# TP, première séquence d'exercices.

Benoît Valiron
benoit.valiron@lipn.univ-paris13.fr

7 novembre 2010

## Introduction

Vous écrirez les réponses aux questions courtes sur une feuille à rendre à la fin de la séance. Ne rédigez que les parties pertinentes des codes demandés. Pour les exercices 5, 6, 7, 8 et 9, vous placerez les codes `.java` dans un fichier `zip` que vous m'enverrez par mail à la fin de la séance. Il est de votre responsabilité de vous assurer que je l'ai reçu.

## Exercice 1.

1. Ouvrez un terminal de ligne de commande. Assurez-vous d'être dans votre répertoire de travail en tapant

   ```
   $ cd
   ```

   (le symbole `$` est le prompt. Ne l'entrez pas !)
2. Créez un répertoire JAVA et un répertoire pour le TP :

   ```
   $ mkdir JAVA; cd JAVA
   $ mkdir TP1; cd TP1
   ```

   Dans les TPs qui suivront, vous procéderez de la même façon.
3. Ouvrez un éditeur de texte, par exemple :

   ```
   $ xemacs &
   ```

   ou

   ```
   $ gedit &
   ```

   Créez un nouveau fichier `Exo1.java` et écrivez le code

```
1 public class Exo1 {
2
3   public static void main(String[] args) {
4     int i, r;
5
6     r = 0;
7     for(i = 0; i < 5; ++i) {
8       r += i;
9     }
```

```
10
11      System.out.println(r);
12    }
13  }
```

Sauvez le fichier.

4. À l'aide de la ligne de commande, compilez-le :

   ```
   $ javac Exo1.java
   ```

   Cette commande crée un fichier `Exo1.class`, que vous pouvez lancer avec la "Java Virtual Machine" (JVM)

   ```
   $ java Exo1
   ```

   Qu'écrit-elle à l'écran ? Comment est calculé le nombre affiché ?

# Exercice 2.

Créez un fichier `Exo2.java` avec le même contenu que `Exo1.java`, par exemple avec les commandes :

```
$ cp Exo1.java Exo2.java
```

Modifiez le nom de la classe pour correspondre au nom du fichier ! Sinon ça ne compilera pas.

Transformez le code de `Exo2.java` pour faire les choses suivantes :

1. instancier un tableau de 10 entiers `int`.
2. placer dans chaque cellule d'indice `i` du tableau la valeur `i*2`, à l'aide d'**une boucle** `while`.
3. à l'aide d'une boucle `for`, imprimer les valeurs du tableau dont les indices sont pairs (0, 2, 4, 6, ...).

# Exercice 3.

1. Exécutez le code suivant :

   ```
   public class Exo3 {

     public static void main(String[] args) {
       float r = 3 / 4;
       double s = 3.0 / 4.0;

       System.out.println(r);
       System.out.println(s);
     }
   }
   ```

   Pourquoi le nombre flottant (c'est à dire à virgule) ne vaut pas 0.75 comme on pourrait s'y attendre ?

2. Changez `double` en `float` et recompiler. Que dit le message d'erreur ?

## Exercice 4. Notion de classe

Dans cet exercice, vous allez travailler avec la classe `Point` du cours.
La classe correspondante est donnée par

```
public class Point {
  double x;
  double y;

  Point(double a, double b) {
    x = a;
    y = b;
  }

  Point() {
    this(0.0, 0.0);
  }

  public void translate(double vx, double vy) {
    x += vx;
    y += vy;
  }
}
```

**Encore une fois, cette classe doit être placée dans un fichier `Point.java`.**

**Questions :**

1. Ajoutez une méthode non statique `affiche` à la classe `Point` qui imprime les coordonnées du point à l'écran. Compilez votre code.
2. Ajoutez une méthode statique `afficheStatique` qui affiche aussi un point à l'écran. Compilez votre code.
3. Ajoutez une méthode `main` pour pouvoir compiler et exécuter le programme. Dans la méthode, allouez deux points `p` et `q`. Initialisez `p` à $(1.2, 3.4)$, et faites `p = q`. Afficher les coordonnées de `p`. maintenant, déplacez `q` du vecteur $(1.0, 1.0)$. Affichez de nouveau les coordonnées de `p`. Compilez le code et faites-le tourner.
4. Expliquez les résultats.
5. Utisez les deux méthodes `affiche` et `afficheStatique` pour afficher les coordonnées de `p`.

Il est possible d'utiliser la classe `Point` ailleurs.

6. Créez un nouveau fichier `Exo4.java` avec une classe `PointTest` et une fonction `main`. Coupez-collez la méthode `main` de la classe `Point` dans la classe PointTest, sauvegardez les deux fichiers et compilez votre code :

   ```
   $ javac Test.java
   $ javac Exo4.java
   ```

7. Pourquoi la compilation de `Exo4.java` ne marche pas ? Corrigez le problème.
8. Éxécutez la classe qui contient la méthode `main`. Vous devez obtenir la même chose qu'avant.

## Exercice 5 : `String`

1. Créez une classe `Exo5` avec une méthode `main` dans laquelle vous allouez un object `Point p` de valeur $(1.0, 0.0)$. Placez ensuite la commande

   ```
   System.out.println(p);
   ```

   Compilez, lancez le programme. Que constatez-vous ? Pourquoi ?
2. Ajoutez une méthode `public String toString()` dans la classe `Point`, qui rends la chaine de caractère sous la forme "(x, y)". Recompilez `Point` et `Exo5`, et relancez `Exo5`. Que constatez-vous ? Pourquoi ?
3. Modifiez la méthode d'affichage de la classe `Point` pour utiliser la méthode `toString`.

