Programmation Orientée Objet
M2 CCI 2008–2009
hoenen@icps.u-strasbg.fr

Université Louis Pasteur
Département d'informatique
10 décembre 2008

TP1 : premiers pas

1 Introduction à Java

Java, qu'est-ce que c'est ? Java est un langage de programmation orienté objet dont la syntaxe est proche du C/C++ concernant le nom des types et structures de contrôle. La particularité de Java est son indépendance par rapport aux architectures matérielles. Les compilateurs Java ne transforment pas un code source en langage machine (dédié à une architecture cible) mais en langage spécifique à la *plateforme* Java. Ce langage spécifique est appelé le *bytecode* Java et est complètement indépendant du matériel. Le bytecode est alors interprété par une *machine virtuelle* Java (ou JVM), qui elle est spécifique à l'architecture cible.

Comment faire un programme Java ? Le code source est écrit dans des fichiers textes portant l'extension `.java`. Par exemple, le code source contenu dans le fichier `source.java` est compilé par la commande `javac source.java`, ce qui a pour effet de générer le bytecode dans un fichier portant le même nom mais suivi de l'extension `.class`. Ensuite, pour interpréter le bytecode, on peut appeler la machine virtuelle par la commande `java source` (le nom du fichier contenant le bytecode mais sans son extension).

Quelques conventions pour l'écriture de code source Java

- le nom d'une classe commence par une lettre majuscule, et si il comprend plusieurs *mots*, ceux-ci sont attachés et chaque première lettre est majuscule : `MaClasseJava`
- le nom d'une méthode ou d'une variable suit la même règle que pour le nom q'une classe, à la différence près que la toute première lettre est minuscule : `maClasseJava`
- le nom d'une constante est donné en majuscules avec un `_` entre les différents *mots* : `MA_CONSTANTE`
- le fichier contenant la définition d'une classe doit avoir le même nom suivi de l'extension `.java` : `MaClasseJava.java`. On devrait avoir idéalement une classe par fichier et un fichier par classe.

Documentation de l'API en ligne La documentation de l'API officielle de Java se trouve sur le site http://java.sun.com/javase/6/docs/api/ (pour la version 1.6 de l'API). Prendre le temps de se familiariser avec cette documentation et ne pas hésiter à utiliser les fonctions de recherche du navigateur (ctrl+f) pour trouver une classe en particulier. La documentation de l'API est organisée comme suit :

- en haut à gauche : liste de tous les packages
- à gauche en dessous : liste de toutes les classes
- à droite : documentation de la classe sélectionnée

2 Premier programme en Java : HelloWorld

1. Coder, compiler le bytecode et lancer le programme HelloWorld ci dessous :

```
import java.lang.System;
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello world!");
  }
}
```

   À quoi sert la première ligne ? Peut-on s'en passer ?

2. Faire de même, mais cette fois avec une version Applet :

```
import java.awt.*;
import javax.swing.*;
public class MonApplet extends JApplet {
  public void paint(Graphics g) {
    g.drawString("Hello world!",20,20);
  }
}
```

Il faut ensuite le compiler pour obtenir du bytecode et créer un fichier `html` qui appelle l'Applet :

```
<html>
  <head>
    <title>Mon applet</title>
  </head>
  <body>
    Mon applet :
    <applet code="MonApplet.class" width="90" height="90"></applet>
  </body>
</html>
```

Le fichier `html` doit ensuite être ouvert avec l'application `appletviewer`. ou par le navigateur web.

3. Pareil pour la version suivante qui permet d'afficher le texte dans une fenêtre :

```
import javax.swing.JOptionPane;
public class HelloWorldDiag {
  public static void main(String[] args) {
    JOptionPane.showMessageDialog(null, "Hello world!");
    System.exit(0);
  }
}
```

3 Arguments en ligne de commande et types primitifs

Tout programme Java classique doit contenir une *classe* qui définit une *méthode* avec le profil suivant :

`public static void main(String[] args)`

C'est l'équivalent de la fonction `main` en C, c'est elle qui est appelée lorsque le bytecode est interprété par la machine virtuelle. Comme en C, la fonction prend en paramètres les arguments passés à la ligne de commande. Ces arguments sont placés dans un tableau de chaînes de caractères appelé `args` [1]. En Java, un tableau est un objet qui possède un attribut `length` qui donne la taille du tableau.

1. Reprendre le code source du programme HelloWorld, et le modifier pour afficher le nombre et la liste des arguments passés en ligne de commande.
2. Quels sont les types primitifs qui existent en Java et à quoi servent les classes *wrapper*.
3. Modifier à nouveau HelloWorld pour qu'il affiche son message n fois, n étant un argument passé en ligne de commande.
4. Écrire un programme qui calcule la moyenne (décimale) d'un certain nombre d'entiers passés en argument de la ligne de commande.

