Université de Béjaia
Faculté des Sciences Exactes
Département d'informatique

2ème Année Licence Informatique
Semestre : 04

Module : Programmation Orientée Objet

Responsable : H. EL BOUHISSI

Introduction au langage Java

Notions fondamentales
(Partie : 02)

Cours élaboré par H. EL BOUHISSI

Les commentaires

Tout programme (grand ou petit, simple ou complexe) contient (ou devrait contenir) des commentaires. Ils ont pour but d'expliquer : Ce qu'est sensé faire le programme, les conventions adoptées, Tout autre information rendant le programme lisible à soi même et surtout à autrui.

Java dispose de trois types de commentaires :

1) Les commentaires multilignes, Un commentaire multiligne commence par les caractères ``/*'' et se terminent par ``*/''.

```
/* Ce programme imprime la chaîne
de caractères "bonjour" à l'écran
*/
```

2) Les commentaires lignes, Les commentaires lignes débutent avec les symboles ``//'' et qui se terminent à la fin de la ligne. // Ce programme imprime la chaîne.

3) Les commentaires de type documentation. Ces commentaires, appelés aussi commentaires javadoc, servent à documenter les classes que l'on définit. Ces commentaires sont encadrés entre ``/**'' et ``*/'' .

```
/** Documentation de la classe .
*/
```

Types et variables

Valeur

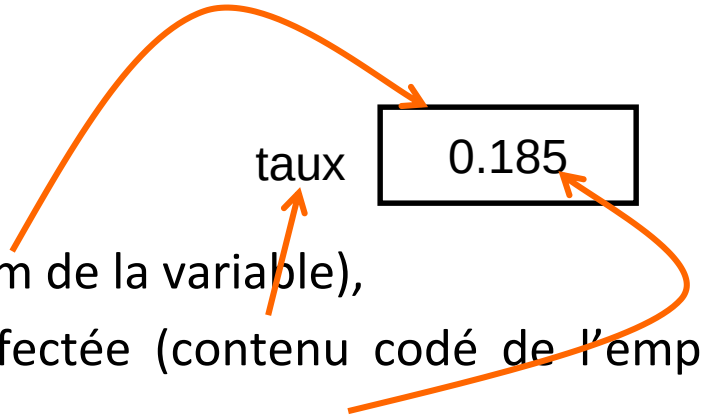
- Information réduite au maximum (non représentable sous une forme plus concise).

  Ex: 2 "1+3" (mais pas 1+3) 4.75

- Chaque valeur est codée/représentée dans la mémoire de l'ordinateur.
- Les valeurs sont les éléments concrets utilisés par la machine.

Variable

- Une variable désigne
  - un **emplacement mémoire**,
  - muni d'un **identificateur** (nom de la variable),
  - auquel une **valeur** a été affectée (contenu codé de l'emplacement mémoire).



Valeurs, types et variables

- Ne pas confondre variable et valeur. La valeur change pendant l'exécution du programme.

Type

- Ensemble de valeurs.
- Détermine la nature de ces valeurs (leur 'sémantique').
- Indique la méthode de **codage** (implantation) et les **opérations** qui sont applicables aux valeurs.

  Ex : *int* est un des types Java pour les valeurs entières; définit comment ces valeurs sont codées et comment on les additionne/multiplie. Idem pour *double* pour les valeurs réelles mais le codage et les opérations sont différentes.

Types primitifs

- Regroupe des valeurs simples directement codées en mémoire comme les valeurs numériques entières ou réelles.
- Principaux types fondamentaux en Java :

type	nature
int	entier entre -2^{31} et $2^{31}-1$
float	Nombre réel
double	nombre réel en double précision
boolean	vrai ou faux
char	un caractère (lettre, chiffre, ponctuation ...)
String	Chaine de caractères

Déclaration de variable

- Un programme ne peut manipuler des valeurs que via des variables qui les contiennent.
- On doit **déclarer** les variables afin d'indiquer leur type.
- **Déclaration simple** :

  type identificateur;

  Ex : *int var1; // déclare une variable var1 entière*

  double valeur1; // déclare une variable valeur1 réelle
- **Déclaration multiple** (N variables avec le même type) :

  type identif1, identif2, …, identifN;

  Ex : *int num1, num2, val_entiere;*

  double valeur, aux;
- Les déclarations peuvent être placées à n'importe quel endroit **précédant** l'utilisation de la variable.

Identificateur

- Suite de caractères parmi :
  - lettres (minuscules et majuscules non confondues),
  - chiffres,
  - symbole soulignement '_' ('underscore').
- Ne peut pas débuter par un chiffre.
- Convention classique : minuscules + soulignements pour séparer les mots.

