Introduction à la Programmation 1

Séance 1 de cours/TD

Université Paris-Diderot

Objectifs:

- Utiliser le langage JAVA comme une calculatrice.
- Identifier et donner un sens aux différentes constructions du langage (déclaration et utilisation des variables, boucles for, conditionnelles, procédures et fonctions).
- Apporter une modification mineure à un programme existant.

```
1        int stopSecs = (stopHour * 60 + stopMin) * 60;
2        int startSecs = (startHour * 60 + startMin) * 60;
3        int numberOfSecs = stopSecs - startSecs;
4        int alertCode = 0;
5        for (int i = 0; i < numberOfSecs; i++) {
6            waitUntilNextSecond ();
7            if (numberOfSecs - i < 30)
8                alertCode = 1;
9            drawInt (numberOfSecs - i, alertCode);
10       }
```

Listing 1 – Que fait ce programme ?

1 Le langage des expressions arithmétiques

Sous-langage des expressions arithmétiques [COURS]

- Le type int est l'ensemble des valeurs entières comprises entre -2147483648 ($= -2^{31}$) et 2147483647 ($= 2^{31} - 1$).
- Une **expression arithmétique de type** int peut être :
  (a) une constante entière (0, 1, 2, -1, -2, -2147483648, 2147483647 ...) ;
  (b) deux expressions séparées par une opération arithmétique binaire ($1 + 2$, $1 + 2 * 3 / 4$, ...) ;
  (c) une expression entourée de parenthèses ($(1 + 2)$, $(1 + 2 * 3 / 4)$, ...) ;
  (d) une expression précédée du signe moins (-2, $-(1 + 2)$, ...).
- Les opérateurs ont des priorités relatives, par exemple
  $\{*, /, \ \% \} \preceq \{+,-\}$
  où $\preceq$ signifie "être prioritaire sur".
- Les opérateurs binaires précédents sont associatifs à gauche.
- Par exemple, l'évaluation de l'expression $1 + 2 * 3 - 4$ se décompose en (i) $2 * 3$ donne 6 ; (ii) $1 + 6$ en 7 et (iii) $7 - 4$ donne 3.
- Les opérations ($+$, $-$, $/$, $*$, ...) ont le sens usuel *tant que l'on reste entre les bornes du type* int. Dans le type int, la division / est une *division entière*. Par exemple, $31/7$ vaut 4.
- L'opérateur % désigne le *modulo* : a%b est le reste dans la division entière de a par b. Par exemple, 31%7 vaut 3 car $31 = 7 \times 4 + 3$.

Exercice 1 (Java comme une calculatrice, ⋆)
Prévoir l'évaluation des expressions arithmétiques suivantes :

```
6 * 7 + 3
6 * (7 + 3)
45 / 7
3 * 7 / 4
(3 * 7) / 4
(45 / 7) * 7 + 45 % 7
(1 + 2 - 3 + 4 - 5 + 6 - 7 + 8 - 9 + 10 - 11 + 12 - 13) / (1 - 2 + 3 - 4
 + 5 - 6 + 7 - 8 + 9 - 10 + 11 - 12 + 13)
```

□

2 Utilisation des variables

Déclaration et utilisation des variables [Cours]

— Une **variable** a un *nom* et contient une *valeur*, c'est-à-dire le résultat d'un calcul.
— On déclare une variable x de type int qui contient initialement le résultat du calcul 6 ∗ 7 ainsi :

```
int x = 6 * 7;
```

— Dans l'exemple précédent, la valeur de la variable x est 42.
— Exemples de noms de variable : x, y, foo, foobar42...
— On peut utiliser la valeur d'une variable dans une expression en faisant référence à son nom. Ainsi, l'expression "x + 1" vaut 43 si x vaut 42.
— On peut changer la valeur d'une variable qui a été déclarée auparavant en lui *affectant* le résultat d'un nouveau calcul. L'opérateur d'affectation est « = » :

```
x = 2 * 10;
```

Instructions [Cours]

— Une expression calcule une valeur tandis qu'une instruction a un effet sur la machine.
— On dit qu'une machine exécute une instruction.
— On peut utiliser une **procédure** en écrivant son nom suivi d'un ou plusieurs paramètres séparés par des virgules et entourés de parenthèses. Par exemple :

```
printInt(1 + 2);
```

affiche 3 à l'écran. Un autre exemple :

```
putPixel(0, 0, 255, 255, 255);
```

affiche un pixel blanc en position (0, 0) d'une image.
— Une procédure est définie en termes d'une suite d'instructions (les détails viendront plus tard).
— On peut écrire des suites d'instructions qui sont exécutées dans l'ordre par la machine.

