Cours Introduction à la Programmation
Java II
(IP1 JAVA)

Arnaud Sangnier
sangnier@irif.fr

Mercredi 27 Septembre 2017
INFO et MATHS-INFO

Dans l'épisode précédent - I

- Les programmes s'écrivent dans des langages
- Cette année : **Java**
- Pour écrire un programme, on utilise un éditeur
- Pour exécuter un programme, deux phases :

  1) **Compiltation** : traduction du code source en binaire

  ```
  javac MonProgramme.java
  ```

  2) **Exécution** :

  ```
  java MonProgramme
  ```

- Un programme est composé de différentes instructions
- Pour qu'un programme affiche quelque chose, il faut qu'on lui dise, par exemple :

  ```
  System.out.println ("Hello World !")
  ```

Dans l'épisode précédent - II

- Un programme manipule des données auxquelles sont associées un type
- Nous avons vu deux types : **int** et **String**
- Il faut toujours savoir quel est le type de la donnée manipulée
- Il faut éviter de mélanger des données de type différente dans les opérations
  - Par exemple : éviter de faire 3 + ‘’Hello’’

Dans l'épisode précédent - III

- Présentation des variables
- Elle servent à stocker des données
- Un nom de variable commence par une lettre
- Trois opérations possibles
  - **Affectation**, **lecture** et **modification**
- Le signe = sert à affecter ou à modifier et n'est pas comme l'égal mathématique
- Pour interpréter x = y + 5 , le programme calcule d'abord y + 5 et mais ensuite le résultat dans x

Plusieurs valeurs possibles mais un seul programme

- Une clef essentielle de la programmation est d'écrire des programmes qui vont marcher pour différentes valeurs possibles
- Par exemple :
  - Un programme qui calcule n! (factorielle de n)
  - Un programme qui calcule la somme de deux entiers a et b
  - Un programme qui calcule le pgcd de deux entiers a et b
- Ici les valeurs de n, a et b ne sont pas connus à l'avance, on sait juste qu'il s'agit d'entier

Les fonctions

- Une fonction d'un programme est une liste d'instructions
- Elle peut être appelée plusieurs fois
- Elle peut prendre des valeurs en entrée
  - Il s'agit des **arguments**
- Elle peut calculer une valeur et la renvoyer
  - Il s'agit de la valeur de retour
- On lui donne un nom (le nom de fonctions)
- Exemple : la fonction System.out.print( ) qui ne retourne pas de valeur mais qui affiche à l'écran

Exemple de fonctions

```
public static int f (int x) {
        return (2 * x) ;
}
… main(...){
  int a = f(3) ;
  int b = f(5) ;
  System.out.println(a);
  System.out.println(b);
}
```

Définition de la fonction f

Appel de la fonction f

- Affiche

```
6
10
```

- **Si on enlève les** System.out.println**, le programme n'affiche rien**

Que fait la machine ?

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

- Si par exemple on a une ligne int z = f (4)
  1) Elle remplace la valeur x de f par 4
  2) Elle calcule 2*4
  3) Elle renvoie la valeur 8
  4) Elle stocke cette valeur dans z

Exemple

```
public static int f (int x) {
        return (2 * x) ;
}
… main … {
  int z = 10 ;
  z = f(z) ;
  System.out.println(z) ;
}
```

- Affiche

```
20
```

Exemple

```
public static int f (int x) {
        return (2 * x) ;
}
… main(…) {
  int x = 10;
  int z = f(x);
  System.out.println(x);
  System.out.println(z);
}
```

ATTENTION :
Les deux x ne sont pas la même variable

- Affiche

```
10
22
```

Exemple

```
public static int f (int x) {
        return (2 * x) ;
}
… main(…) {
  int x = 10;
  int z = f(x);
  System.out.println(x);
  System.out.println(z);
}
```

- Plus sûr en écrivant :

```
public static int f (int y) {
        return (2 * y) ;
}
… main(…) {
  int x = 10;
  int z = f(x);
  System.out.println(x);
  System.out.println(z);
}
```

On évite ainsi les confusions possibles

Les fonctions sans return

- Dans certains cas, on peut vouloir qu'une fonction ne retourne aucune valeur
- Il n'y a pas de return dans le code
- Le type de retour est alors **void**
- Pourquoi faire ?
  - Par exemple une fonction d'affichage
  - On peut vouloir afficher un message 'paramétré' par un argument
  - Par exemple, afficher le message Hello nom où nom est remplacé par un nom donné en argument
- Si dans une fonction f, il n'y a pas de return, il ne faut pas faire x = f () → **cela n'a aucun sens** !

