# TD et TP de Compléments en Programmation Orientée Objet n° 6 : Classes scellées, énumérations

## I) Fin du TP 5

Assurez vous d'avoir au moins terminé cet exercice du TP précédent.

### Exercice 1 : Rectangles et carrés, une solution ?

Comme cela a été illustré la semaine dernière par l'étude de la modélisation des rectangles et des carrés (ou des cercles et des ellipses etc ..), le problème est qu'on ne peut pas concilier à la fois l'invariant "cette figure est un XXX" et la spécification des mutateurs héritée du supertype. Une version immuable des figures devrait être plus robuste.

Évidemment, les mutateurs fournissaient une fonctionnalité utile. Pour les remplacer, il est possible d'écrire des méthodes retournant un nouvel objet identique à `this`, sauf pour la propriété qu'on souhaite "modifier". Exemple : `rect.withLongueur(15)` retourne un rectangle identique à `rect` mais dont la longueur est 15 ; en revanche, `carre.withLongueur(15)`, pour préserver l'indépendance des propriétés largeur et longueur, retournera aussi un rectangle et non un carré.

**À faire :**

1. Écrivez les classes immuables `RectangleImmuable` et `CarreImmuable` sous-classes de `Rectangle` (classe abstraite fournissant les fonctionnalités communes, sans mutateur, mais avec les opérations `withXXX()` en tant que méthodes abstraites). Notez que les méthodes de "modification" de `CarreImmuable` respectent automatiquement l'invariant du carré car `this` n'est pas modifié.
2. Ci-dessus, pour empêcher la création de sous-types mutables, `RectangleImmuable` n'est pas extensible ; pour cette raison, `CarreImmuable` n'en est pas un sous-type.

   Cependant la technique des classes scellées permet, pour une classe donnée, de définir une liste fermée de sous-types et donc de garantir l'immuabilité du supertype. Utilisez cette technique pour définir des types immuables `Rectangle` et `Carre` avec `Carre` sous-type de `Rectangle` (imbriquez vos déclarations dans une classe-bibliothèque `FormesImmuables`).

## II) Cartes

### Exercice 2 : Jeu de cartes – Modélisation

Le jeu de cartes classique se compose de 54 cartes : 52 cartes dans 4 familles correspondant à 4 enseignes différentes (pique, cœur, carreau, trèfle), chacune composée de 13 valeurs (as, deux, trois, .... neuf, dix, valet, dame, roi), ainsi que 2 jokers.

1. Pour le jeu à 52 cartes (sans joker), proposez une modélisation objet utilisant les énumérations, permettant de représenter toutes les cartes de ce jeu de cartes. Pour une carte donnée, il faut être capable de récupérer :
   — son enseigne (pique/cœur/carreau/trèfle, appelée souvent "couleur", mais pour éviter les confusions, nous éviterons ce terme)
   — sa valeur

— sa couleur (rouge ou noir)
— l'information si cette carte une figure (valet, dame, roi) ou pas
Essayez de faire en sorte de placer le plus d'implémentation possible au sein des `enum`, quitte à ce que les méthodes des cartes se contentent d'appeler celles des `enum`.

2. Adaptez vos classes (et peut-être même la structure de sous-typage) afin que les classes directement instanciables soient immuables.
3. Proposez une modélisation pour ajouter les jokers (il faut que les cartes "normales" soient instances d'un même type `CarteNormale` dont les jokers ne sont pas... mais il faut aussi que toutes les cartes, jokers compris, soient instances du type `Carte`).
4. Rendez le type `Joker` immuable.
   Maintenant, `CarteNormale` et `Joker` sont immuables. Mais est-ce que `Carte` l'est ? Si ce n'est pas le cas, corrigez ce problème (faites une classe scellée).
5. Programmez des méthodes statiques permettant de générer : un jeu de 52 cartes standard (retourné sous la forme d'une collection), un jeu de 32, un jeu de 54 (avec jokers),
6. Même question pour générer une collection contenant plusieurs exemplaires d'un jeu de carte (par exemple, pour jouer au Rami, il faut deux jeux de 54).

### Exercice 3 : Jeu de cartes – 8 américain (pour ceux qui ont vraiment fini le reste)

En utilisant les cartes programmées à l'exercice 2, programmez le jeu du 8 américain (cf. Wikipédia), ancêtre du jeu Uno. Programmez juste la version présentée comme "version minimale". Une version simple à 2 joueurs conviendra (humain contre humain, prévoyez un code simple pour désigner les cartes en ligne de commande ; intégrez judicieusement la reconnaissance de ce code dans les classes et énumérations de l'exercice 2).