[1] attention, contrairement au C, ici `args[0]` contient le premier argument

4 Premiers *objets* : les dés

En Java, la définition d'une classe par se fait par le mot-clef `class` et pour la définition d'une méthode, le nom est précédé d'un type de retour et suivi d'une liste de paramètres entre parenthèses. Exemple :

```
class MaClasse {
  ...
}
```

```
<type> maMethode(<type1> arg1, ...) {
  ...
}
```

Le corps d'une classe peut contenir des attributs et des méthodes (constructeurs, accesseurs, etc...). Chaque déclaration peut être précédée de modificateurs d'accès pour contrôler la visibilité des classes, attributs et méthodes. Pour l'instant on se contentera des modificateurs `public` (visible par tout le monde) et `private` (visible uniquement à l'intérieur de la classe).
Un constructeur est une méthode spécifique qui est appelée à la création de l'objet. Contrairement aux méthodes traditionnelles, le constructeur ne possède pas de type de retour et a le même com que la classe qu'il instancie. Il peut avoir un, plusieurs ou aucun paramètre. Un objet est une instance d'une classe. Pour créer un objet, on commence par déclarer une référence comme une variable dont le type correspond au nom de la classe à instancier. Puis on créé l'objet avec l'opérateur `new` qui va faire appel au constructeur de la classe. Exemple :

```
public MaClasse(<type1> arg1) {
  ...
}
```

```
MaClasse monObjet;
monObjet = new MaClasse(...);
...
```

Pour accéder à un attribut d'un objet ou invoquer une méthode d'un objet, on utilise le nom de la référence de l'objet suivi de l'opérateur `.` (point) et du nom de l'attribut ou de la méthode. Exemple :

```
<type de unAttribut> var1 = monObjet.unAttribut;
<type de maMethode> var2 = monObjet.maMethode(...);
```

On veut écrire un petit programme qui réalise le lancé d'un certain nombre de dés d'un type donné.

1. Écrire la classe `De` qui modélise un dé possédant un certain nombre de faces `nFaces` (numérotées de 1 à `nFaces`). Ajouter un constructeur permettant d'initialiser l'objet.
2. Ajouter une méthode qui effectue un lancé aléatoire pour un dé et en renvoie le résultat. `Math.random()` renvoie un `double` positif compris entre 0.0 (inclus) et 1.0 (exclus).
3. Écrire une classe `LanceDe` qui contient une méthode `main` qui affiche le résultat de 2 lancés d'un dés à 6 faces ainsi que la somme de ces résultats.
4. Faire de même mais cette fois le nombre de lancés et le type du dés sont deux arguments passés en ligne de commande.
5. Modifier légèrement la classe `De` pour que celle-ci permette de différencier les dés normaux et les dés *pipés* (qui font toujours des scores au dessus de la moyenne : par exemple, un dé à 6 faces pipé ne fera que des résultats compris entre 4 et 6). On ne veut pas modifier le constructeur, donc il faudra rajouter une méthode *pipeDe* qui transforme un dé classique en dé pipé.
6. Prendre en compte un troisième argument en ligne de commande qui dit si oui ou non le dé dont on veut faire un certain nombre de lancés est pipé ou classique.

5 Références et portée des objets

En Java on manipule toujours les objets par référence (par pointeur). Du coup, lorsque l'on passe un objet en argument d'une méthode, c'est la référence de l'objet qui est transmise : l'objet peut donc être modifié par toutes les opérations effectuées dans la méthode. De même, un même objet peut être référencé plusieurs fois. Le cycle de vie d'un objet est le suivant : un objet est créé (donc alloué en mémoire) à

chaque appel à l'opérateur `new`, et il est détruit (déallocation de la mémoire qui lui était attribué) lorsque le *ramasse-miette* (ou *garbage collector*) se rend compte qu'il n'est plus référencé nulle part.

Les attributs et les méthodes d'une classe sont généralement des variables d'instance, elles ne peuvent pas être référencées en dehors d'une instance de la classe (un objet). Il existe un moyen de rendre accessible certains attributs et méthodes en dehors de toute instance : on les définit pour cela avec le mot-clef `static`. Une variable ou une méthode déclarées `static` sera accessible accesible directement en appelant le nom de la classe suivi d'un point et du nom de la variable/méthode. Exemple :

```
class MaClasse {
  public static <type1> maMethode(...);
  public static <type2> maVariable;
}
```

```
<type1> var1;
<type2> var2;
var1 = MaClasse.maMethode(...);
var2 = MaClasse.maVariable;
```

On veut écrire un petit programme pour manipuler des couleurs (exercice de Benjamin Schwartz).

1. Écrire une classe `Couleur` qui contiendra trois attributs entiers r, g et b (pour chacun des canaux usuels *rouge*, *vert* et *bleu* à valeur entre 0 et 255).
2. Ajouter des méthodes d'accès à chaque canal ainsi qu'une méthode `toString()` pour convertir une couleur en un objet `String`, ce qui facilitera l'affichage des canaux d'une couleur (par exemple sous la forme `#(r,g,b)`).
3. Écrire un programme `TestCouleurs` dans lequel on définit des objets de type Couleur pour les couleurs usuelles : le rouge (255,0,0), le bleu, le vert, le noir, et le blanc.
4. Ajouter à la classe `Couleur` une méthode `plus(char composante, int valeur)` qui augmente d'une certaine valeur l'intensité d'une des composantes de la couleur, puis une méthode statique `static moins(Couleur C, char composante, int valeur)` qui diminue l'intensité d'une certaine valeur l'une des composantes de la couleur passée en argument.
5. En supposant que le programme `TestCouleurs` contienne les lignes suivantes :

```
Couleur c;
c = new Couleur(255,0,0);
Couleur d = c;
d = new Couleur(0,0,255);
d = new Couleur(0,255,0);
d.plus('r',16);
Couleur.moins(c,'r',16);
d = null;
```

   Que valent les couleurs de `c` et de `d` à chaque ligne de ce programme ? Quelles sont les lignes où le ramasse-miette pourra rentrer en action ?
6. Modifier la classe `Couleur` pour que les couleurs usuelles soient accessibles directement sans avoir besoin de les redéfinir. Après ces modifications, le code du programme `TestCouleurs` devra ressembler à :

```
Couleur c;
c = Couleur.rouge;
System.out.println(c);
c = Couleur.bleu;
System.out.println(c);
```