Introduction à la Programmation 1

Séance 6 de cours/TD

Université Paris-Diderot

Objectifs:

- Savoir écrire un programme en entier (structure du programme avec une procédure main et des déclarations de fonctions, de procédures et de variables globales).
- Savoir compiler et exécuter un programme.

```java
public class ExoExecution {

    public static void main (String[] args) {
        int p = 6;
        int q = 3;
        if (p % q == 0)
            printString ("q divise p.");
        if (power (q, 2) <= p)
            printString ("p est plus grand que le carré de q.");
        if (p / q >= 2)
            printString ("p est plus grand que le double de q.");
    }

    /* Retourne base élevée à la puissance exponent. */
    public static int power (int base, int exponent) {
        int result = 1;
        while (exponent > 0) {
            result = result * base;
            exponent = exponent - 1;
        }
        return result;
    }

    /* Affiche une chaîne de caractères et saute une ligne. */
    public static void printString (String s) {
        System.out.println (s);
    }
}
```

Listing 1 – Un programme Java complet

1 Structure d'un programme Java

Structure d'un programme Java [Cours]

- Le nom du programme suit le mot-clé class.
- Le nom du fichier doit être le nom du programme suivi du suffixe `.java`, le fichier du programme ci-dessus s'appellera donc `ExoExecution.java`.
- Le programme contient une procédure qui s'appelle main. L'exécution du programme consiste à exécuter la procédure main. Elle débute donc à la première ligne du corps de cette procédure.
- Pour éviter un main trop long et illisible, on peut bien sûr (et c'est fortement conseillé !) introduire des procédures et des fonctions auxiliaires qui permettent de regrouper des suites instructions similaires en des "briques" réutilisables dont les noms rendent plus clair le sens du programme.

Exercice 1 (Ordre d'exécution, ⋆)
Écrire la séquence des lignes du code du programme ExoExecution *dans l'ordre de leur exécution.* □

Exercice 2 (Ajouter des fonctions, ⋆)
Ajouter la fonction divides *suivante au programme* ExoExecution *et modifier le corps du* main *en conséquence.*

```
public static bool divides (int q, int p) {
  return (p % q == 0);
}
```

□

Structure des fonctions [Cours]

- Une fonction est une série d'instructions *qui retourne une valeur*.
- Les fonctions permettent de rendre un programme plus compréhensible en augmentant le vocabulaire du langage : on donne un nom à une série d'instructions calculant une certaine valeur et il n'est alors plus nécessaire de se souvenir de cette série d'instructions car on peut utiliser le nom de fonction introduit comme si il s'agissait d'une nouvelle primitive du langage.
- Les fonctions permettent de factoriser du code : plutôt que d'écrire plusieurs fois la même série d'instructions (avec éventuellement quelques variations artificielles), on écrit une seule fois ces instructions en *explicitant des paramètres* pour représenter les variations. Par exemple, à la place de :

```
int z = y * y + (1 + y) * (1 + y) + (2 + y) * (2 + y);
```

On peut introduire la fonction square suivante :

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

et remplacer la déclaration de la variable z par :

```
int z = square (y) + square (1 + y) + square (2 + y);
```

- Une fonction est caractérisée par :
  - **un corps** : le bloc d'instructions qui décrit ce que la fonction calcule ;
  - **un nom** : par lequel on désignera cette fonction ;
  - **des paramètres** (l'*entrée*) : l'ensemble des variables extérieures à la fonction dont le corps dépend pour fonctionner et qui prennent une valeur au moment de l'appel de la fonction ;
  - **une valeur de retour** (la *sortie*) : ce que la fonction renvoie à son appelant.

  Par exemple, le code du programme ExoExecution ci-dessus définit une fonction de nom power, qui prend en paramètre deux valeurs entières base et exponent, de type int, et renvoie en sortie une valeur de type int qui vaut $\texttt{base}^{\texttt{exponent}}$:

```
    public static int power (int base, int exponent) {
        int result = 1;
        while (exponent > 0) {
```

```
            result = result * base;
            exponent = exponent - 1;
        }
        return result;
    }
```

