Université de Nice – Sophia Antipolis — Deug MASS MIAS SM 1re année
UEF1 : Informatique & Programmation — 2000–2001

TP N° 3

Classes & méthodes de classes – La classe `Console`

Ça va mieux en le révisant . . . La *classe* est le concept principal en programmation JAVA. Ecrire une application consiste à définir une ou plusieurs classes qui coopèrent à la résolution d'un problème.
Le nom d'un fichier JAVA est composé du nom de son unique classe suivi du suffixe `.java`. JAVA contient un très grand nombre de *classes prédéfinies* comme la classe `Math`.
Les *méthodes de classe* (précédées de `static`) sont des portions de codes qui ont un rôle bien précis : pour ce faire, elles renferment une suite d'instructions. Pour utiliser une méthode de classe d'une classe donnée (ou une constante), il suffit[1] de faire précéder le nom de classe (resp. de la constante) du nom de la classe (par exemple `Math.sqrt`(...) et `Math.PI`).

Dorénavant, on vous demande de soigner la mise en page de vos classes. Prenez-donc exemple sur celles du cours et des TP, en respectant l'*indentation*[2] proposée. Remarquez que cette indentation n'est pas du tout fantaisiste mais qu'elle obéit à des règles précises. Tâchez de les comprendre et de les utiliser.

Il faut toujours garder à l'esprit que les programmes que l'on écrit sont un jour ou l'autre lus par d'autres. Du coup, écrire clairement fait entièrement partie de l'activité de programmation.

1 Classes & méthodes de classes

Exercice 1) Nous allons créer une classe toute simple appelée `Numerik` que nous étofferons tout au long de l'année. Aujourd'hui, ce n'est pas tant son contenu qui importe que son utilisation. Créez la classe `Numerik`. Définissez-y les méthodes `distance`(...), `maximum`(...) et `minimum`(...). Elles auront chacune deux arguments de type `double` et retourneront un résulat également de type `double`. On se servira des formules suivantes :

$$distance(a,b) = \mid a - b \mid$$
$$maximum(a,b) = \frac{|a-b|+a+b}{2}$$

1. En fait, il faut aussi y être autorisé : soit la classe sollicitée se trouve dans notre répertoire, soit (comme la classe `Math`) elle est déclarée *publique* et est située à un endroit auquel on a le droit d'accéder.

2. L'indentation est le fait de décaler convenablement une ligne par rapport à la marge gauche d'un document afin d'accroître sa lisibilité. Cette indentation, si elle est optionnelle pour le compilateur, ne l'est pas pour l'œil d'un programmeur, qui peut aller jusqu'à refuser de lire un programme mal indenté !

$$minimum(a,b) = \frac{|a-b|-a-b}{-2}$$

Testez votre classe en lui adjoignant une méthode `main`(...) qui utilisent ces trois fonctions. ◇

Remarque : Pour nous, un fichier contient toujours une seule classe. C'est d'ailleurs elle qui donne son nom au fichier. Si on veut exécuter directement cette classe, elle doit contenir une méthode `main`(...). Si la classe ne sert qu'à proposer des méthodes utilitaires pour d'autres classes, elle n'a pas besoin de méthode `main`(...).

Exercice 2) Ajoutez à la classe `Numerik` les deux fonctions `randomInt`(...) et `randomDouble`(...) de *signatures* suivantes :

`randomInt`: `(int, int)` → `int`

`randomDouble`: `(double, double)` → `double`

Chacune de ces méthodes appliquée aux arguments (a, b) donnera comme résultat un nombre aléatoire du type concerné et compris dans l'intervalle $[a, b]$.
Pour ce faire, la classe `Math` contient une méthode prédéfinie `random()` qui s'utilise sans argument et qui retourne un nombre approché de type `double` choisi au hasard et appartenant à $[0, 1[$. ◇

Exercice 3) Dans un autre fichier, créez une classe `TP3Exo3`. Dans sa méthode `main`(...), elle devra engendrer, stocker et afficher deux réels aléatoires compris entre 10 et 20, puis afficher la distance entre ces 2 réels, leur minimum et leur maximum. Vous ferez appel aux méthodes de classe de votre classe `Numerik`. ◇

On attire votre attention sur le fait que les méthodes de classe se divisent en deux sortes :

- les *procédures* : elles sont précédées du mot-clé `void` : elles ne contiennent pas le mot-clé `return` vu qu'elles ne calculent pas de résultat. Leur appel (le nom suivi des arguments) tient lieu d'instruction (e.g. `main`(...)).
- les *fonctions* : elles sont précédées du nom d'un type ; le mot-clé `return` permet de retouner un résultat du type indiqué. Leur appel se fait en partie droite d'une affectation ou bien lors d'un affichage, d'un test (e.g. `Numerik.distance`(...)).