Exercice 2 (Nommer, ⋆)
Quelle est la valeur des variables, à la suite des instructions suivantes.

```
int x = 6 * 7;
int y = x + x;
```

□

Exercice 3 (Affectations, ⋆⋆)
Quelle est la valeur des variables, à la suite des instructions suivantes.

```
int x = 1;
int y = 4;
x = y + 2;
```

□

Exercice 4 (Affectations, ⋆⋆)
Quelle est la valeur des variables, à la suite des instructions suivantes.

```
int x = 1;
int y = x - 1;
x = 2;
```

□

Exercice 5 (Affectations, ⋆⋆)
Quelle est la valeur des variables, à la suite des instructions suivantes.

```
int x = 1;
x = x - 1;
```

□

3 Fonctions

Nommer un calcul comme une fonction [Cours]

— La fonction qui attend en paramètre une valeur x de type int et calcule une valeur de type int qui est le triple de x s'écrit :

```
public static int triple (int x) {
  return (x * 3);
}
```

— On peut utiliser une fonction en écrivant son nom suivi de son paramètre entouré de parenthèses, comme par exemple dans l'expression :

```
  triple (2) * 6
```

— Dans cet exemple, triple (2) vaut 6 donc l'expression vaut 36.
— Voici des exemples de noms de fonction : triple , add3, fooBar, ...
— Une fonction peut prendre plusieurs paramètres.
— On peut utiliser des fonctions définies dans des bibliothèques de fonctions.

Exercice 6 (Utiliser une fonction, ⋆)
Étant donnée la fonction suivante, que vaut l'expression twice(3) *?*

```
public static int twice (int x) {
    return (2 * x);
}
```

□

Exercice 7 (Utiliser une fonction, ⋆⋆)
On considère la fonction suivante.

```
1  public static int cube (int x) {
2      return (x * x * x);
3  }
```

Quelle est la valeur de la variable a *à la suite des instructions suivantes ?*

```
1 int b = 5;
2 int a = 3;
3 a = b - a;
4 a = cube(a);
```

□

Exercice 8 (Écrire une fonction, ⋆⋆)
Voici une fonction

```
1  public static int f(int x) {
2      return (x * x + 5);
3  }
```

Quel est son nom ? Que calcule-t-elle ? Modifier son corps pour qu'elle calcule la division entière par 3. Modifier son nom pour qu'elle se nomme third.

□

Exercice 9 (Écrire une fonction, ⋆⋆)
Écrire des fonctions effectuant les calculs suivants :

1. *La puissance 5 d'un entier donné en paramètre*
2. *Le produit de deux entiers donnés en paramètre moins leur somme*
3. *Le produit de trois entiers donnés en paramètre au carré*

□

4 Conditionnelle

Instruction conditionnelle [Cours]

— Une instruction conditionnelle permet d'exécuter des instructions en fonction d'une condition.
— On écrit par exemple

```
1  int x, y;
2  x = ...;
3  y = ...;
4  if (x <= 0) {
5      y = x;
6  } else {
7      y = -x;
8  }
9
```

pour affecter la valeur de x à y si x <= 0 ou pour lui affecter la valeur −x dans le cas contraire.

— Les conditions peuvent par exemple être des comparaisons entre deux expressions de type int (e1 == e2, e1 != e2, e1 < e2, e1 <= e2, e1 > e2, e1 >= e2).

Exercice 10 (Le max, ⋆)
À la suite des instructions ci-dessous, quelle est la valeur de la variable max *?*

```
int x = 3;
int y = 4;
int max = 0;
if (x > y) {
    max = x;
} else {
  max = y;
}
```

Transformer la suite d'instructions pour calculer le minimum de x *et* y *et le mettre dans une variable* min.

□

Exercice 11 (Différents tests, ⋆⋆)
Quelle est la valeur des variables a, b *et* c *après la suite d'instructions suivante :*

```
int a = 2;
int b = a * a + 3;
int c = b - a;
if (c == a) {
    a = 1;
} else {
    a = a + 3;
}
if ((b + c) < a) {
    b = 2;
} else {
    b = 4;
}
if (b != c * c){
    c = 12;
} else {
    c = -6;
}
```

□

5 Boucles

Boucles [Cours]

— Une boucle permet de répéter plusieurs fois les mêmes instructions.
— Le numéro de l'itération est disponible dans une variable qui s'appelle le *compteur de boucle*.
— La boucle suivante affiche les entiers de 0 à 9 :

```
for(int i = 0; i <= 9; i++) {
    printInt(i);
}
```