- La première ligne qui déclare la fonction est appelée **en-tête** ou **signature** de la fonction et précise le type de sa valeur de retour, son nom et les types et les noms de chacun de ses paramètres.
- Le sens précis de public et static seront traités dans le cours de programmation orientée objet en Java.
- Le choix des noms de la fonction et de ses paramètres fournit une documentation minimale du comportement de la fonction, il est donc souhaitable qu'ils soient parlant. Par exemple, le programme est moins compréhensible si on remplace l'en-tête de power par « public static int function1 (int x, int y) ».
- Dans le corps de la fonction se trouvent les instructions qui détaillent la manière dont la valeur de sortie est calculée. Ces instructions utilisent les paramètres de la fonction (base et exponent) et, éventuellement, de nouvelles variables déclarées localement dans le corps (dans notre exemple, on utilise la variable result).
- La valeur retournée par la fonction est indiquée par l'instruction

```
        return expression ;
```

  où expression a le même type que celui de la valeur de sortie déclaré dans l'en-tête de la fonction.
- L'instruction return fait deux choses :
  1. elle précise la valeur qui sera fournie par la fonction en résultat,
  2. elle met fin à l'exécution des instructions de la fonction.
- Il est possible d'avoir plusieurs occurrences de `return` dans le même corps, comme dans la fonction suivante :

```
public static int minimum (int a, int b) {
  if (a < b)
    return a;
  else
    return b;
}
```

- On doit s'assurer que tout chemin d'exécution possible du corps d'une fonction mène à un return, sinon Java rejette le programme. Par exemple :

```
public static int minimum (int a, int b) {
  if (a < b)
    return a;
}
```

  produit l'erreur suivante à la compilation :

```
ExoInvalidMinimum.java:10: missing return statement
```

- Une fois définie, il est possible d'*utiliser la fonction dans toute expression*, on parle alors d'**appel** (où d'**invocation**) d'une fonction. On peut par exemple écrire :

```
int a = 3 + power (4, 2);
```

  Dans cette déclaration, la sous-expression « power (4, 2) » est remplacée par la valeur retournée par le corps de la fonction power pour base valant 4 et exponent valant 2, c'est-à-dire 16. On évalue ensuite $3+16$ pour obtenir la valeur 19 qui sera la valeur initiale de a.
- L'appel d'une fonction doit respecter la signature de cette fonction, c'est-à-dire le nombre et les types des paramètres ainsi que le type de la valeur de sortie.

Exercice 3 (Évaluer un appel, $\star$)
Soit power *la fonction définie dans le programme* ExoExecution *ci-dessus, quelle est la valeur de la variable* x

après exécution de la suite de instructions qui suit :

```
int x = 2;
int y = 2;
x = power (x, y);
x = power (x, y);
x = power (x - 8, y);
```

□

Exercice 4 (Appelle-moi bien !, ⋆)
Soient « public static int power(int base, int exponent) » une signature et x, y *deux variables de type* int. *Trouver les appels incorrects parmi ceux de la liste suivante. Motiver votre réponse.*

1. « y = power (2, x); »
2. « int a = power ("2", 3); »
3. « String a = power (2, x); »
4. « if (power (2, x) > 5) { ... } »
5. « while (power (2, x)) { ... } »
6. « x = power (x, power (x, x)); »
7. « printInt (power(x, y)); »
8. « x = power(x, printInt (y)); »

□

Exercice 5 (Mauvais en-tête, ⋆⋆)
Dire pourquoi aucun des en-têtes suivants n'est correct.

1. « public static int fonc1 (int a; int b) »
2. « public static fonc2 (boolean c) »
3. « public static int, fonc3 (String w) »
4. « public static boolean fonc4 (int) »
5. « public static String fonc5 (String t, int d, int t) »
6. « public static int 6fonc (int x) »

□

Exercice 6 (En-tête déduite d'une spécification, ⋆)
Donner l'en-tête d'une fonction qui prend en paramètre une chaîne de caractères et renvoie sa longueur. □

Exercice 7 (En-tête déduite d'un corps de fonction, ⋆⋆)
Donner l'en-tête de la fonction dont le corps est défini comme suit. (La fonction « public static int intArrayLength (int [] t) *» prend en paramètre un tableau d'entiers et renvoie sa longueur.) :*

```
... {
  int r = 0;
  int i = 0;

  if (intArrayLength (t) != intArrayLength (u)) {
    return -1;
  }

  while (i < intArrayLength (t)) {
    r = r + t[i] * u[i];
    i = i + 1;
  }

```

```
14   return r;
15 }
```

□

Les procédures [Cours]

- Les **procédures** sont des suites d'instructions qui ne renvoient pas de valeur mais qui produisent des *effets* sur le système qui les exécute. Par exemple, la procédure « void printString(String s) » affiche un message à l'écran.
- Les procédures ont un en-tête et un corps, mais le type de leur valeur de retour est void (*vide* en français) et le corps ne contient donc pas obligatoirement l'instruction return. Par exemple :

```
1 public static void printRange(int from, int to) {
2   for (int i = from; i <= to; i++) {
3     printInt(i);
4   }
5 }
```

- La procédure main a un rôle particulier car elle contient les premières instructions à exécuter lorsque le programme est lancé : on dit que main est le *point d'entrée* du programme.
- La signature de main est obligatoirement « public static void main(String [] args) ». On verra par la suite à quoi sert le tableau de chaînes de caractères args.