Exemple

```
public static void affiche(String s) {
  System.out.println(‘’Hello ‘’+s);
}
… main (…) {
  affiche("Bob");
  affiche("Alice");
}
```

- Affiche

```
Hello Bob
Hello Alice
```

Exemple

```
public static void affiche(String s) {
  System.out.println(‘’Hello ‘’+s);
}
… main (…) {
 String x = affiche("Martin");
}
```

Pas de return
dans la fonction
Le type de retour est void

Aucun sens
→ Qu'il y a-t-il dans x ?????

Il faut toujours savoir si une fonction renvoie une valeur et aussi quel est le type de la valeur retournée

Règles pour l'écriture de fonctions

- Pour écrire une fonction il faut se poser les questions suivantes:
  - Quelle est le nom de la fonction ?
  - Combien d'arguments prend la fonction et quels sont leur type ?
  - Quel est le type de retour de la fonction
- **Attention :**
  - pour chacun des arguments, on doit le nommer avec **type + variable**
  - pour le retour : **juste un type**, **pas de variable**

Les tests

- Comme un programme doit être paramètrable selon les valeurs qu'on lui donne, il est important de pouvoir tester ces valeurs
- Pour cela on a l'instruction if … else …
- Par exemple, pour dire si la valeur dans x est positive ou nulle fait quelque chose sinon fait autre chose
- Pour tester ou comparer deux entiers on a les opérateurs : <, >, >=,<=, ==,!=
- Pour tester ou comparer deux chaînes de caractères, on ne peut pas utiliser ==, c'est plus compliqué
  - On fait ‘’Hello’’.equals(‘’Hallo’’) ou s.equals(‘’Bim’’) si s est une variable de type String

Exemple

```
if (x == 5) {
        System.out.println("AH");
} else {
        System.out.println("OH");
}
```

Affiche AH si x vaut 5 et OH sinon

```
if (v.equals("Hi")){
  System.out.println("Salut");
} else {
  System.out.println("Salut");
}
```

Affiche Salut si v vaut "Hi" et sinon affiche Que veux-tu ?

Une première fonction

- Énoncé : faire une fonction qui renvoie la valeur absolue d'un entier
- D'abord déterminer les informations suivantes :
  - Combien de paramètres : un seul qui prend des valeurs de type int
  - Retourne-t-elle une valeur : oui de type int
  - Comment va-t-on l'appeler : par exemple valabs

```
public static int valabs (int v) {


        return
}
```

Il nous faut remplir ici et savoir quoi retourner

Une première fonction

- Énoncé : faire une fonction qui renvoie la valeur absolue d'un entier
- Comment fonctionne la valeur absolue :
  - Si la la valeur de v est positive ou nulle, c'est la valeur de v
  - Si la valeur de v est négative, c'est – la valeur de v

```
public static int valabs (int v) {
        int x = 0;
        if (v >=0) {
                x = v;
        } else {
                x = -v;
        }
        return x;
}
```

Exemple

```
public static int valabs (int v) {
        int x = 0;
        if (v >=0) {
                x = v;
        } else {
                x = -v;
        }
        return x;
}
… main(...){
   int y = valabs(-10);
   int z = valabs(12);
   System.out.println(y);
   System.out.println(z);
}
```

- Affiche

```
10
12
```

Une deuxième fonction

- Énoncé : faire une fonction qui prend en arguments deux chaines de caractères et teste si elles ont la même longueur et affiche 'Meme' si c'est le cas et 'Different' sinon
- Pour récupérer la longueur d’une chaîne de caractères ch, on peut faire ch.length() (il s’agit d’un int)
- Déterminer les informations suivantes :
  - Combien de paramètres : deux qui prennent des valeurs de type String
  - Retourne-t-elle une valeur : non
  - Comment va-t-on l'appeler : par exemple complen

Exemple

```
public static void complen (String s1,String s2){
        if (s1.length() == s2.length()){
          System.out.println("Meme");
        } else {
          System.out.println("Different");
        }
}
… main(...){
  complen("Bob", "Lol");
  complen("a","ab");
}
```

- Affiche

```
Meme
Different
```

Les boucles

- Une autre chose intéressante sont les boucles
- Un programme répète souvent les mêmes instructions en boucle
- Il existe une instruction pour dire au programme de répéter une instruction un certain nombre de fois
  - Par exemple for (int i=0;i<10;i =i + 1)
  - En français, il faut lire pour i allant de 0 (inclus) à 10 (exclus) en augmentant de 1 à chaque étape
- Par exemple pour dire à un programme d'afficher 1000 fois Bonjour
  - On peut copier-coller 1000 fois print("Bonjour") →
  - On peut utiliser une boucle

