vous allez pour cela créer un nouveau projet contenant trois packages :

- Un package nommé *catalogue* chargé du remplissage et de la navigation dans le catalogue comme stipulé dans le cas pratique 3 du TP portant sur le pattern **Abstract Factory**, avec la possibilité pour l'utilisateur de choisir le véhicule qu'il souhaite acheter en entrant le caractère '*c*'puis le numéro du véhicule choisit. De plus, vous allez considérer que chaque véhicule est aussi caractérisé par son prix en euros (entiers), en plus des autres caractéristiques (modèles, couleur, etc.). Ce prix doit aussi être affiché lors de la navigation dans le catalogue et vous choisirez vous-même la valeur du prix de chaque véhicule lors du remplissage du catalogue. Aucune classe de ce package ne doit contenir la fonction *main*.
- Un package nommé *documents* chargé de la génération de la liasse des documents relatifs à l'achat du véhicule comme stipulé dans le cas pratique 3 du présent TP portant sur le pattern **Builder**. A la différence que chaque document ne doit plus contenir le numéro du véhicule, mais plutôt les informations détaillées relatives au véhicule. Le client n'a plus besoin d'entrer le prix du véhicule dans ce package car il est déjà disponible dans le catalogue. Aucune classe de ce package ne doit contenir la fonction *main*.
- Un package nommé *principale* composé d'une seule classe contenant la fonction *main* qui va faire appel aux fonctions des deux autres package pour gérer l'ensemble du processus séquentiel des opérations suivantes réalisées par le client : *navigation dans le catalogue* → *choix du véhicule* → *choix du format des documents*. Ces opérations doivent reposer sur les interfaces graphiques suivantes :