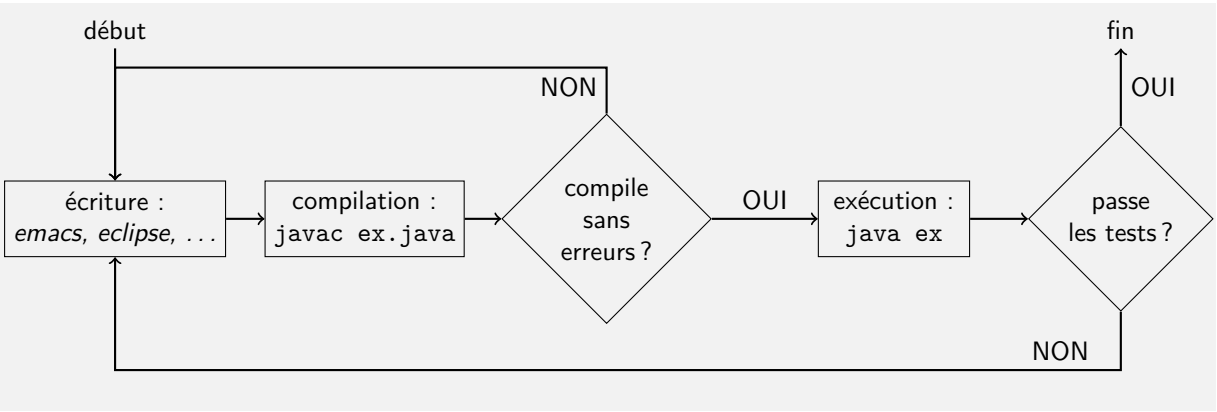
2 Taper, Coder, Exécuter

Les trois étapes principales de la programmation [Cours]

Pour obtenir un programme, on suit les trois étapes suivantes :

1. L'**écriture** du code source en utilisant un éditeur de texte (par exemple `Emacs`, `Eclipse`, ...).
   - Le nom du fichier contenant la procédure main doit être le nom du programme suivi du suffixe `.java`. Par exemple, pour l'exemple de la page 1, le code se trouve dans le fichier ExoExecution.java.
   - La procédure main décrit les instructions principales du programme et, éventuellement, utilise d'autres fonctions auxiliaires définies dans le même fichier ou dans des bibliothèques (comme par exemple la bibliothèque Math de `Java`).
2. La **compilation** du code source, effectuée par le programme `javac` (*java compiler*) auquel on a donné le nom de fichier contenant le code source (qui porte le suffixe `.java`).
   - Cette commande transforme le code qui est un texte lisible par un être humain en *bytecode* lisible par la machine virtuelle JVM (*Java Virtual Machine*). Le nom du fichier en bytecode se termine par le suffixe `.class`.
   - Au passage le compilateur `javac` vérifie que le programme est syntaxiquement correct et qu'il est bien typé, c'est-à-dire que les expressions produisent bien des valeurs du type attendu. Si le fichier ne respecte pas l'un de ces points (par exemple, si on oublie un ; après une instruction), une erreur se produit à la compilation et aucun fichier `.class` n'est créé. Il faut alors revenir au point 1 et corriger le code source.
3. L'**exécution** du bytecode par la machine virtuelle JVM. Cette exécution s'effectue par la commande `java` suivie du nom du programme contenant la procédure main.
   - La machine virtuelle exécute les instructions de la procédure main.
   - **Attention** : une compilation qui se passe bien ne signifie pas que le programme est correct, des erreurs peuvent se produire à l'exécution. Il faut donc tester le programme et vérifier que le comportement est celui attendu. Dans le cas contraire, il faut revenir au point 1 et corriger le code source.