  Ex : *int indice_de_boucle;*

  boolean condition_remplie;
- Conseil : choisir des identificateurs qui aident à comprendre le programme.

  Ex : une variable qui stocke une moyenne de notes doit être appelée *moyenne* ou *moy_notes* plutôt que *x* ou *truc* !

Affectation

- Modifie **le contenu** d'une variable.

  Ex: *var1=37;* affecte la valeur 37 à la variable var1.
  var1 étant de type entier on ne doit pas affecter une valeur réelle (3.5) ou booléenne (*true*)

- Même si on dit «var1 égale 37» ce n'est pas une comparaison ! L'affectation met 37 dans var1.

- Syntaxe : ***identificateur=valeur littérale du type;***
  ou ***type identificateur=valeur littérale type;***
  (déclaration et affectation simultanée).

Ex :

```
int num1,num2,val_entiere;
val_entiere=-65;
double valeur,aux;
num2=5; valeur=2.7;
num1=8762;
boolean valb;
num2=2; aux=-3.98e-14;
valb=false;
```

```
int num1,num2=5,val_entiere=-65;
num1=8762;
num2=2;
char c='b';
```

Affectation du contenu d'une variable à une autre variable

- **Recopie le contenu** d'une variable dans une autre.

  Syntaxe : ***identificateur2=identificateur1;***

  Ex : *int num1,num2=5,val_entiere;*

  num1=8762;

  val_entiere=num2;

  num2=num1;

- La variable dont le contenu est recopié doit avoir été initialisée (contenir une valeur).
- On peut affecter une valeur de type T à une variable dont le type déclaré « contient » T.

  Ex: l'ensemble des réels « contient » l'ensemble des entiers

  double somme; int val=9;

  somme=6; *// int vers double : ok*

  somme=val; *// int vers double : ok*

  val=somme; */* double vers int : interdit */*

Note : 2 types de commentaires

Saisie au clavier de la valeur d'une variable

- Les mécanismes **d'entrées-sorties** permettent au programme de communiquer avec l'extérieur. Ils sont complexes et seront détaillés plus tard.
- **Saisie au clavier** :
  - importer la classe *Scanner* et créer une variable *Scanner* :

```
import java.util.Scanner;          // en début du fichier .java
Scanner sc = new Scanner(System.in);
```

  - saisir la valeur :

```
int v;                  double v;                String v;
v =sc.nextInt();        v=sc.nextDouble();       v=sc.next();
// entier               // réel                  // chaîne sans espace
```

Pour une chaîne avec espaces, écrire : *v=sc.nextLine();* mais *nextLine()* lit la fin de ligne contrairement aux autres méthodes ! Si un *nextInt()* précède un *nextLine()* ce dernier lit la fin de ligne qui suit l'entier. Il faut un 2ème *nextLine()* pour saisir une chaîne après.

Expressions

- Permettent de « fabriquer » des valeurs à partir d'autres valeurs : des **opérateurs** sont appliqués à des valeurs ou à des (sous-)expressions placées entre parenthèses.
- **Evaluées** pendant l'exécution. Peuvent alors être assimilées à leur valeur.
- **Expressions arithmétiques** (addition, multiplication de valeurs numériques) et **expressions logiques** (comparaisons dont la valeur est de type *boolean)*.
- Expressions arithmétiques ≈ formules mathématiques.

  Ex : *2*3*

  2/(12+1) division entière!!

  34%5 reste de la division entière (modulo)

Expressions

- Si au moins un des opérandes est de type *double* alors la valeur de l'expression est de type *double* sinon elle est de type *int* (règle simplifiée).

```
2*3.7          // résultat double
5/2            // résultat entier = 2
5.0/2          // opération réelle, résultat réel = 2.5
```

- Expressions logiques : donnent vrai ou faux.

  Utilisent les opérateurs de comparaison et les opérateurs booléens :

<	Inférieur	*23<35*
<=	Inférieur ou égal	*5<=5*
>	Supérieur	*-14>45*
>=	Supérieur ou égal	*6>=-3*
==	Égal	*6==7*

!	négation	*(34<23)||(45 6!=654)*
||	ou	
&&	et	*true && false*

Priorité des opérateurs

- Sans parenthèses : évaluation selon les priorités décroissantes suivantes :

1. opérateurs unaires : ! et -
2. opérateurs multiplicatifs : * / %
3. opérateurs additifs : + -
4. opérateurs de comparaison : < <= >= >
5. opérateurs d'égalité : == !=
6. et logique : &&
7. ou logique : ||

Avec des ( ) on évalue les expressions entre ( ) de la plus interne à la plus externe.

Ex :

*34.7+45.6*4* *// équivaut à 34.7+(45.6*4)*

true && 98>=87 || 31+8<56 *// (true && (98>=87)) || ((31+8)<56)*

- Opérateurs de même priorité : évalués de gauche à droite.

