

Syntaxe du langage Java

Les commentaires

- `/*` bloc de commentaires, sur une ou plusieurs lignes `*/`
 - **Identiques à ceux existant dans le langage C**
- `//` commentaire de fin de ligne
 - **Identiques à ceux existant en C++**
- `/**` commentaire d'explication `*/`
 - **Les commentaires d'explication se placent généralement juste avant une déclaration (d'attribut ou de méthode)**
 - **Ils sont récupérés par l'utilitaire javadoc et inclus dans la documentation ainsi générée.**

Instructions, blocs et blancs

- Les instructions Java se terminent par un ;
- Les blocs sont délimités par :

{ pour le début de bloc

} pour la fin du bloc

Un bloc permet de définir un regroupement d'instructions. La définition d'une classe ou d'une méthode se fait dans un bloc.

- Les espaces, tabulations, sauts de ligne sont autorisés. Cela permet de présenter un code plus lisible.

Point d'entrée d'un programme Java

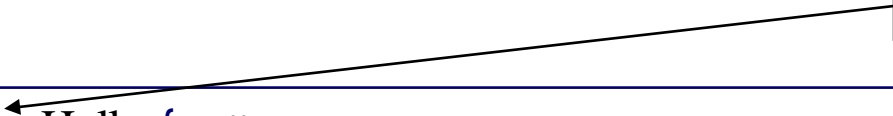
- Pour pouvoir faire un programme exécutable il faut toujours une classe qui contient une méthode particulière, la méthode « main »
- c'est le point d'entrée dans le programme : le microprocesseur sait qu'il va commencer à exécuter les instructions à partir de cet endroit

```
public static void main(String arg[ ]) {  
    //executions...  
}
```

Exemple (1)

Fichier Hello.java

La classe est l'unité de base de nos programmes. Le mot clé en Java pour définir une classe est **class**



```
public class Hello { //Accolade débutant la classe Bonjour

    public static void main(String args[]) { //Accolade débutant la méthode main

        /* Pour l'instant juste une instruction */

        System.out.println("Hello World");

    } //Accolade fermant la méthode main

} //Accolade fermant la classe Hello
```

Exemple (2)

Fichier Hello.java

```
public class Hello {  
    public static void main(String args[]) {  
        System.out.println("Hello World");  
    }  
}
```

Accolades délimitant le début et la fin de la définition de la class Bonjour

Accolades délimitant le début et la fin de la méthode main

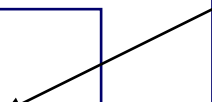
Les instructions se terminent par des ;

Exemple (3)

Fichier Hello.java

```
public class Hello {  
    public static void main(String args[]) {  
        System.out.println("Hello World");  
    }  
}
```

Une méthode peut recevoir des paramètres. Ici la méthode main reçoit le paramètre args qui est un tableau de chaîne de caractères.



Identificateurs (1)

- On a besoin de nommer les classes, les variables, les constantes, etc. ; on parle d'identificateur.
- Les identificateurs commencent par une lettre, _ ou \$
Attention : Java distingue les majuscules des minuscules
- Conventions sur les identificateurs :
 - Si plusieurs mots sont accolés, mettre une majuscule à chacun des mots sauf le premier. (**camelCase**)
 - exemple : uneJolieFenetre
 - La première lettre est majuscule pour les classes et les interfaces (**PascalCase**)
 - exemples : UneJolieFenetre
 - Tous dans la même case et les mots sont séparés par un tiret du bas (**snake_case**)
 - exemple: une_jolie_fenetre

Identificateurs (2)

- Conventions sur les identificateurs :
 - **Classes** : PascalCase
 - **Méthodes, attributs et variables** : camelCase
 - exemples : setLongueur, i, uneFenetre
 - **Constantes**: SCREAMING_SNAKE_CASE
 - exemple : LONGUEUR_MAX

