Examen de TP

Informatique Fondamentale (IF121)
15 Décembre 2004

Lorsqu'un exercice demande d'écrire une *méthode*, seule sera jugée et notée la méthode.

Il vous est parfois vivement conseillé d'appeler dans celle-ci des méthodes faisant l'objet des questions précédentes, ce que vous pouvez faire quelle que soit votre réussite à ces questions. Il vous est aussi possible (bien que jamais nécessaire) d'utiliser des méthodes auxiliaires de votre invention, qui feront alors partie intégrante de la note.

Il vous est toutefois fortement conseillé de tester vous-même vos méthodes en les incluant dans une classe exécutable avec une méthode `main`, bien que celle-ci ne sera pas jugée.

En revanche, lorsqu'un exercice vous demande d'écrire un *programme*, la réponse attendue est une classe exécutable (précédée des importations nécessaires à son fonctionnement) qu'il nous sera alors possible d'exécuter directement dans l'interpréteur java.

Le seul document autorisé est la documentation en ligne de la classe *Deug* à l'adresse :
`http://www.liafa.jussieu.fr/~yunes/deug/Deug/`

1 Algèbre

Exercice 1 : *Division entière*

- Écrire une méthode `diviser` qui, étant donnés deux entiers positifs k et n, calcule la division entière de k par n. Bien entendu, il est interdit d'utiliser l'opérateur `/` de java, puisqu'il s'agit de l'implémenter soi-même.
  La méthode aura donc la signature suivante :
  ```
  static int diviser(int k,int n){
      ...
      }
  ```
- Écrire une méthode `modulo` qui, étant donnés deux entiers positifs k et n, calcule *le reste* de la division entière de k par n. Bien entendu, il est interdit d'utiliser l'opérateur `%` de java, puisqu'il s'agit de l'implémenter soi-même.
  La méthode aura donc la signature suivante :
  ```
  static int modulo(int k,int n){
      ...
      }
  ```

Exercice 2 : *Nombres premiers*

- Écrire une méthode `isprime` qui, étant donné un entier positif n, renvoie `true` si n est un nombre premier, et `false` sinon.
  La méthode aura donc la signature suivante :
  ```
  static boolean isprime(int n){
      ...
      }
  ```
  Il sera bien vu d'utiliser la méthode `modulo` de l'exercice 1.
- En appelant la méthode `isprime`, écrire une méthode `prime` qui, étant donné un entier positif n, affiche tous les entiers premiers inférieurs ou égaux à n (un par ligne).
  La méthode aura donc la signature suivante :
  ```
  static void prime(int n){
      ...
      }
  ```

Exercice 3 : *Suite pour calculer π*

- Écrire une méthode `Un` qui, étant donné un flottant a et un entier positif n, renvoie la valeur de la suite parametrée u_n^a définie ainsi :

$$\begin{aligned} u_0^a &= a^2 \\ u_{n+1}^a &= \frac{a^2}{6+u_n^{a+2}} \end{aligned}$$

  La méthode aura donc la signature suivante :

```
static double Un(double a, int n){
    ...
    }
```

- On admet que la suite (u_n^1) converge vers $\pi - 3$. Inclure la méthode précedente dans un programme (classe exécutable `CalculDePi`) prenant un paramètre n et affichant l'approximation de π à partir du terme u_n^1. Nous testerons ce programme avec par exemple l'instruction `java CalculDePi 93` Il s'agit bien entendu de récupérer la valeur 93 à l'intérieur du programme pour calculer u_{93}^1.

2 Graphiques

Exercice 4 : *Boîtes imbriquées*

- Ecrire une méthode `afficheBoite` qui, étant donné trois entiers x, y et *taille* affiche un carré dont les coordonnées du point supérieur gauche sont x et y et dont les cotés sont de longueur *taille*.
  La méthode aura donc la signature suivante :

```
static void afficheBoite(int x, int y, int taille){
    ...
    }
```

- Ecrire un programme (classe exécutable `Dessin`) prenant quatre paramètres x, y, n et t et affiche n boîtes imbriquées dont le point supérieur gauche de la plus petite boîte (de taille t) est de coordonnées x et y.
  La taille de la fenêtre graphique sera 800x600 et la distance entre deux boîtes sera de 10.

3 Tableaux

Pour tous ces exercices, les méthodes ne devront pas génerer l'erreur *ArrayIndexOutofBoundException* quelles que soient les valeurs passées en paramètre.

Exercice 5 : *Recherche*

- Ecrire une méthode `rechercheTableau` qui, étant donné un tableau de caractères *tab*, un caractère c et 2 entiers a et b, teste si c existe dans le tableau entre les indices a et b inclus.
  La méthode aura donc la signature suivante :

```
static boolean rechercheTableau(char[] tab, char c, int a, int b){
    ...
    }
```

- Ecrire une méthode `nombreOccurences` qui, étant donné un tableau de caractères *tab* et un caractère c, calcule le nombre d'occurences de c dans le tableau. On peut éventuellement utliser la méthode précédente.
  La méthode aura donc la signature suivante :

```
static int nombreOccurences(char[] tab, char c){
    ...
    }
```

Exercice 6 : *Décalage*

- Ecrire une méthode `decal1` qui, étant donné un tableau d'entiers *tab*, décale les valeurs de ce tableau d'une case vers la droite. La denière valeur doit devenir la première.
  La méthode aura donc la signature suivante :
  ```
  static int[] decal1(int[] tab){
      ...
      }
  ```
- Ecrire une méthode `decalN` qui, étant donné un tableau d'entiers *tab* et un entier positif n, décale les valeurs de ce tableau de n cases vers la droite. La denière valeur doit devenir la première.
  La méthode aura donc la signature suivante :
  ```
  static int[] decalN(int[] tab, int n){
      ...
      }
  ```

Exercice 7 : *Moyenne olympique*

Ecrire une méthode `moyenneOlympique` qui, étant donné un tableau d'entiers positifs *tab*, calcule la moyenne olympique de ses valeurs. Dans cette moyenne, le plus grand et le plus petit entier ne sont pas pris en compte (référence au système de notation en usage aux J.O.). Le tableau ne devra être parcouru **qu'une seule fois**.
La méthode aura donc la signature suivante :

```
static double moyenneOlympique(int[] tab){
    ...
    }
```

Exercice 8 : *Recherche dichotomique*

Ecrire une méthode `rechercheDicho` qui, étant donné un tableau d'entiers `tab` ordonné et un entier `n`, recherche `n` dans le tableau par une méthode dichotomique : le principe pour localiser `n` entre la i^{ieme} et la j^{ieme} case de `tab` est de prendre la case du milieu entre i et j (i.e. $\frac{i+j}{2}$) et de comparer son contenu avec `n`. Suivant le résultat de cette comparaison, on réitère l'opération entre i et $\frac{i+j}{2}$ ou entre $\frac{i+j}{2}$ et j, et ainsi de suite.

On commence avec tout le tableau ($i = 0$ et $j =$ `tab.length`) et on s'arrête lorsqu'on ne peut plus découper ($j = i + 1$), auquel cas on renvoit i.

La méthode aura donc la signature suivante :

```
static int rechercheDicho(int[] tab, int n){
    ...
    }
```