Des Opérateurs

- Incrémentation : ++ (ex: i=i+1)
- Décrémentation : -- (ex : i=i-1)
- Multiplication : *
- Division décimale : /
- Modulo : %
- Addition : +
- Soustraction : -

Affichage à l'écran

- Affichage d'un **texte** à l'écran :

  System.out.println("bonjour"); *// affiche bonjour*

- Affichage d'une **valeur, variable, expression** :

  System.out.print(expression); ***// sans retour à la ligne***

  System.out.println(expression); *// avec retour à la ligne*

  Ex: *System.out.println(2.8/4); // affiche 0.7*

  double val1=3.4,val2=2.3;

  System.out.print(val1>val2); // affiche true sans retour à la ligne

  int entier=34,diviseur=5;

  System.out.println((entier%diviseur)==1); // affiche false

Structures Conditionnelles et Répétitives

(Partie : 03)

Cours élaboré par H. EL BOUHISSI

Instruction conditionnelle

- Souvent utile de n'exécuter des instructions que quand **certaines conditions sont vérifiées**. On parle « d'instructions conditionnelles ».

Ex : algorithme qui décide si un étudiant valide ou non un module; il comporte une note d'oral de coefficient 1 et une note d'écrit de coefficient 2; la moyenne doit être supérieure ou égale à 10.

Données initialement disponibles : notes d'écrit *ne* et d'oral *no*.

moyenne ← *(2*ne+no)/3*
si *moyenne ≥10*
  alors *validation*
  sinon *non validation*
fsi

Traduction en Java :

– résultat *boolean* , - données ne et no *double* supposées déjà initialisées

```
double moyenne;
moyenne=(2*ne+no)/3;
boolean valide;
if (moyenne>=10)
   valide=true;
else
    valide=false;
```

- Les **retours à la ligne** et les **indentations** ne sont pas obligatoires mais sont essentiels pour la lisibilité du programme.

Instruction conditionnelle

- Syntaxe :

```
if (condition)
    instruction1;
else
    instruction2;
```

– *Condition est* une expression logique (de type *boolean*).

– instruction peut être remplacée par bloc; donc les instructions conditionnelles peuvent contenir des instructions conditionnelles (imbriquées).

– Conseil : toujours utiliser des blocs, même avec une seule instruction, pour faciliter la compréhension du programme.

– La partie ***else*** est facultative.

Ex:
```
if (a<b) {
    double aux=a;
    a=b;
    b=aux;
}
```

Exemple : attribuer aussi le module à un étudiant ayant entre 9 et 10 de moyenne s'il a plus de 15 à l'oral.

Algorithme

```
moyenne ← (2*ne+no)/3
si moyenne ≥10
   alors validation
   sinon
      si moyenne < 9
         alors non validation
         sinon
            si no ≥15
               alors validation
               sinon non validation
            fsi
      fsi
fsi
```

Java

```
double moyenne;
moyenne=(2*ne+no)/3;
boolean valide;
if (moyenne>=10) {
    valide=true;
} else {
    if (moyenne<9) {
        valide=false;
    } else {
        if (no>=15) {
            valide=true;
        } else {
            valide=false;
        }
    }
}
```

Instruction de choix : *switch*

- Permet d'effectuer un choix **entre plus de deux cas** sans utiliser une imbrication complexe de *if*.

Ex:

```
System.out.println("Choisir une option : 1, 2 ou 3");
Scanner sc=new Scanner(System.in);
int choix=sc.nextInt();
switch(choix) {
    case 1:
       System.out.println("Choix 1");
       break;
    case 2:
       System.out.println("Choix 2");
       break;
    case 3:
       System.out.println("Choix 3");
       break;
    default:
       System.out.println("Choix non reconnu");
       break;
}
```

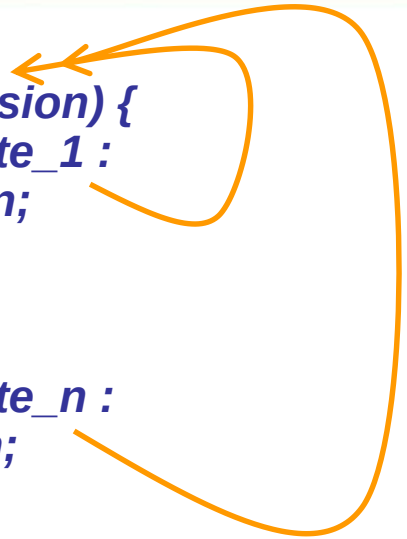
Menu avec *choix* qui vaut 3

Programme équivalent avec des *if* imbriqués :

```
System.out.println("Choisir une option : 1, 2 ou 3");
Scanner sc=new Scanner(System.in);
int choix=sc.nextInt();
if (choix==1) {
    System.out.println("Choix 1");
} else {
     if (choix==2) {
          System.out.println("Choix 2");
     } else {
          if (choix==3) {
               System.out.println("Choix 3");
          } else {
               System.out.println("Choix non reconnu");
          }
     }
}
```