Les mots réservés de Java

<code>abstract</code>	<code>default</code>	<code>goto</code>	<code>null</code>	<code>synchronized</code>
<code>boolean</code>	<code>do</code>	<code>if</code>	<code>package</code>	<code>this</code>
<code>break</code>	<code>double</code>	<code>implements</code>	<code>private</code>	<code>throw</code>
<code>byte</code>	<code>else</code>	<code>import</code>	<code>protected</code>	<code>throws</code>
<code>case</code>	<code>extends</code>	<code>instanceof</code>	<code>public</code>	<code>transient</code>
<code>catch</code>	<code>false</code>	<code>int</code>	<code>return</code>	<code>true</code>
<code>char</code>	<code>final</code>	<code>interface</code>	<code>short</code>	<code>try</code>
<code>class</code>	<code>finally</code>	<code>long</code>	<code>static</code>	<code>void</code>
<code>continue</code>	<code>float</code>	<code>native</code>	<code>super</code>	<code>volatile</code>
<code>const</code>	<code>for</code>	<code>new</code>	<code>switch</code>	<code>while</code>

Les types de bases (1)

- En Java, tout est objet sauf les types de base.
- Il y a huit types de base :
 - un type booléen : **boolean** avec les valeurs associées **true** et **false**
 - un type pour représenter un caractère : **char**
 - quatre types pour représenter les entiers de divers taille :
 - **byte**
 - **short**
 - **int**
 - **long**
 - deux types pour représenter les réelles :
 - **float**
 - **double**

Les types de bases (2) : les entiers

- Les entiers (avec signe)
 - **byte** : codé sur 8 bits, peuvent représenter des entiers allant de -2^7 à $2^7 - 1$ (-128 à +127)
 - **short** : codé sur 16 bits, peuvent représenter des entiers allant de -2^{15} à $2^{15} - 1$ (-32.768 à 32.767)
 - **int** : codé sur 32 bits, peuvent représenter des entiers allant de -2^{31} à $2^{31} - 1$ (-2.147.483.648 à 2.147.483.647)
 - **long** : codé sur 64 bits, peuvent représenter des entiers allant de -2^{63} à $2^{63} - 1$ (-9.22e+18 à 9.22e+18)

Les types de bases (3) : les entiers

- Notation

- 2 entier **normal** en base décimal
- 2L entier au format **long** en base décimal
- **0**10 entier en valeur **octale** (base 8)
- **0x**F entier en valeur **hexadécimale** (base 16)

- Opérations sur les entiers

- opérateurs arithmétiques $+$, $-$, $*$
- $/$: division entière si les 2 arguments sont des entiers
- $\%$: reste de la division entière
 - exemples :
 - $15 / 4$ donne 3
 - $15 \% 2$ donne 1

Les types de bases (4) : les entiers

- Opérations sur les entiers (suite)
 - les opérateurs d'incrément `++` et de décrémentation `--`
 - ajoute ou retranche 1 à une variable
`int n = 12;`
`n ++;` // Maintenant n vaut 13
 - `n++`; « équivalent à » `n = n+1;`
`n--`; « équivalent à » `n = n-1;`
 - `8++`; est une instruction illégale
 - peut s'utiliser de manière suffixée : `++n`. La différence avec la version préfixée se voit quand on les utilisent dans les expressions.
En version suffixée la (dé/inc)rémentation s'effectue en premier

```
int m=7; int n=7;  
int a=2 * ++m; //a vaut 16, m vaut 8  
int b=2 * n++; //b vaut 14, n vaut 8
```

Les types de bases (5) : les réels

- Les réels
 - **float** : codé sur 32 bits, peuvent représenter des nombres allant de -10^{35} à $+10^{35}$
 - **double** : codé sur 64 bits, peuvent représenter des nombres allant de -10^{400} à $+10^{400}$
- Notation
 - 4.55**D** réel double
 - 4.55**F** réel simple

