# PCOO – TD/TP 2

*Vous devez terminer l'implémentation des programmes en dehors des TP*

### Exercice 1 – Formules (en TD et en TP)

Une fois l'exercice sur les formules précédent terminé, on peut remarquer que le code des `Sum` et `Product` sont similaires. En effet, les champs et le code du ou des constructeurs sont identiques. De plus, il est possible de rendre identique les codes des méthodes `String asString()` et `double asValue()` en effectuant quelques modifications.

1. Dans la méthode `String asString()`, seul le caractère inséré entre les formules est différent. Dans chacune des classes `Sum` et `Product`, définissez une méthode `String symbol()` qui retourne ce caractère. Réécrivez ensuite les méthodes `String asString()` de `Sum` et `Product` de sorte qu'elles utilisent la méthode `String symbol()`. Que peut-on dire de ces méthodes ?

2. De même, définissez les méthodes `double initialValue()` et `double cumulativeValue(double accumulator, double value)` dans les classes `Sum` et `Product`. Ces méthodes doivent retourner les valeurs suivantes :

| | Sum | Product |
|---|---|---|
| initialValue | 0 | 1 |
| cumulativeValue | accumulator + value | accumulator * value |

   Réécrivez ensuite la méthode `asValue()` des classes `Sum` et `Product` de sorte qu'elles utilisent les méthodes `initialValue` et `cumulativeValue`. Que peut-on dire de ces méthodes ?

3. Définissez une classe abstraite `VariadicOperator` qui "factorise" le code des méthodes `asValue()` et `asString()`. Les classes `Sum` et `Product` doivent étendre la classe `VariadicOperator`. Quelles sont les méthodes abstraites ? Est-ce que la classe `VariadicOperator` doit implémenter `Formula` ?

### Exercice 2 – Vecteur et pile d'objets (en TD et en TP)

1. Écrivez les classes `Vector` et `Stack` qui gèrent respectivement un vecteur et une pile d'objets. Ces deux classes fournissent les mêmes services que les classes `Vector` et `Stack` du TP 1.

2. Redéfinissez la méthode `String toString()` dans les classes `Vector` et `Stack` afin de générer une chaîne à partir des chaînes générées par les méthodes `toString` des éléments présents dans les deux structures.

3. Définir une classe `Card` qui contient deux champs entiers `suit` et `rank`. L'entier `suit` représente la couleur de la carte : 0 = Trèfle, 1 = Carreau, 2 = Coeur et 3 = Pique. L'entier `rank` représente la valeur de la carte : 0 = 2, 1 = 3, ..., 10 = Dame, 11 = Roi, 12 = As. Le constructeur de la classe `Card` permet d'initialiser la couleur et la valeur de la carte.

4. Redéfinissez la méthode `String toString()` de façon à générer une chaîne de caractères qui décrit la carte (par exemple, "2 de trèfle").

5. Dans la méthode `main`, ajoutez des cartes générées aléatoirement dans un vecteur et affichez le vecteur. Faites de même avec une pile. Pour générer un nombre aléatoirement, vous devez instancier la classe `Random` de la bibliothèque Java et utiliser sa méthode `nextInt(int n)` qui retourne un entier compris entre 0 et $n - 1$ inclus.

6. Comment doit-on procéder pour mettre des entiers dans les instances des classes `Vector` et `Stack` ?

### Exercice 3 – Tracer un vecteur d'objets (en TD et en TP)

Écrivez la classe `VerboseVector` de façon à pouvoir tracer les différentes opérations effectuées par la classe `Vector`. Utilisez l'extension afin que cette classe puisse très facilement remplacer une instance de la classe `Vector` de l'exercice 1 de ce TD. Par exemple, le code suivant :

```
Vector vector = new VerboseVector();
vector.add(new Card(0, 0));
vector.add(new Integer(2));
vector.set(1, new Integer(3));
Object object = vector.get(1);
```

doit produire la sortie suivante :

```
add(2 de trèfle);
add(2);
set(1,3);
get(1);
```