## Exercice 6 : Tampon d'entiers

Les tableaux possèdent l'inconvénient d'avoir une taille fixée au départ et qui ne peut être modifiée au cours de l'utilisation.

Dans cet exercice vous allez créer une classe `TamponEntiers` avec :

- un champ `int[] tableau`, avec une taille initiale communiquée en argument du constructeur unique de la classe.
- un champ `int index` qui indique quelle est la première case vide du tableau.
- une méthode `public void agrandir()` qui créé un nouveau tableau plus grand, copie le tableau `tableau` dedans, et remplace `tableau` par ce tableau plus grand.
- une méthode `public void ajouter(int e)` qui ajoute l'élément donné en argument à la fin du tableau. Si il n'y a plus de place, on utilise au préalable la méthode `agrandir`.
- une méthode `public void vider()` qui vide `tableau` : la méthode remet simplement la valeur `index` à zero.
- une méthode `toString` qui écrit le tableau dans une chaine sous la forme `"[1,2,2,2,1,3,7...]"`. Attention, il n'y a pas plus d'éléments que la valeur `index` !
- une méthode `void afficher()` qui affiche les éléments dans l'ordre sur la sortie standard, avec un saut de ligne entre chaque.
- une méthode `int element(int i)` qui rends l'élément d'indice `i`.
- une méthode `int nbElements()` qui rend le nombre d'éléments du tampon.

Toutes les méthodes sont publiques.

Questions :

1. Créez la classe `tamponEntiers`.
2. Testez-la en ajoutant une méthode `main` à la classe, et allouez un tampon de 5 éléments. Puis faites une boucle sur $i = 0...25$ dans laquelle pour chaque pas de boucle vous ajoutez `i` au tampon puis affichez la chaine de caractères correspondant au tampon. Après la boucle, videz le tampon, et affichez la chaine de caractères correspondante.
3. On construit la suite d'entiers suivante. Choisissez un entier quelconque $n$. Si $n$ est pair, l'entier suivant est $n/2$. Si $n$ est impair, l'entier suivant

est $3n+1$. Recommencez avec ce nouvel entier. La conjecture de Syracuse dit que la suite produite termine toujours sur la valeur 1, quel que soit l'entier choisi. Par exemple, en commençant par 6, on obtient :

$$6, 3, 10, 5, 16, 8, 4, 2, 1.$$

Créez une classe `Syracuse` avec une méthode `main` qui calcule la suite de Syracuse commençant avec 5. On utilisera un object de type `tamponEntier` pour stocker les éléments successif de la suite. Lorsque 1 est atteint, on affichera la suite, ainsi que le nombre d'éléments dans la suite.

Faites le test avec les suite commençant par $4, 5, 27$ et $29$.

4. Utilisez la méthode statique `int parseInt(String s)` de la classe `java.lang.Integer`, pour que l'entier de départ soit le premier argument du programme :

   ```
   $ java Syracuse mon_entier_favori
   ```

## Exercice 7 : `StringBuffer`

Dans l'exercice 7, vous avez probablement fait une boucle pour la méthode `toString`. Au niveau de la gestion de la mémoire, à chaque appel de `String.concat` (ou "`+`"), une nouvelle chaine de caractère est allouée et initialisée. Ça n'est pas efficace. Pour répondre à ce type de demande, une classe `StringBuffer` existe en java : elle fonctionne sur le même modèle que la classe `TamponEntiers`. La documentation de la classe se trouve en annexe. Modifiez la méthode `toString` de l'exercice précédant pour utiliser un object `StringBuffer`. Vous êtes sensé :

- allouer un object `StringBuffer` ;
- concaténer toutes les chaines de caractères dedans en utilisant la méthode idoine ;
- renvoyer une chaine de caractères avec le contenu de l'objet.

## Exercice 8 : Fractions

1. Sur le modèle de la classe `Point`, écrivez une classe `Fraction`. Vous inclurez des méthodes `add`, `mult` et `neg` pour additionner, multiplier et prendre la négation d'une fraction. Les méthodes demandées sont sensées modifier la fraction considérée.

   On veut aussi une méthode `toString` qui écrit la fraction dans le format standard "x/y" et une méthode d'affichage qui affiche la fraction à l'écran.

   On veut deux constructeurs : un qui prend deux entiers, le numérateur et le dénominateur, et un autre constructeur qui prend un seul entier $n$, et rends la fraction $n/1$.

   On rappelle qu'une fraction est la donnée de deux entiers, un numérateur et un dénominateur :

   $$\frac{numerateur}{denominateur}$$

   Le dénominateur n'est jamais nul. Quand elles sont valides, les opérations demandées sont définis comme suit :

   $$\frac{a}{b} + \frac{c}{d} = \frac{ad+bc}{bd}, \quad \frac{a}{b} \times \frac{c}{d} = \frac{a \times c}{b \times d}, \quad neg\left(\frac{a}{b}\right) = \frac{-a}{b}.$$

Vous placerez une méthode `main` dans la classe pour tester les méthodes définies.

2. Créez une classe FractionTest avec une méthode `main` qui calcule la somme des fractions
$$\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \cdots + \frac{1}{10}$$
en utilisant une boucle `while`, et qui affiche le résultat.
3. Ajoutez une méthode statique à la classe `FractionTest` qui calcule la puissance $n$ d'une fraction, pour un entier $n$ donné.
4. Utilisez votre nouvelle méthode en ajoutant à la suite de la méthode `main` le code qui permet de calculer
$$\frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^{10}}.$$