IP1
TP noté – Groupe B

Jeudi 4 décembre 2014

Merci de rendre vos fichiers **.java** (et pas .class) dans le **bon** travail Didel **TP noté n°2 - INFO1 - groupe B (14h45)**, et de le faire **au fur et à mesure** du TP.
Les exercices sont **indépendants** et peuvent être traités dans n'importe quel ordre. Il n'est pas nécessaire de tout faire pour avoir une bonne note. On rappelle cependant qu'un exercice qui ne compile pas (c'est-à-dire qui bugue quand on lance `javac`) ne donne pas de points.

L'archive contenant les sources à compléter doit être récupéré à l'adresse :

`http://alturl.com/tubru`

Les travaux doivent être rendus sur Didel, sur le travail TP noté n°2 - INFO1 - groupe B (14h45).

Aucun document n'est autorisé. Les recherches sur Internet et l'utilisation de téléphone portable ou d'autre moyen de communication sont interdites. Toute triche sera sévèrement sanctionnée.

1 Somme de nombres non premiers

Dans cet exercice, on demande de remplir le fichier `Prime.java`. Regardez bien les fonctions qui vous y sont fournies, et notamment `isPrime`, que vous n'avez pas à reprogrammer. Cette fonction `isPrime` prend en argument un entier et renvoie le booléen `true` s'il est premier et `false` sinon.

1. En utilisant la fonction `isPrime`, définissez et programmez la fonction `sumPrime` qui prend en argument un entier n et retourne l'entier correspondant à la somme des nombres qui ne sont pas premiers contenus dans $\{1, \ldots, n\}$.
   Par exemple, pour $n = 12$, les nombres premiers de 1 à 12 étant 2, 3, 5, 7 et 11, la fonction calculera
   $$1 + 4 + 6 + 8 + 9 + 10 + 12 \;=\; 50$$
   et renverra donc 50.
2. Ajoutez ensuite une fonction `main`, de sorte que le programme `Prime` prenne en argument un entier et affiche le résultat de `sumPrime(n)`. En reprenant l'exemple, on demande que la commande

   ```
   java  Prime  12
   ```

   affiche

   ```
   La somme des nombres non premiers compris entre 1 et 12 est 50.
   ```

2 Lecture de tableaux

Dans cet exercice, on demande de remplir le fichier `ReadArrays.java`. Regardez bien les fonctions qui vous y sont fournies.

1. Remplissez la fonction `read`, qui prend en argument deux tableaux d'entiers `t1` et `t2` et renvoie une chaîne de caractères contenant les éléments de `t1` lus à l'endroit et séparés par des espaces, puis ceux de `t2` lus à l'envers et séparés par des espaces.

   Par exemple, le code suivant :

   ```
   int[] t1 = {3,2,5};
   int[] t2 = {8,9,10};
   showString(read(t1,t2));
   ```

   affichera

   3 2 5 10 9 8

   A noter que ce code est déjà présent dans la fonction `main` du programme. Après avoir programmé la fonction `read`, vous pouvez donc le compiler et le lancer, et vérifier qu'il affiche bien la ligne ci-dessus.
2. Programmez la fonction `convert` qui prend en argument un tableau de chaînes de caractères et renvoie le tableau d'entiers obtenu en convertissant chacune de ses cases en entier.
3. Modifiez alors la fonction `main` afin que votre programme affiche ses arguments, d'abord à l'endroit, puis à l'envers, puis retourne à la ligne. Par exemple, l'exécution

   ```
   java ReadArrays 27 8 32
   ```

   affichera

   ```
   27 8 32 32 8 27
   ```

   On pensera à utiliser les fonctions précédemment programmées.

3 Décompte dans un tableau de tableaux

Dans cet exercice, on demande de remplir le fichier `CountArrayOfArrays.java`. Regardez bien les fonctions qui vous y sont fournies.

Remplissez la fonction `compte` prenant en argument un tableau de tableaux d'entiers `t` et un entier `n`, et qui renvoie le nombre d'occurences de `n` dans `t`.

La fonction `main` du programme est déjà remplie. Si tout est correct, votre programme devrait afficher 3.

4 Boucle while et jets de dé

Dans cet exercice, on demande de remplir le fichier `HowMany.java`. Regardez bien les fonctions qui vous y sont fournies. La fonction `random` permet notamment d'obtenir un entier entre 1 et 6 au hasard. On l'utilisera pour simuler le jet d'un dé.

1. Programmez une fonction `decompte` prenant en argument un entier `n` et renvoyant le nombre d'appels à la fonction `random` nécessaires pour obtenir `n` fois la valeur 4.

   En d'autres termes, `decompte` compte le nombre de fois qu'il a fallu jeter le dé pour obtenir `n` fois un 4.

2. Ecrivez une fonction `main` utilisant le premier argument donné au programme pour appeler `decompte`, et affichant ensuite le résultat. Par exemple,

   ```
   java HowMany 10
   ```

   pourra renvoyer

   ```
   Il a fallu 60 jets de dé pour obtenir 10 fois un quatre.
   ```

   Attention, il est tout à fait normal que le programme n'affiche pas la même chose deux fois de suite : le lancer de dé étant aléatoire, on peut obtenir beaucoup de valeurs différentes. Cependant, la "moyenne" est 60. Si vous trouvez dix fois de suite un nombre plus petit que 40 ou plus grand que 80, il y a de bonnes chances qu'il y ait une erreur quelque part.