— L'*entête* est formé du mot clé `for` suivi d'une initialisation, d'une condition, et d'une incrémentation, séparées par des points-virgules et entourées de parenthèses ;
— L'*initialisation* `int i = 0` définit le type et la valeur initiale du compteur de boucle ;
— La *condition de boucle* `i <= 9` définit sous quelle condition l'exécution de la boucle continue ;
— L'*instruction d'incrémentation* `i++`. Cette instruction est équivalente ici à l'instruction $i = i + 1$.
— Le *corps de la boucle*, entre accolades, est exécuté à chaque itération de la boucle.
— La même boucle aurait pu être écrite comme suit :

```
for(int i = 0; i < 10; i++) {
    printInt(i);
}
```

Exercice 12 (Afficher des suites d'entiers, ⋆)

1. *Écrivez une boucle qui affiche les 100 premiers entiers, en commençant à 0. Quel est le dernier entier affiché ?*
2. *Écrivez une boucle qui affiche les 50 premiers entiers pairs, en commençant à 0. Quel est le dernier entier affiché ?*
3. *Écrivez une boucle qui affiche 1000 fois le nombre 3.*

□

6 Fonctions et procédures utilisées

Liste des fonctions [Cours]

```
/*
 * Affiche un entier sur l'écran.
 * x est l'entier à afficher.
 */
public static void printInt (int x) {
    System.out.println(x);
}
```

7 Do it yourself

Exercice 13 (Valeurs d'expression entières, ⋆)
Donner la valeur des trois expressions suivantes : $3 + 5 / 3$ $4 * 1 / 4$ $2 / 3 * 3 - 2$ □

Exercice 14 (Modulo 9, ⋆⋆)
Donner la valeur des expressions suivantes :
$18 \% 9$ $81 + 18 \% 9$ $(81 + 18) \% 9$

□

Exercice 15 (Valeur absolue, ⋆)
En vous inspirant de l'exercice 10, donner une suite d'instructions permettant de calculer la valeur absolue d'une variable. □

Exercice 16 (Utiliser une fonction, ⋆)
On considère la fonction suivante.

```
  public static int sumsquare (int x, int y){
    return (x * x + y * y);
  }
```

Quelle est la valeur des variables a, b *et* c *à la suite des instructions suivantes :*

```
int a = sumsquare(2, 3);
int b = sumsquare(3, 1);
int c = sumsquare(4, 3);
```

□

Exercice 17 (Boucles simples, ⋆⋆)

1. *Quel est l'affichage produit par la suite d'instructions suivante :*

```
for(int i = 0; i < 5; i++){
    printInt(8);
}
```

2. *Quelle est la valeur de* x *après la suite d'instructions suivante :*

```
int x = 0;
for(int i = 0; i < 5; i++){
    x = x + 8;
}
```

3. *Quelle est la valeur de* x *après la suite d'instructions suivante :*

```
int x = 0;
for(int i = 0; i < 5; i++){
    x = 10 * x + 8;
}
```

□

Exercice 18 (Des entiers qui s'ajoutent, ⋆⋆/⋆⋆⋆)
Quelle est la valeur de somme *après la suite d'instructions suivante :*

```
int i = 0;
int somme = 0;
somme = somme + i;
i = i + 1;
somme = somme + i;
i = i + 1;
somme = somme + i;
i = i + 1;
somme = somme + i;
i = i + 1;
somme = somme + i;
i = i + 1;
somme = somme + i;
i = i + 1;
somme = somme + i;
i = i + 1;
```

Si on veut adapter le code ci-dessus pour aller non plus jusqu'à 5 mais jusqu'à 100, 1000 ou plus, on a un problème : on obtiendrait un programme beaucoup trop long !

On peut remplacer la longue liste d'instructions par une boucle for *en remarquant que l'instruction* somme = somme + i; *est exécutée 10 fois avec* i *prenant pour valeurs les différents entiers entre 1 et 5 car la variable est systématiquement incrémentée (sa valeur est augmentée de 1) grâce à l'instruction* i = i+1;.

On obtient ainsi la boucle :

```
1 int borne = 5;
2 int somme = 0;
3 for(int i = 0; i <= borne; i++){
4     somme = somme + i;
5 }
```

On remplace la première ligne du code précédent par int borne = 100; *quelle est la valeur de* somme *après la suite d'instructions ? Même question si on remplace la première ligne par* int borne = 1000; *?*

Modifier le code précédent pour calculer la somme des entiers de 10 à 100.

Écrire une fonction int sumIntegers (int n) *qui retourne la somme des entier de 0 à n.* □