Fiche 1 : syntaxe de Java et programmation impérative

IPO – Introduction à la programmation objet

Thibaut Balabonski @ Université Paris-Saclay
http://www.lri.fr/~blsk/IPO/

V1, automne 2022

Structure minimale. Un *programme* Java est un texte, écrit dans un fichier d'extension `.java`. Voici la structure minimale pour un programme `hello`, à écrire dans un fichier nommé `hello.java`, où les points de suspension doivent être remplacés par le code à exécuter.

```
public class hello {
  public static void main(String[] args) {
    ...
  }
}
```

On verra plus tard ce que signifient tous ces mots, vous pouvez les considérer comme une formule magique en attendant. Note : le nom du programme, écrit après **`public class`**, doit correspondre exactement au nom du fichier, majuscules comprises.

On applique le *compilateur* Java à un tel programme avec la commande

```
javac hello.java
```

(à faire dans un terminal). Ceci produit un nouveau fichier `hello.class`, qui est une traduction du programme dans le format de la *machine virtuelle* Java (JVM). On peut ensuite exécuter le programme avec la commande

```
java hello
```

(toujours dans le terminal). Pour que notre programme `hello` affiche `Hello, world!` lors de son exécution, il suffit de placer à la place des points de suspension la ligne

```
    System.out.println("Hello, world!");
```

Types, données et opérations. En java, on manipule des données qui peuvent être :

- des *valeurs de base*, comme des nombres entiers (**`int`**), des nombres décimaux (**`double`**), des booléens (**`boolean`**), des caractères (**`char`**) ; et
- des *valeurs complexes*, comme des chaînes de caractères (`String`), des tableaux (**`int`**`[]`), ou d'autres objets à définir soi-même.

Chaque type de données est associé à ses opérations. En voici quelques unes.

- Pour les entiers (**`int`**) : opérations arithmétiques `+`, `-`, `*`, `/`, `%`, comparaisons `==`, `!=`, `<`, `<=`, `>`, `>=`.
- Pour les booléens : conjonction `&&` (« et »), disjonction `||` (« ou »), négation `!` (« non »).
- Pour les chaînes de caractères : l'opérateur `+` est la concaténation, c'est-à-dire la mise bout à bout de deux chaînes.

Variables. Une variable est un *nom*, faisant référence à une *valeur stockée en mémoire*. On déclare une variable en donnant un nom et le type de la valeur à stocker, et éventuellement une valeur initiale. On peut déclarer plusieurs variables d'un même type sur une seule ligne en les séparant par des virgules.

```
int x = 3;
String s1, s2;
```

On peut consulter la valeur d'une variable en utilisant simplement son nom, ou la modifier à l'aide de l'opérateur `=` d'*affectation*.

```
y = x + 1;
```

Attention à ne pas confondre :

- l'opérateur `==` qui teste l'égalité de deux valeurs (et produit un booléen),
- l'opérateur `=` qui modifie la valeur d'une variable (et ne produit rien).

Structures de contrôle. En programmation impérative, le corps d'un programme est une séquence d'instructions, à exécuter dans l'ordre. Les *structures de contrôle* permettent d'organiser la séquence.

Le *branchement conditionnel* sélectionne un bloc (c'est-à-dire une séquence d'instructions) à exécuter, en fonction du résultat d'un test. Le deuxième bloc, introduit par **else**, est facultatif.

```
if (score >= 100) {
  System.out.println("Gagné");
} else {
  System.out.println("Perdu");
}
```

La *boucle conditionnelle* (« **while** ») répète l'exécution d'un bloc tant qu'une condition est vérifiée.

```
int x = 1;
while (x < n) {
  x = 2*x;
}
System.out.println(x);
```

La *boucle inconditionnelle* (« **for** ») répète l'exécution d'un bloc pour chaque valeur d'un intervalle entier.

```
for (int i = 0; i < n; i++) {
  System.out.println(i * i);
}
```

Fonctions. Les fonctions permettent d'*isoler* un fragment de programme en lui donnant un nom, que l'on s'efforce de choisir suffisamment explicite pour aider la compréhension du programme. La définition d'une fonction contient, dans l'ordre :

1. le type de la valeur qui sera renvoyée à la fin de l'exécution de la fonction (ou **void** si aucune valeur renvoyée),
2. le nom de la fonction,
3. la liste des arguments attentus, chacun donné par son nom et son type,
4. le bloc d'instructions à exécuter.