- Syntaxe :

```
switch(expression) {
  case étiquette_1 :
     instruction;
     …
     break;
  ...
  case étiquette_n :
     instruction;
     ...
     break;
  default:
     instruction;
     …
     break;
}
```



- Etiquette = expression obligatoirement constante (sans variables).
- Etiquette : type compatible avec le type de l'expression (souvent *char* ou *int*).

- Chaque *case* (et le *default*) avec autant d'instructions que souhaité.
- Le *switch* avec autant de *case* que nécessaire, à condition que les étiquettes soient **uniques**.
- Au maximum un *default*.
- **Fonctionnement** :
  1) le processeur évalue l'expression qui suit le mot *switch,*
  2) le processeur cherche la première étiquette égale à la valeur trouvée et exécute toutes les instructions à partir d'elle,
  3) Si le processeur ne trouve pas une telle étiquette et si l'étiquette *default* existe, il exécute les instructions du *default*,
  4) quand le processeur rencontre l'instruction *break*, il termine l'exécution du *switch* en sautant à l'instruction qui suit l'accolade fermante.

  Oubli du *break* $\rightarrow$ le processeur continue l'exécution en passant au *case* suivant sans tester l'étiquette !

Répétition inconditionnelle (*for*)

- Elle s'écrit :

```
for (int i=1;i<=9;i++) {
   System.out.println(i*9);          // table de 9
}
```

- Dans la parenthèse qui suit le *for* on distingue 3 parties séparées par des ';' :
  - *int i=1* : déclaration et initialisation du **compteur de boucle**, représenté par la variable i, dont la portée est limitée à cette instruction for.
  - *i<=9* : **condition** (type *boolean*) que doit remplir le compteur de boucle pour que le corps de la boucle (bloc) soit **exécuté**.
  - *i++* : manière dont **évolue le compteur** (les deux plus symbolisent l'incrémentation de 1); on peut trouver aussi i-- (compteur décroissant).

Ex :
```
for (int i=5;i>=0;i--) {
   System.out.println(i);          // 5,4,3,2,1,0
}
```

- Le *for* permet aussi d'établir peu à peu un résultat de façon récurrente.

  Ex : calcul de la somme des 100 premiers carrés

```
int s=1;
for (int i=2; i<=100; i++) {
   s=s+(i*i);
}
```

- Le corps d'une boucle est une suite d'instructions qui peut donc contenir une **boucle** utilisant un compteur différent. A chaque passage dans le corps de la boucle principale la **boucle imbriquée** est exécutée en totalité.

  Ex : afficher un triangle constitué de caractères '*' dont la hauteur est donnée

```
    *
   ***
  *****
 *******
*********
```

```
for (int i=1; i<=h; i++) {
   for (int j=1; j<=h-i; j++) {
      System.out.print(' ');
   }
   for (int j=1; j<=2*i-1; j++) {
      System.out.print('*');
   }
   System.out.println();
}
```

Si h=5, pour i=1 le 1er j varie de 1 à 4 et affiche 4 espaces
le 2e j varie de 1 à 1 et affiche une *
pour i=2 le 1er j varie de 1 à 3 et affiche 3 espaces
le 2e j varie de 1 à 3 et affiche trois *
pour i=3 le 1er j varie de 1 à 2 et affiche 2 espaces
le 2e j varie de 1 à 5 et affiche cinq *
pour i=4 le 1er j varie de 1 à 1 et affiche 1 espace
le 2e j varie de 1 à 7 et affiche sept *
pour i=5 le 1er j varie de 1 à 0 et affiche 0 espace
le 2e j varie de 1 à 9 et affiche neuf *

Répétition conditionnelle

- Le *for* implique de connaître à l'avance le nombre d'itérations (valeur finale du compteur).
- On peut souhaiter répéter une instruction **tant qu'une certaine condition est remplie** alors qu'il est impossible de savoir combien d'itérations seront nécessaires.
- C'est le but de l'instruction *while* :

  while (condition) instruction;

  où condition est une expression logique (type *boolean*).

Ex :

```
System.out.println("indiquez la largeur du rectangle");
Scanner sc=new Scanner(System.in);
int l=sc.nextInt();
while (l<1) {
    System.out.println("erreur : indiquez une valeur > 0");
    System.out.println("indiquez la largeur du rectangle");
    l=sc.nextInt();
}
```