Les types de bases (6) : les réels

- Les opérateurs
 - opérateurs classiques $+$, $-$, $*$, $/$
 - attention pour la division :
 - si l'un des termes de la division est un réel, la division retournera un réel
 - puissance : utilisation de la méthode `pow` de la classe `Math`.
 - `double y = Math.pow(x, a)` équivalent à x^a , `x` et `a` étant de type `double`

Les types de bases (7) : les booléens

- **boolean** contient soit vrai (**true**) soit faux (**false**)
- Les opérateurs logiques de comparaisons
 - Egalité : opérateur `==`
 - Différence : opérateur `!=`
 - strictement supérieur et strictement inférieur à :
opérateurs `>` et `<`
 - supérieur et inférieur ou égal :
opérateurs `>=` et `<=`

Les types de bases (8) : les booléens

- Notation

```
boolean x;
```

```
x= true;
```

```
x= false;
```

```
x= (5==5); // l'expression (5==5) est évaluée et la valeur est  
affectée à x qui vaut alors vrai
```

```
x= (5!=4); // x vaut vrai, ici on obtient vrai si 5 est différent  
de 4
```

```
x= (5>5); // x vaut faux, 5 n'est pas supérieur strictement à 5
```

```
x= (5<=5); // x vaut vrai, 5 est bien inférieur ou égal à 5
```

Les types de bases (9) : les booléens

- Les autres opérateurs logiques
 - et logique : **&&**
 - ou logique : **||**
 - non logique : **!**
 - Exemples : si a et b sont 2 variables booléennes

boolean a, b, c;

a= true;

b= false;

c= (a && b); // c vaut false

c= (a || b); // c vaut true

c= !(a && b); // c vaut true

c=!a; // c vaut false

Les types de bases (10) : les caractères

- Les caractères
 - **char** : contient une seule lettre
 - le type char désigne des caractères en représentation Unicode
 - Codage sur 2 octets contrairement à ASCII/ANSI codé sur 1 octet. Le codage ASCII/ANSI est un sous-ensemble d'Unicode
 - Notation hexadécimale des caractères Unicode de ' \u0000 ' à ' \uFFFF '.
 - Plus d'information sur Unicode à : www.unicode.org
 - **String:**
 - Pas un type de base
 - Objet

Les types de bases (11) : les caractères

- Notation

char a, b,c; // a, b et c sont des variables du type char

a='a'; // a contient la lettre 'a'

b= '\u0022' //b contient le caractère *guillemet* : "

c=97; // x contient le caractère de rang 97 : 'a'

Les types de bases (12)

exemple et remarque

```
int x = 0;  
int y = 0;  
float z = 3.1415F;  
double w = 3.1415;  
long t = 99L;  
boolean test = true;  
char c = 'a';
```

- Remarque importante :
 - Java exige que toutes les variables soient définies et initialisées. Le compilateur sait déterminer si une variable est susceptible d'être utilisée avant initialisation et produit une erreur de compilation.

Les structures de contrôles (1)

- Les structures de contrôle classiques existent en Java :
 - **if, else**
 - **switch, case, default, break**
 - **for**
 - **while**
 - **do, while**

Les structures de contrôles (2) : if / else

- Instructions conditionnelles

```
if ( condition ) {  
    bloc d'instructions  
}
```

// condition doit être un booléen ou renvoyer une valeur booléenne

- Else:

```
if ( condition ) {  
    1er bloc d'instructions  
} else if ( condition2 ) {  
    2ème bloc d'instruction  
} else {  
    3ème bloc d'instruction  
}
```


Les structures de contrôles

Max.java

```
Scanner scan = new Scanner(System.in);

System.out.print( "Veuillez saisir un premier entier : " );
int a = scan.nextInt();

System.out.print( "Veuillez saisir un second entier : " );
int b = scan.nextInt();

int result = a + b;
System.out.printf( "La somme de %d et de %d vaut %d\n", a, b, result );
```

