Introduction à la Programmation 1

Séance 1 de cours/TD

Université Paris-Diderot

Objectifs:

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

1 De quoi est fait un programme ?

Qu'est-ce qu'un programme ? [COURS]

- Un **programme** est la représentation d'une séquence d'instructions à exécuter. Par exemple, une recette est un programme exécuté par un cuisinier. Un code source JAVA est un programme exécuté par un ordinateur.

Qu'est-ce qu'un ordinateur ? [COURS]

- En première approximation, un **ordinateur** est une machine à calculer munie d'une mémoire et connectée à des périphériques (comme un écran, une carte réseau, etc).
- Un ordinateur peut soit **faire des calculs** qui produisent des **valeurs** ; soit **effectuer des actions** en transmettant des valeurs à sa mémoire ou à des périphériques.
- Les calculs peuvent dépendre de valeurs récoltées par les périphériques ou lues dans la mémoire.

Les constituants d'un programme [COURS]

- Les **expressions** décrivent des calculs à faire.
- Les **instructions** décrivent des actions à effectuer.
- Les **déclarations** donnent des noms aux constituants du programme.

```
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            }
10           drawInt (numberOfSecs - i, alertCode);
11       }
```

Listing 1 – Que fait cette séquence d'instructions?

Exercice 1 (Premier décodage, ⋆)
Dans le programme JAVA *précédent, classifiez les parties du code source correspondant aux expressions, aux instructions et aux déclarations.* □

2 Le langage des expressions arithmétiques

Sous-langage des expressions arithmétiques [COURS]

— Les expressions sont classifiées à l'aide de **types** correspondant à la forme des valeurs qu'elles calculent.
— Le type `int` est celui des expressions qui calculent des valeurs entières, les expressions arithmétiques.
— Le type `int` est l'ensemble des valeurs 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
  {`x9/F189TJ/F549.9626Tf6.0861.`

— 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 *assignant* le résultat d'un nouveau calcul. L'opérateur d'affectation est « = » :

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

— Sur papier, une mémoire contenant les variables `x = 42`, `y = 73` et `z = 37` sera écrite :

x	y	z
42	73	37

Instructions [Cours]

— Une expression calcule une valeur tandis qu'une instruction a un effet sur la machine.
— Une affectation est un exemple d'instruction.
— On dit qu'une machine exécute une instruction.

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

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

□

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

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

□

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

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

□

Exercice 6 (Affectations, ⋆⋆)
Quelle est la valeur des variables, à la suite des instructions suivantes.

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

□

4 Fonctions et procédures

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 attendre plusieurs paramètres en les séparant par des virgules.
— On peut utiliser des fonctions prédéfinies dans des *bibliothèques de fonctions*.

Exercice 7 (Utiliser une fonction, ⋆)
Étant donnée la fonction suivante, que vaut l'expression `twice(3)` *?*

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

□

Exercice 8 (Utiliser une fonction, ⋆⋆)
On considère la fonction suivante.

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

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

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

□

Exercice 9 (Écrire une fonction, ⋆⋆)
Voici une fonction

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

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 10 (Écrire une fonction, ⋆⋆)
Écrire des fonctions e ectuant 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.*

□

Utiliser une procédure [Cours]

— Un **appel de procédure** est une instruction écrite à l'aide du nom de la procédure suivi d'un ou plusieurs paramètres séparés par des virgules et entourés de parenthèses. Par exemple :

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

affiche 3 à l'écran à l'aide de la procédure prédéfinie `printInt`. Un autre exemple :

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

affiche un pixel blanc en position (0, 0) d'une image à l'aide de la procédure prédéfinie `putPixel`.

Écrire une procédure [Cours]

— Une procédure est définie en termes d'une suite d'instructions.
— Par exemple, la procédure qui attend deux entiers `x` et `y` et qui affiche successivement leur somme et leur produit s'écrit :

```
public static void showProductAndSum (int x, int y) {
  printInt (x + y);
  printInt (x * y);
}
```

Exercice 11 (Différences syntaxiques, ⋆⋆)

1. *Quelles di érences trouvez-vous entre la syntaxe de déclaration des fonctions et des procédures ?*
2. *Même question pour l'appel de fonctions et l'appel de procédures.*

□

5 Conditionnelle

Instruction conditionnelle [Cours]

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

```
int x = 10;
int y = 42;
if (x <= 0) {
    y = x;
} else {
    y = -x;
}
```

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 12 (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;
}
```

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

□

Exercice 13 (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;
}
```

□

6 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'*en-tête* 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`" déclare le compteur de boucle, son type et sa valeur initiale.
— 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);
}
```

— Nous verrons qu'il existe un autre type de boucle introduite par le mot-clé `while`.

Exercice 14 (Afficher des suites d'entiers, ⋆)

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

□

7 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);
}
```

8 Do it yourself

Exercice 15 (Valeurs d'expressions entières, ⋆)
Donnez la valeur des trois expressions suivantes : `3 + 5 / 3` `4 * 1 / 4` `2 / 3 * 3 - 2` □

Exercice 16 (Modulo 9, ⋆⋆)
Donnez la valeur des expressions suivantes :
`18 % 9` `81 + 18 % 9` `(81 + 18) % 9`

□

Exercice 17 (Valeur absolue, ⋆)
En vous inspirant de l'exercice 12, donner une suite d'instructions permettant de calculer la valeur absolue d'une variable. □

Exercice 18 (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 19 (Boucles simples, ⋆⋆)

1. *Quel est l'a chage 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 20 (Des entiers qui s'ajoutent, ⋆⋆)
Quelle est la valeur de sum *après la suite d'instructions suivante :*

```
int i = 0;
int sum = 0;
sum = sum + i;
i = i + 1;
sum = sum + i;
i = i + 1;
sum = sum + i;
i = i + 1;
sum = sum + i;
i = i + 1;
sum = sum + i;
i = i + 1;
sum = sum + 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 "*`sum = sum + i;`*" est exécutée 10 fois avec* `i` *prenant pour valeurs les di érents 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 bound = 5;
2 int sum = 0;
3 for (int i = 0; i <= bound; i++) {
4     sum = sum + i;
5 }
```

1. *On remplace la première ligne du code précédent par "*`int bound = 100;`*".*
   Quelle est la valeur de `sum` *après l'exécution de cette suite d'instructions ?*
   Même question si on remplace la première ligne par "`int bound = 1000;`*" ?*
2. *Modifiez le code précédent pour calculer la somme des entiers de* 10 *à* 100*.*
3. *Écrivez une fonction "*`int sumIntegers (int n)`*" qui retourne la somme des entiers de* 0 *à* n*.*

□