Les trois premiers éléments forment la *déclaration* de la fonction, le quatrième élément est son *corps*. En java, on ajoutera pour l'instant les deux mots-clés **public static** devant la déclaration de chaque fonction (dans le contexte de java, ceci ne s'appelle pas une « fonction » mais une « méthode », nous y reviendrons plus tard).

```
public static int binaryLog(int n) {
  int x = 1;
  while (x < n) {
    x = 2*x;
  }
  return x;
}
```

On *appelle* une fonction en fournissant des valeurs concrètes pour chacun des arguments.

```
int a = binaryLog(1024);
```

Note : un argument entier est passé *par valeur*.

Tableaux. Un tableau permet de stocker un ensemble de données de même type, et d'accéder à chacune par son *indice*. La première donnée a l'indice 0. On note avec des crochets `[]` le type d'un tableau.

```
int[] tab;
```

On crée un tableau avec le mot-clé **new**. Il faut alors préciser le nombre d'éléments que contiendra le tableau.

```
tab = new int[12];
```

On accède à un élément d'un tableau, en lecture comme en écriture, en précisant son indice.

```
tab[2] = tab[0] + tab[1];
```

Les tableaux (et d'autres structures de données similaires) dispose d'un mécanisme d'*itération* (« **for** each »), consistant à énumérer tous les éléments et répéter un même bloc d'instruction pour chacun.

```
  for(int x: tab) { // tab contient des entiers
    System.out.println(x*x);
  }
```

En java, passer un tableau en argument à une fonction se fait uniquement (et implicitement) *par référence*. Une fonction s'appliquant à un tableau est donc susceptible de le modifier définitivement.

Décodage des arguments de la ligne de commande. La fonction principale `main` prend obligatoirement en argument un tableau de chaînes de caractères (type `String[]`), contenant les arguments donnés au programme sur la ligne de commande. Pour « décoder » ces arguments lorsqu'ils sont censés représenter des nombres, on utilise des fonctions notées `Interger.parseInt` ou `Double.parseDouble`.

```
  public static void main(String[] args) {
    int x = Integer.parseInt(args[0]);
    double d = Double.parseDouble(args[1]);
    System.out.println(x + d);
  }
```

Pour passer des arguments sur la ligne de commande, on les place après le nom du programme.

```
  java hello 42 1.0001
```

Un programme. Ce programme doit être donné dans un fichier `mandelbrot.java`.

```
public class mandelbrot {
    public static boolean escapeTime(double cx, double cy) {
        double zx = 0;
        double zy = 0;
        for (int i = 0; i < 1000; i++) {
            double z = zx * zx - zy * zy;
            zy = 2 * zx * zy + cy;
            zx = z + cx;
            if (zx*zx + zy*zy > 4) { return false; }
        }
        return true;
    }

    public static void printMandelbrot(double x0, double y0, double radius, int res) {
        double xMin = x0 - radius;
        double yMin = y0 - radius;
        double pixelSize = radius / res;
        for (int y = 0; y < 2*res+1; y++) {
            for (int x = 0; x < 2*res+1; x++) {
                if (escapeTime(xMin + x*pixelSize, yMin + y*pixelSize)) {
                    System.out.print("@@");
                } else {
                    System.out.print("..");
                }
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        double x0 = Double.parseDouble(args[0]);
        double y0 = Double.parseDouble(args[1]);
        double radius = Double.parseDouble(args[2]);
        int res = Integer.parseInt(args[3]);
        printMandelbrot(x0, y0, radius, res);
    }
}
```

On peut l'exécuter par exemple avec

```
  java mandelbrot -.5 0 1 20
```