

# Les Patrons de Construction

## Exercice 1

Nous souhaitons créer des identifiants uniques dans une application. Pour cela, une solution simple consiste à utiliser une étiquette "ID\_" suivi d'un compteur qui augmente strictement à chaque création.

**Question 1** • Créer un générateur qui permet d'avoir une chaîne de caractères différente à chaque appel en suivant la méthode ci-dessus. Faire de ce générateur un **singleton** de façon à garantir l'unicité du générateur, et donc des identifiants créés.

**Question 2** • Implanter et tester le générateur.

## Exercice 2

Nous souhaitons gérer dans une application interne d'une entreprise la classe *Imprimante*. Les employeurs utilisent une seule et unique imprimante. Nous souhaitons avoir un modèle qui permet de créer un seul objet de type *Imprimante* et de pouvoir maintenir et récupérer à n'importe quel moment le nombre de pages imprimés depuis le début de service de l'imprimante en question.

**Question 1** • Quel pattern de conception permet-il de modéliser le fonctionnement de l'imprimante ?

**Question 2** • Donner le diagramme de classes qui permet de répondre au besoin de l'entreprise.

**Question 3** • Implanter et tester l'imprimante.

## Exercice 3

Les clients d'une banque sont classés en deux catégories : Ceux qui ont le droit au crédit ; Ceux qui n'ont pas ce droit. Lors de la demande d'une carte de paiement, les premiers reçoivent une carte de crédit ( à débit différé sur leur compte) alors que les seconds peuvent seulement avoir une carte de débit ( à débit immédiat sur leur compte).

**Question 1** • Quel pattern de conception permet-il de modéliser la création de la carte de paiement en fonction du client ?

**Question 2** • Modéliser son utilisation par un diagramme de classes.

Il existe deux modèles de cartes de débit et de crédit, à savoir les cartes VISA et les cartes MASTERCARD.

**Question 3** • Modéliser, à l'aide d'un diagramme de classes, la création d'une carte de paiement en fonction de sa famille (crédit/débit).

**Question 4** • Implanter et tester les cartes.

#### Exercice 4

## Partie UML

### 1 Factory Pattern

Au cours de cet exercice, nous allons construire un diagramme de classe de manière incrémentale. Chacune des questions enrichira donc celui-ci. Pour chaque classe, indiquer ses attributs (*privés*), ses opérations (incluant la redéfinition de la méthode `toString()`), et son ou ses constructeurs. Pour simplifier, on omettra les accesseurs.

#### 1.1 Diagramme de base

On souhaite gérer des usines fabriquant des avions. Vous baserez votre conception sur le diagramme suivant : Un avion est caractérisé par une marque (*String*) et un modèle (*String*), et est associé à un fuselage et à ses réacteurs.

- Le fuselage d'un avion est caractérisé par un poids (*int*) et une capacité (*int*) décrivant le nombre de passagers qu'il peut accueillir.

**Question 1** • créer le diagramme de classes comprenant la classe **Fuselage**.

- Un réacteur est caractérisé par une marque (*String*), un poids (*int*) et une poussée (*int*).

**Question 2** • Compléter le diagramme avec la classe **Reacteur**.

- Un avion est caractérisé par une marque (*String*) et un modèle (*String*). Il est associé à un fuselage et à une liste d'au moins deux réacteurs.

**Question 3** • Compléter le diagramme avec la classe **Avion**.

**Question 4** • Compléter le diagramme avec les classes **Airbus** et **Boeing** qui sont des sous-classes d'**Avion**.

*La subtilité ici est que le constructeur n'a pas besoin de la marque, mais qu'il a besoin des autres arguments...*

#### 1.2 Factory Pattern

Mettons maintenant en place l'usine...

- Une usine de fabrication d'avions est caractérisée par un constructeur (*String*) et une ville (*String*).

**Question 5** • Compléter le diagramme avec la classe **Usine** contenant une méthode abstraite `fabriqueAvion(modeleAvion, marqueReacteur)` qui retourne un avion.

- Compléter le diagramme avec les classes concrètes **UsineAirbus** et **UsineBoeing** qui héritent de **Usine**.

La première ne fabrique que des **Airbus**, la seconde que des **Boeing**.

### 1.3 Abstract Factory Pattern

Apprenez maintenant que ces deux constructeurs, Boeing et Airbus, construisent non seulement des avions, mais aussi des satellites.

**Question 6** • Ajouter à votre diagramme la méthode abstraite `Satellite fabriqueSatellite(modeleSatellite)` dans la classe **Usine**, ainsi que la classe abstraite **Satellite**.

Airbus fabrique des satellites d'observation terrestre, Boeing fabrique des satellites d'observation spatiale.

**Question 7** • Ajouter à votre diagramme les classes **SatelliteTerrestre** et **SatelliteEspace**, qui héritent de **Satellite**. Complétez avec les dépendances aux usines de deux constructeurs.

### 1.4 Singleton Pattern

Tous les avions sont enregistrés dans un registre mondial unique. (En fait, chaque pays en possède une partie, mais c'est un détail...)

**Question 8** • Compléter votre diagramme avec une dernière classe **Immatriculation**. Cette classe doit être un singleton (donc instanciée au plus une seule fois). Pour ne pas surcharger le diagramme, on se limitera à un lien de dépendance unique à partir de la classe **Usine**.

## Partie Java

L'objectif de cette partie est d'implanter et de tester les différents diagrammes que vous avez réalisés en TD.

**Question 9** • Reprenez pour cela votre diagramme, et implantez les différentes classes en commençant par les plus simples. N'oubliez pas de tester **dès que possible** !

Afin de tester, nous vous proposons de créer les avions suivants :

Constructeur	Modèle	Fuselage (tonnes)	Capacité	Réacteurs			
				Nb	Marque	Poids (t)	Poussée (t)
Airbus	320	30	120	2	Rolls-Royce	5	10
Airbus	320	30	120	2	Trent	6	9
Airbus	380	110	550	4	Rolls-Royce	8	12
Airbus	380	110	550	4	Trent	9	13
Boeing	737	28	110	2	Rolls-Royce	4	8
Boeing	737	28	110	2	Trent	7	9
Boeing	747	100	375	4	Rolls-Royce	20	38
Boeing	747	100	375	4	Trent	22	41

Pour les satellites, on se limitera au *Jason1* de Airbus et à l'*Argos* de chez Boeing.