Exercice 4) Supposez que dans une classe se trouve une fonction d'entête :

```
static int foo (int x)
```

Que pensez-vous de l'affectation `foo(x)=2`? ◇

2 La classe `Console`

A notre niveau, nous appellerons *entrée* le fait de lire des informations au clavier et *sortie* le fait que l'ordinateur en affiche à l'écran. Les entrées/sorties ne sont pas aisées en JAVA (surtout les entrées), REALJ n'arrange rien, aussi allons-nous utiliser une classe `Console`[3] qui va nous

3. Ce n'est pas une classe prédéfinie JAVA, elle a été concoctée par vos enseignants.

faciliter la tâche. Voici un aperçu des méthodes statiques que cette classe propose (`x` dénote indifféremment un nombre ou une chaîne de caractères, `s` est une chaîne de caractères) :

`Console.print(x)`	affiche `x` -synonyme de `System.out.print(x)`-
`Console.println(x)`	affiche `x` et passe à la ligne -synonyme de `System.out.print(x)`-
`Console.readLine(s)`	fait surgir le message `s` puis retourne la chaîne lue au clavier
`Console.readDouble(s)`	fait surgir le message `s` puis retourne le `double` lu au clavier
`Console.readInt(s)`	fait surgir le message `s` puis retourne l'`int` lu au clavier

Pour pouvoir utiliser cette classe, deux choses sont nécessaires :

- ajouter en début de fichier, avant même le mot `class`, l'instruction :
```
import unsa.Console;
```
- afin que le programme termine[4], ajouter à la fin de votre fonction `main`(...) :
```
System.exit(0);
```

Les exercices suivants vous donneront l'occasion d'utiliser la classe `Console`, qui se trouve dans un ensemble de classes que nous avons nommé `unsa`, et auquel vous avez bien sûr accès :

Exercice 5) Ecrivez un programme qui demande trois entiers a (avec $a \neq 0$), b et c à l'utilisateur, qui les lit, puis qui affiche[5] :
```
L'équation ax^2+bx+c=0 est du second degré.
La somme de ses racines est : ...
Le produit de ses racines est : ...
```
Indication : Inutile de calculer les racines complexes, il existe des formules pour obtenir directement ces valeurs. Si besoin, demandez-les autour de vous. ◇

Exercice 6) Nous allons calculer la progression d'une somme d'argent placée pendant dix ans. Dans un premier temps, écrivez une classe dans laquelle vous demanderez à l'utilisateur le montant de la somme initiale `somme` et le taux d'intérêt annuel `taux` en pourcentage. Ensuite, vous calculerez le montant de la somme grâce à la méthode de classe fournie ci-après :

```
static double calculEpargne(double s, double t)
 {   return s * Math.pow(1 + t / 100, 10);
 }
```
◇

4. La classe `Console` fait appel à des mécanismes complexes d'écoute du clavier, qui restent actifs même après la dernière instruction du `main`(...). C'est pourquoi il est nécessaire de les *tuer* proprement avec un `System.exit(0)`, le code 0 signifiant au système d'exploitation : *"Tout s'est bien passé"*.

5. Dans le cas où l'utilisateur entre 0, ce programme aura un comportement étrange. Il faudra attendre le cours 5 avant de savoir *tester* si on est dans ce cas et le traiter à part.

3 Approfondissement

Il existe en JAVA un moyen de créer des constantes : il s'agit de donner un nom à des valeurs fixées de façon définitive. Vous avez déjà rencontré la constante `PI` de la classe `Math`.
Les noms des constantes sont traditionnellement écrits en majuscules. Une constante est définie dans une classe donnée, à l'extérieur de toute fonction (attention : même de la fonction `main`(...) !). Pour créer une constante, il suffit de faire précéder sa définition des mots-clés `static final`. On les utilise comme les méthodes de classe, en faisant précéder leur nom du nom de leur classe. C'est pour cela qu'on les appelle des *constantes de classe* ou *statiques*.

Exercice 7) Dans le programme précédent, on décide maintenant de demander seulement le montant de la somme initiale à l'utilisateur. Le taux d'intérêt sera une constante prédéfinie `TAUX` du programme, ainsi que le nombre d'années. Apportez quelques modifications à la classe précédente afin de fixer une fois pour toute le taux d'intérêt à 3% et le nombre d'années à 10. Votre programme devra être aménagé de telle sorte que la méthode `calculEpargne`(...) puisse s'écrire :

```
static double calculEpargne(double s)
 {   return  s * Math.pow(1 + TAUX / 100, ANS);
 }
```

Faites en sorte que votre résultat en francs soit un nombre entier. ◇

4 Abrégé de la classe `Math`

`Math.PI`	π
`Math.E`	e
`Math.sqrt(x)`	$\sqrt{x}$ $(x \geq 0)$
`Math.pow(x,n)`	x^n
`Math.sin(x)`	$sin(x)$ (x en radians)
`Math.cos(x)`	$cos(x)$
`Math.tan(x)`	$tan(x)$
`Math.exp(x)`	e^x
`Math.log(x)`	$ln(x)$ $(x > 0)$
`Math.ceil(x)`	plus petit entier $\geq x$
`Math.floor(x)`	plus grand entier $\leq x$
`Math.abs(x)`	$\mid x \mid$
`Math.random()`	renvoie un `double` $x \in [0, 1[$