Le diagramme suivant résume ces trois étapes :



Exercice 8 (Compiler, exécuter, ⋆⋆)

Considérez le code du programme ExoExecution *présenté en première page du TD.*

- *Quel doit être le nom du fichier contenant ce code source ?*
- *Quelle commande permet de le compiler ?*
- *Quel est le nom du fichier* bytecode *obtenu*
- *Comment réalise-t-on son exécution ?*
- *Que se passe-t-il si on élimine la ligne 5 ?*
- *Que se passe-t-il si on la remplace par l'instruction suivante ?*

```
int q = 0;
```

□

Passer des arguments à la procédure main [Cours]

- Comme dit précédemment, l'en-tête de la procédure main est « public static void main(String [] args) ». Elle prend en paramètre un tableau de chaînes de caractères, où sont stockées les valeurs qui suivent le nom du programme dans la commande qui provoque son exécution.
- Par exemple, pour le programme ExoExecution, la commande

```
1 java ExoExecution a1 a2 a3
```

fixe le contenu du tableau args a :

```
1 { "a1", "a2", "a3" }
```

Exercice 9 (Utilisation args, ⋆)

Considérons le programme suivant :

```
1  public class ExoArgs {
2
3      public static void main (String[] args) {
4          printString ("Bonjour " + args[0]);
5      }
6
7      /* Affiche une chaîne de caractères. */
8      public static void printString (String s) {
9          System.out.print (s);
10     }
11 }
```

code/ExoArgs.java

Après compilation du fichier ExoArgs.java, *qu'affichent à l'écran les commandes suivantes ?*

1. « `java ExoArgs Jean-Paul` »

2. « `java ExoArgs Jean Paul` »
3. « `java ExoArgs` »

□

Exercice 10 (Option de verbosité, ⋆⋆)
Considérons le code source suivant :

```java
public class ExoVerbose {
    public static void main (String [] args) {
        int r = 0,
        int x = 10;
        for (int i = 1; i <= x; i++) {
            r = r + i;
        }
        printString (" La somme des premiers ");
        printInt (x);
        printString (" nombres entiers est ");
        printInt (r);
    }

    /* Affiche une chaîne de caractères s. */
    public static void printString (String s) {
        System.out.print (s);
    }

    /* Affiche un entier n. */
    public static void printInt (int n) {
        System.out.print (n);
    }
}
```

Modifier le programme de façon à ce que :
– *l'exécution sans option (*`java ExoVerbose`*) affiche sur l'écran la somme des dix premiers nombres entiers ;*
– *l'exécution avec l'option* `verbose` *(*`java ExoVerbose verbose`*) affiche sur l'écran toutes les valeurs des affectations exécutées sur* `i` *et* `r` *et ensuite la somme des* 10 *premiers entiers ;*
– *l'exécution avec toute autre option (par exemple* `java ExoVerbose verbose all`*) n'exécute pas le calcul mais affiche sur l'écran :* `"Ces options ne sont pas connues"`*.*
Vous pouvez utiliser la fonction auxiliaire « public static int stringArrayLength (String [] t) », qui calcule la longueur d'un tableau de String*, et « public static boolean stringEquals (String st1, String st2) », qui teste l'égalité de deux chaînes de caractères.* □

3 Fonctions utilisées

Liste des fonctions [Cours]

```java
/* Retourne base élevée à la puissance exponent. */
public static int power (int base, int exponent) {
  int result = 1;
  while (exponent > 0) {
    result = result * base ;
    exponent = exponent - 1;
  }
  return result;
```

```java
}

/* Affiche une chaîne de caractères s. */
public static void printString (String s) {
   System.out.print (s);
}

/* Affiche un retour de ligne a l'écran*/
public static void newLine () {
  System.out.print ("\n");
}

/* Retourne le nombre de cases d'un tableau d'entiers. */
public static int intArrayLength (int[] t) {
   return t.length;
}

/* Retourne le nombre de cases d'un tableau de tableau d'entiers. */
public static int intArrayArrayLength (int[][] t) {
  return t.length;
}

/* Retourne le nombre de cases d'un tableau de chaînes de
caractères. */
public static int stringArrayLength (String[] t) {
   return t.length;
}

/* Teste si deux chaînes de caractères st1 et st2 ont le même contenu.*/
public static boolean stringEquals (String st1, String st2) {
   return st1.equals (st2);
}

/* Transforme une chaîne de caractères en un entier n. */
public static int stringToInt (String s) {
    return Integer.parseInt (s);
}
```