Exemple

```
for (int i=0;i<1000;i=i+1){
  System.out.println("Bonjour");
}
```

- Affiche

```
Bonjour
Bonjour
Bonjour
Bonjour
Bonjour
.
.
Bonjour
```

Piège avec les boucles

- Il faut être attentif à combien de fois on fait la boucle
  - Par exemple for (int i=1;i<9;i=i+1) → on fait la boucle 8 fois
  - for (int i=1;i<10;i=i+2) → on fait la boucle 5 fois (valeur de i : 1,3,5,7,9)
- La variable i dans for (int i=1;i<9;i=i+1) :
  - Est une variable normale, dont la valeur change à chaque tour de boucle
  - **Il ne faut pas la modifier dans la boucle for**

Exemple

```
for (int i=0;i<4;i=i+1){
  System.out.println(i);
  System.out.println(2*i);
}
```

- Affiche

```
0
0
1
2
2
4
3
6
```

Utilité des boucles

- Est-ce-que les boucles ne servent qu'à afficher ? NON
- Elles sont aussi utiles
  - pour faire des calculs mathématiques (par exemple somme des n premiers entiers, factorielle, etc)
  - pour parcourir des listes (on verra ça plus tard)
- Le comportement répétitif est vraiment la base de la plupart des programmes
  - Pensez à votre ordinateur ou tablette, le programme sous-jacent fait tout le temps la même chose, il attend que vous lanciez d'autres programmes ou que vous fassiez une action

Encore une fonction

- Énoncé : faire une fonction qui prend en arguments un entier n et renvoie la somme des entiers de 1 à n
- Déterminer les informations suivantes :
  - Combien de paramètres : un qui prend une valeur de type int
  - Retourne-t-elle une valeur : oui une valeur de type int
  - Comment va-t-on l'appeler : par exemple sumN
- Comment la programmer
  - Il faut ajouter 1 puis 2 puis 3 puis 4 etc jusqu'à n
  - On a l'impression qu'il y a une boucle
  - À chaque fois on ajoute i
  - Où stocker le résultat de ce calcul → Utilisons une variable

Exemple

```
public static int sumN (int n){
  int x = 0;
  for (int i=1;i<n+1;i=i+1){
    x = x + i;
  }
  return x;
}
… main(…) {
  int y = sumN (10);
  System.out.println(y);
}
```

Observez bien les bornes\ dans la boucle

- Affiche

```
55
```

Une fonction classique

- Énoncé : faire une fonction qui prend en arguments un entier n et renvoie factorielle de n
- Déterminer les informations suivantes :
  - Combien de paramètres : un qui prend une valeur de type int
  - Retourne-t-elle une valeur : oui une valeur de type int
  - Comment va-t-on l'appeler : par exemple fact
- Comment la programmer → Très similaire à la fonction précédente
  - On multiplie 1 par 2 puis par 3 puis par 4 etc jusqu'à n
  - On a l'impression qu'il y a une boucle
  - À chaque fois on ajoute i
  - Où stocker le résultat de ce calcul → Utilisons une variable

Exemple

```
public static int fact (int n){
  int x = 0;
  for (int i=1;i<n+1;i=i+1){
    x = x * i;
  }
  return x;
}
… main(…) {
  int y = fact (4);
  System.out.println(y);
}
```

**Remplace + par *
Et le tour est joué**

- Affiche

```
24
```


FAUX

Exemple

```
public static int fact (int n){
  int x = 0;
  for (int i=1;i<n+1;i=i+1){
    x = x * i;
  }
  return x;
}
… main(…) {
  int y = fact (4);
  System.out.println(y);
}
```

La variable est mal initialisée
Le programmera affichera 0

- Affiche

```
0
```

Exemple

```
public static int fact (int n){
  int x = 1;
  for (int i=1;i<n+1;i=i+1){
    x = x * i;
  }
  return x;
}
… main(…) {
  int y = fact (4);
  System.out.println(y);
}
```

- Affiche

```
24
```

Combiner les différents types d'instructions

- Comme on l'a vu on peut utiliser des boucles et des tests au sein d'une fonction
- On peut globalement faire toutes les combinations possibles
  - Utilisation de tests dans une boucle
  - De boucle au milieu d'un test
- On peut aussi utiliser une fonction à l'intérieur d'une fonction qui elle même utilise une fonction etc