Les structures de contrôles (4) : while

- Boucles indéterminées
 - On veut répéter une ou plusieurs instructions un nombre indéterminés de fois : on répète l'instruction ou le bloc d'instruction tant que une certaine condition reste vraie
 - nous avons en Java une première boucle while (tant que)
 - **while** (*condition*) {
 bloc d'instructions
}
 - les instructions dans le bloc sont répétées tant que la condition reste vraie.
 - On ne rentre jamais dans la boucle si la condition est fausse dès le départ

Les structures de contrôles (5) : while

- Boucles indéterminées
 - un autre type de boucle avec le while:
 - **do** {
 bloc d'instructions
} **while** (*condition*)
 - les instructions dans le bloc sont répétées tant que la condition reste vraie.
 - On rentre toujours au moins une fois dans la boucle : la condition est testée en fin de boucle.

Les structures de contrôles (6) : while

Facto1.java

```
System.out.print( "Veuillez saisir un entier : " );
int n = scan.nextInt();
int result = 1;
int i = 1;

while (i <= n){
    result *= i;
    i++;
}

System.out.printf("le factoriel de %d vaut %d\n", n, result);
```

Les structures de contrôles (7) : for

- Boucles déterminées
 - On veut répéter une ou plusieurs instructions un nombre déterminés de fois : on répète l'instruction ou le bloc d'instructions pour un certain nombre de pas.
 - La boucle for

```
for (int i = 1; i <= 10; i++) {  
    System.out.println(i);  
}
```

Les structures de contrôles (8) : for

Facto2.java

```
System.out.print( "Veuillez saisir un entier : " );  
int n = scan.nextInt();  
int result = 1;  
for (int i = 2; i <= n ; i++) result *= i  
System.out.printf("le factoriel de %d vaut %d\n", n, result);
```

Les structures de contrôles (9) : switch

- Sélection multiples
 - l'utilisation de if / else peut s'avérer lourde quand on doit traiter plusieurs sélections et de multiples alternatives
 - pour cela existe en Java le **switch** / **case** assez identique à celui de C/C++
 - La valeur sur laquelle on teste doit être un char ou un entier (à l'exclusion d'un **long**).
 - L'exécution des instructions correspondant à une alternative commence au niveau du **case** correspondant et se termine à la rencontre d'une instruction **break** ou arrivée à la fin du **switch**

Les structures de contrôles (10) : switch

Alternative.java

```
System.out.print( "Veuillez saisir un entier : " );
int nb = scan.nextInt();
switch(nb) {
    case 1:
        System.out.println("Un"); break;
    case 2:
        System.out.println("Deux"); break;
    default:
        System.out.println("Autre nombre"); break;
}
```


Les tableaux (1)

- Les tableaux permettent de stocker plusieurs valeurs de même type dans une variable.
 - Les valeurs contenues dans la variable sont repérées par un indice
 - En langage java, les tableaux sont des objets
- Déclaration
 - `int tab[];`
`String chaines[];`
- Création d'un tableau
 - `tab = new int[20]; // tableau de 20 int`
 - `chaines = new String[100]; // tableau de 100 chaine`

Les tableaux (2)

- Le nombre d'éléments du tableau est mémorisé. Java peut ainsi détecter à l'exécution le dépassement d'indice et générer une exception.
- Mot clé `length` permet de récupérer la taille du tableau
 - **`nomTableau.length`**
`int taille = tab.length; //taille vaut 20`
- Comme en C/C++, les indices d'un tableau commencent à '0'. Donc un tableau de taille 100 aura ses indices qui iront de 0 à 99.