4 DIY

Exercice 11 (Devine la fonction, ⋆⋆)
Définir la fonction average *utilisée dans le code suivant (les variables* m *et* grades *sont supposées être définies avant cette portion de code) :*

```java
        while (e < intArrayArrayLength (grades)) {
            if (intArrayLength (grades[e]) == 0) {
                printString ("l'étudiant numéro ");
                printInt (e);
                printString (" n'a aucune note !\n");
            } else {
                m = average (grades[e]);
                if (m < 10) {
                    printString ("l'étudiant numéro ");
                    printInt (e);
```

```
                    printString (" n'a pas passé le semestre.\n");
                } else {
                    printString ("l'étudiant numéro ");
                    printInt (e);
                    printString (" valide avec la moyenne de ");
                    printInt (m);
                    printString ("\n");
                }
            }
            e = e + 1;
        }
    }

    /* Retourne la moyenne des entiers contenus dans le tableau tab. */
    public static int average (int[] t) {
```

code/ExoGuess.java

□

Exercice 12 (Primalité, ⋆⋆)
Un entier p est premier si $p \geq 2$ et s'il n'a pas d'autres diviseurs que 1 et lui-même.

1. *Écrire une fonction* prime *qui prend en paramètre un entier n et qui renvoie* true *si n est premier, ou* false *sinon.*
2. *Écrire une fonction* next *qui prend en entrée un entier x et qui renvoie le plus petit nombre premier $p \geq x$. On pourra bien sûr se servir de la fonction* prime *précédente.*
3. *Écrire une fonction* number *qui prend en entrée un entier y et qui renvoie le nombre de nombres premiers $p \leq y$. On pourra bien sûr se servir de la fonction* prime.

□

Exercice 13 (Addition, ⋆⋆⋆)
Le but de cet exercice est de programmer l'addition décimale. Les deux nombres à additionner sont donnés sous forme de tableaux.

Par exemple, x $=$ { 7, 4, 3 } *et* y $=$ { 1, 9 }*, dont la somme doit être retournée sous forme de tableau, dans cet exemple,* $\{7, 6, 2\}$.

1. *Écrire une fonction* reverse *qui prend en paramètre un tableau d'entiers, retourne le tableau qui contient les mêmes chiffres, dans l'ordre inverse. Par exemple, si l'entrée est le tableau* { 7, 4, 3 }*, on doit rendre un tableau de trois chiffres* { 3, 4, 7 }.
2. *Écrire une fonction* add *qui prend en paramètre deux tableaux et fait l'addition de gauche à droite des deux nombres représentés par les tableaux.*
   Par exemple, si les entrées sont les tableaux t1 $=$ { 3, 4, 7 }, t2 $=$ { 9, 1 }*, cela représente la somme* $743 + 19$. *On calcule d'abord* $9 + 3$ *ce qui fait* 2 *avec une retenue de* 1. *Puis on calcule* $4 + 1$*, plus la retenue, ce qui donne* 6*, avec une retenue de* 0. *Enfin, le dernier chiffre est* 7. *À la fin, on doit retourner le tableau* { 2, 6, 7 }.
3. *Écrire un fragment de code qui permet de faire l'addition* $2435 + 436$.

□

Exercice 14 (Perroquet, ⋆)
Écrire un programme `ExoPerroquet` *dont l'exécution affiche à l'écran "*`Je suis un Perroquet`*" et puis, dans une suite de lignes, tous les mots donnés en argument au programme :*

```
java ExoPerroquet toto titi
je suis un Perroquet
toto
titi
```

□

Exercice 15 (Perroquet avec options, ⋆⋆)
Écrire un programme `ExoPerroquetOpt` *dont l'exécution affiche à l'écran "*`Je suis un Perroquet`*" sur la première ligne ; sur la ligne suivante, il affiche d'abord "*`Options: `*", puis la liste des mots donnés en argument au programme si et seulement si ils commencent avec le caractère "−" (tiret). La présence d'un mot en argument qui ne commence pas par un tiret doit engendrer l'arrêt du traitement des mots suivants et l'affichage du message "*`erreur: MOT n'est pas une option`*" où* `MOT` *est le premier mot qui ne commence pas avec un tiret.* □

Exercice 16 (Pascal, ⋆⋆⋆)
Écrire un programme `ExoPascal` *qui affiche le triangle de Pascal jusqu'au rang n, passé en paramètre au moment du lancement du programme :*

```
1 java ExoPascal 4
2 1
3 1 1
4 1 2 1
5 1 3 3 1
6 1 4 6 4 1
```

La procédure `main` *doit faire appel à une fonction auxiliaire qui, étant donné n, renvoie le triangle de Pascal sous forme d'un tableau de tableaux d'entiers de $n + 1$ lignes, la i-ème ligne contenant les coefficients de la i-ème puissance du binôme $(a + b)$. On utilisera une fonction prédéfinie «* `int stringToInt(String s)` *» qui transforme une chaine de caractères* `"n"` *dans l'entier* `n` *qui contient.* □