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.

On rappelle que la documentation en ligne de la classe *Deug* est 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 Tableaux

Exercice 4 : *Permutations*

Le but de l'exercice est d'écrire une méthode `permut` qui affiche toutes les permutations des cases d'un tableau `tab`.

- Ecrire une méthode `affiche` qui affiche les éléments d'un tableau sur une même ligne, séparés par des espaces, et suivis d'un retour à la ligne.
  La méthode aura donc la signature suivante :

```
static void affiche(int[] tab){
    ...
    }
```

- Pour les permutations, on programme une méthode `aux` un peu plus générale que `permut` qui prend un paramètre n supplémentaire ($0 \leq n <$ `tab.length`) et affiche tous les tableaux qui ne touchent pas aux cases `tab[0]`...`tab[n-1]` mais permutent les cases `tab[n]`...`tab[tab.length-1]`.
  L'idée de la méthode `aux(int[] tab,int n)` est la suivante :
  -soit `tab.length` $= n+1$, auquel cas la seule permutation autorisée est le tableau lui-même, donc on appelle `affiche(tab)`.
  -soit `tab.length` $> n+1$, et on fait alors une boucle sur $n \leq i <$ `tab.length` dans laquelle : a) on échange `tab[i]` et `tab[n]` et b) on appelle la même méthode `aux` sur les paramètres `tab`, $n+1$.
  La méthode aura donc la signature suivante :

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

- Ecrire la méthode `static void permut(int[] tab){...}` qui ne fait qu'appeler `aux(tab,0)`.

Bien entendu, la plupart des points de cet exercice sont dans l'écriture de `aux`, le reste est simple.

Exercice 5 : *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){
    ...
    }
```