Travaux Pratiques de Programmation en Java. Séance 02 : structures de contrôle de flux, itérations et sous-fonctions

F.Baudoin, T. de Broca, B. Hugueney, M. Manceny

<2012-10-03 Wed>

Contents

1 Objectifs pédagogiques

Au cours de cette séance, vous aurez l'occasion d'acquérir les compétences suivantes :

- Conception d'un algorithme simple **en pseudo-code** avec des structures contrôle de flux :
  - alternatives
    - if
    - switch
  - itératives
- Implémentation d'un algorithme en Java avec des structures contrôle de flux :
  - alternatives
    - if
    - switch
  - itératives
- Implémentation en Java d'un programme Java composé de plusieurs sous-fonctions
- Implémentation en Java d'un menu permettant de choisir l'exécution le code à lancer

2 Rappels sur les structures de contrôle de flux

Lors de l'exécution d'un programme, le processeur exécute un flux d'instructions. Afin de pouvoir traiter différents cas de figure et répéter l'exécution de certaines instructions autant de fois que nécessaire sans avoir à répéter celles-ci dans le code source (i.e. le programme Java que vous écrivez), on dispose de structures de contrôle de flux.

Même si l'ordinateur n'en tient pas compte, il est **impératif** d'indenter le code en conséquence, en décalant les instructions d'un bloc (entre les accolades `{}`) d'une tabulation vers la droite. En cas de bloc impliqués (tests à l'intérieur d'une boucle ou réciproquement), on décalera autant de fois que nécessaire pour indiquer visuellement toutes les structures de contrôle.

2.1 Exécution conditionnelle

2.1.1 if

Il est possible de coder une alternative qui déclenchera l'exécution conditionnelle d'un bloc d'instructions **si** (`if`) une condition (i.e. résultat d'un test) est vérifiée. Optionnellemment, **dans le cas contraire** (`else`) un autre bloc d'instruction peut être exécuté.

```
if (TEST) {
//   instructions si le test est vrai;
} else {
 //  instructions si le test est faux;
}
```

2.1.2 switch

L'instruction `switch` permet d'éviter des `if` imbriqués en effectuant plusieurs tests de valeurs sur le contenu d'une même variable. La variable doit être obligatoirement de type numérique. Les deux codes suivantes sont équivalentes.

```
switch (variable) {
case VALEUR_1:
    /*Liste d'instructions a executer lorsque variable vaut VALEUR_1
     */
    break;// on ne veut pas executer d'autres instructions dans ce cas
case VALEUR_2:
    /*Liste d'instructions a executer lorsque variable vaut VALEUR_2
     */
    break;// on ne veut pas executer d'autres instructions dans ce cas
case VALEUR_3:
    /*Liste d'instructions a executer lorsque variable vaut VALEUR_3
     */
    break;// on ne veut pas executer d'autres instructions dans ce cas
default:
    /* Liste d'instructions a executer lorsque variable ne vaut
       ni VALEUR_1 ni VALEUR_2 ni VALEUR_3
     */
}
```

et

```
if (variable == VALEUR_1) {
    /*Liste d'instructions a executer lorsque variable vaut VALEUR_1
     */
} else if (variable == VALEUR_2) {
    /*Liste d'instructions a executer lorsque variable vaut VALEUR_2
     */
} else if (variable == VALEUR_3) {
    /*Liste d'instructions a executer lorsque variable vaut VALEUR_3
     */
} else {
    /* Liste d'instructions a executer lorsque variable ne vaut
       ni VALEUR_1 ni VALEUR_2 ni VALEUR_3
    */
}
```

2.2 Itérations (boucles)

2.2.1 for

L'instruction `for` permet d'exécuter plusieurs fois la même série d'instructions jusqu'a ce qu'une condition ne soit plus réalisée. La boucle `for` s'utilise lorsque le nombre d'itérations est déjà connu avant d'entrer dans la boucle.

```
for (/*initialisation avant l'iteration*/;
                                        /*condition pour continuer a iterer*/;
                                        /* modification de fin d'iteration*/) {
    /* liste d'instructions a executer en boucle
       (a chaque iteration)
    */
}
```

Le plus souvent, on utilisera une boucle `for` lorsque le nombre d'itérations est déjà connu à l'avance.

```
// i doit etre deja declare
  for (i=0; i<5; ++i) { // ++i est equivalent a i= i+1 ou i+= 1
     System.out.println(i);
  }
```

affiche en sortie :

```
0
```

```
1
2
3
4
```

Notez que le nombre d'itérations (ici `5`) pourrait être le résultat de l'évaluation d'une expression ou lu dans une variable.