Les tableaux (3)

- Initialisation

```
tab[0]=1;
```

```
tab[1]=2; //etc.
```

```
noms[0] = new String( "Toto");
```

```
noms[1] = new String( "Tata");//etc
```

- Création et initialisation simultanées

```
String noms [ ] = { "Toto", "Tata" };
```

Les tableaux (4)

Tab1.java

```
public class Tab1 {  
    public static void main (String args[]) {  
        int tab[ ] ;  
        tab = new int[4];  
        tab[0]=5;  
        tab[1]=3;  
        tab[2]=7;  
        tab[3]=tab[0]+tab[1];  
    }  
}
```

La classe String (1)

- Attention ce n'est pas un type de base. Il s'agit d'une classe défini dans l'API Java (Dans le package java.lang)

String s="aaa"; // s contient la chaîne "aaa »

String s=**new String**("aaa"); // identique à la ligne précédente

- La concaténation

- l'opérateur **+** entre 2 String les concatène :

String str1 = "Bonjour ! ";

String str2 = "Comment vas-tu ?";

String str3 = str1 + str2; /* Concaténation de chaînes : str3 contient
" Bonjour ! Comment vas-tu ?« */

La classe String (2)

- Longueur d'un objet String :
 - méthode `int length()` : renvoie la longueur de la chaîne
`String str1 = "bonjour";`
`int n = str1.length(); // n vaut 7`
- Sous-chaînes
 - méthode `String substring(int debut, int fin)`
 - extraction de la sous-chaine depuis la position `debut` jusqu'à la position `fin` non-comprise.
`String str2 = str1.substring(0,3); // str2 contient la valeur "bon"`
 - le premier caractère d'une chaîne occupe la position 0
 - le deuxième paramètre de `substring` indique la position du premier caractère que l'on ne souhaite pas copier

La classe String (3)

- Récupération d'un caractère d'une chaîne
 - méthode `char charAt(int pos)` : renvoie le caractère situé à la position `pos` dans la chaîne de caractère à laquelle on envoie se message

```
String str1 = "bonjour";
char unJ = str1.charAt(3); // unJ contient le caractère 'j'
```
- Modification des objets String
 - Les String sont **inaltérables** en Java : on ne peut modifier individuellement les caractères d'une chaîne.
 - Par contre il est possible de modifier le contenu de la variable contenant la chaîne (la variable ne référence plus la même chaîne).

```
str1 = str1.substring(0,3) + " soir"; /* str1 contient maintenant la chaîne
"bonsoir" */
```

La classe String (4)

- Les chaînes de caractères sont des objets :
 - pour tester si 2 chaînes sont égales il faut utiliser la méthode **boolean equals(String str)** et non **==**
 - pour tester si 2 chaînes sont égales à la casse près il faut utiliser la méthode **boolean equalsIgnoreCase(String str)**

```
String str1 = "BonJour";
```

```
String str2 = "bonjour"; String str3 = "bonjour";
```

```
boolean a, b, c, d;
```

```
a = str1.equals("BonJour"); // a contient la valeur true
```

```
b = (str2 == str3); // b contient la valeur false
```

```
c = str1.equalsIgnoreCase(str2); // c contient la valeur true
```

```
d = "bonjour".equals(str2); // d contient la valeur true
```


La classe String (5)

- Quelques autres méthodes utiles
 - `boolean startsWith(String str)` : pour tester si une chaîne de caractère commence par la chaîne de caractère `str`
 - `boolean endsWith(String str)` : pour tester si une chaîne de caractère se termine par la chaîne de caractère `str`

```
String str1 = "bonjour ";
```

```
boolean a = str1.startsWith("bon");//a vaut true
```

```
boolean b = str1.endsWith("jour");//b vaut true
```

La classe Math

- Les fonctions mathématiques les plus connues sont regroupées dans la classe Math qui appartient au package java.lang
 - les fonctions trigonométriques
 - les fonctions d'arrondi, de valeur absolue, ...
 - la racine carrée, la puissance, l'exponentiel, le logarithme, etc...
- Ce sont des méthodes de classe (static)
double calcul = **Math.sqrt**(**Math.pow**(5,2) + **Math.pow**(7,2));
double **sqrt**(double x) // racine carrée de x
double **pow**(double x, double y) // x puissance y