2.2.2 while

L'instruction `while` permet de répéter un jeu d'instructions tant que la condition de boucle est vraie. Deux syntaxes sont possibles:

```
while(/*condition*/){
    /* liste d'instructions
       ...
    */
}
```

ou, lorsque la première itération doit être de toutes façons effectuée (avant de faire le premier test) :

```
do{
    /* liste d'instructions
       ...
    */
} while(/*condition*/);
```

2.3 Fonctions

La fonction `main` est appelée **fonction principale** et sera appelée directement lorsque vous exécutez votre programme. Lorsque vous utilisez l'instruction `System.out.println` par exemple, vous appelez en réalité d'autres fonctions (des **sous-fonctions**) pour réaliser une tâche spécifique. Il est possible de définir ses propres sous-fonctions de la même manière qu'on écrit la fonction principale. L'exemple ci-dessous crée deux sous-fonctions qui sont appelées par la fonction principale. Notez que comme ces fonctions n'utilisent pas d'arguments, les parenthèses `()` qui suivent le nom des fonctions sont vides (pas de `String[] args` contrairement à `main`).

```
// a l interieur d'une classe, par exemple TP2
  public static void sousFonction1(){
```

```
    System.out.print("Hello ");
  }

  public static void sousFonction2(){
    System.out.println("World!");
  }

 public static void main(String[] args){
    sousFonction1();
    sousFonction2();
  }
```

Au cours de cette séance de TP, on fera en sorte que la fonction principale `main` appelle une sous-fonction spécifique à chaque exercice.

3 Exercices

3.1 Menu (fonction principale et sous-fonctions)

Écrire un programme qui affiche un menu demandant à l'utilisateur quel exercice il désire faire. Lorsque l'utilisateur choisira un numéro, le programme appellera la sous-fonction correspondant à l'exercice. (Utilisez les instructions `System.out.println` et `nextInt` de la classe **Scanner**.)

Quel exercice ?

1. Regles
2. Puissance
3. Multiplications
4. Nombres premiers
5. Formes

3.2 Règle graduée

Écrire une fonction qui demande une longueur et affiche une règle de cette longeur avec des tirets:

```
Longueur ? 53

----------------------------------------------------
```

Modifier votre programme pour qu'il affiche des graduations:

```
Longueur ? 53
Intervalle ? 10

|---------|---------|---------|---------|---------|---
```

NB en Java, l'opérateur modulo s'écrit `%`. Ainsi: `13 %10` retourne la résultat `3`.

3.3 Puissances

Écrire une fonction qui, étant donnés deux nombres a et n, calcule a^n.

Bonus Combien de multiplications devez vous faire ? Pensez-vous qu'il soit possible d'en faire moins ? (Par exemple, si n est pair ?)

3.4 Table de multiplications

Écrire une fonction affichant une table de multiplications sous la forme:

```
0  0   0   0   0   0   0   0   0   0   0
0  1   2   3   4   5   6   7   8   9   10
0  2   4   6   8   10  12  14  16  18  20
0  3   6   9   12  15  18  21  24  27  30
0  4   8   12  16  20  24  28  32  36  40
0  5   10  15  20  25  30  35  40  45  50
0  6   12  18  24  30  36  42  48  54  60
0  7   14  21  28  35  42  49  56  63  70
0  8   16  24  32  40  48  56  64  72  80
0  9   18  27  36  45  54  63  72  81  90
0  10  20  30  40  50  60  70  80  90  100
```

3.5 Nombre premier

Écrire une fonction qui teste si n est un nombre premier et renvoie 1 si c'est la cas et 0 sinon.

Bonus Cette fonction pourrait-elle renvoyer un résultat d'un autre type ?

3.6 Formes géométriques en mode 'caractères'

3.6.1 Rectangle

Écrire une fonction qui demande à l'utilisateur un entier n, puis affiche avec des étoiles un rectangle de côté n:

```
Cote ? 4
****
****
****
****
```

3.6.2 Triangle dans un sens

Modifier votre fonction pour qu'elle affiche un triangle:

```
Cote ? 4
*
**
***
****
```

3.6.3 Triangle dans l'autre sens

Même question avec la pointe dans l'autre sens:

```
Cote ? 4
   *
  **
 ***
****
```

3.6.4 Forme géométrique 'à la carte' (mais avec un menu)

Enfin, écrivez une fonction complète qui commence par demander à l'utilisateur ce qu'il veut dessiner puis dessine la bonne forme (utilisez la structure de contrôle de flux `switch`).

3.6.5 Bonus : Cercle

Écrire une fonction qui demande à l'utilisateur un entier n, puis affiche avec des étoiles un cerle de rayon n:

```
Rayon ? 6
       *
    *     *
   *       *
  *         *
 *           *
 *           *
 *           *
 *           *
 *           *
  *         *
   *       *
    *     *
       *
```

Remarque : Ce n'est pas vraiment un cercle parce que les caractères de la console ne sont pas des 'pixels' carrés.