LU2IN002
TYPES DE BASE, VARIABLES, BOUCLES, CONDITIONNELLES
GUIDE DE SURVIE EN JAVA

Vincent Guigue

SCIENCES SORBONNE UNIVERSITÉ
LIP6
MLIA Machine Learning & Deep Learning for Information Access

Types des variables de base en JAVA

- Entier, réel, booléen, caractère: ces types sont disponibles de base en JAVA avec les opérateurs les plus courants.
  `int, double, boolean, char, byte, short, long, float`

Types des variables de base en JAVA

- Entier, réel, booléen, caractère: ces types sont disponibles de base en JAVA avec les opérateurs les plus courants.

  `int`, `double`, `boolean`, `char`, `byte`, `short`, `long`, `float`

La plupart des types et syntaxes associées sont comparables au C/C++...

Sauf le booléen.

> Le booléen vaut `true`/`false` et n'est pas convertible en entier

SCIENCES SORBONNE UNIVERSITÉ

- Entier, réel, booléen, caractère: ces types sont disponibles de base en JAVA avec les opérateurs les plus courants.

  `int, double, boolean, char, byte, short, long, float`

  La plupart des types et syntaxes associées sont comparables au C/C++...

  Sauf le booléen.

Le booléen vaut `true/false` et n'est pas convertible en entier

- Déclaration

```
1 int i; // declaration de i
2 System.out.println(i); // => 0
3 double d = 2.6;
4 boolean b = true; // ou false
5 char c = 'a';
```

Types des variables de base en JAVA

SCIENCES SORBONNE UNIVERSITÉ

- Entier, réel, booléen, caractère: ces types sont disponibles de base en JAVA avec les opérateurs les plus courants.

  `int, double, boolean, char, byte, short, long, float`

  La plupart des types et syntaxes associées sont comparables au C/C++...

  Sauf le booléen.

Le booléen vaut `true/false` et n'est pas convertible en entier

- Déclaration

```
1 int i; // declaration de i
2 System.out.println(i); // => 0
3 double d = 2.6;
4 boolean b = true; // ou false
5 char c = 'a';
1 // operations de base: + - / * ...
2 int j = i+2;
3 int k = 1/2; //=0 Attention a la division entiere
```

SCIENCES SORBONNE UNIVERSITÉ

Gestion des chaines de caractères

String n'est pas un type de base, c'est un objet qui se comporte différemment des types de base... Mais c'est une classe complètement intégrée à JAVA et son caractère immutable la rapproche très nettement d'un type de base.

```
1 String s = "toto"; // creation d'une chaine de caracteres
2 s = s + " va a la fac";
3 System.out.println(s); // affichage de s dans la console
```

SCIENCES SORBONNE UNIVERSITÉ

Gestion des chaines de caractères

String n'est pas un type de base, c'est un objet qui se comporte différemment des types de base... Mais c'est une classe complètement intégrée à JAVA et son caractère immutable la rapproche très nettement d'un type de base.

```
String s = "toto"; // creation d'une chaine de caracteres
s = s + " va a la fac";
System.out.println(s); // affichage de s dans la console
```

Ne pas confondre l'**objet** String et l'affichage dans la console.

Gestion des chaines de caractères

String n'est pas un type de base, c'est un objet qui se comporte différemment des types de base... Mais c'est une classe complètement intégrée à JAVA et son caractère immutable la rapproche très nettement d'un type de base.

```java
String s = "toto"; // creation d'une chaine de caracteres
s = s + " va a la fac";
System.out.println(s); // affichage de s dans la console
```

Ne pas confondre l'**objet** String et l'affichage dans la console.

Les **possibilités sont nombreuses**: extraction de sous-chaines (substring), division en plusieurs chaines (split), recherche de caractères, construction de nouvelles chaines à partir d'expressions régulières (replace)... Toute la documentation sur: http://docs.oracle.com/javase/7/docs/api/java/lang/String.html

SCIENCES SORBONNE UNIVERSITÉ

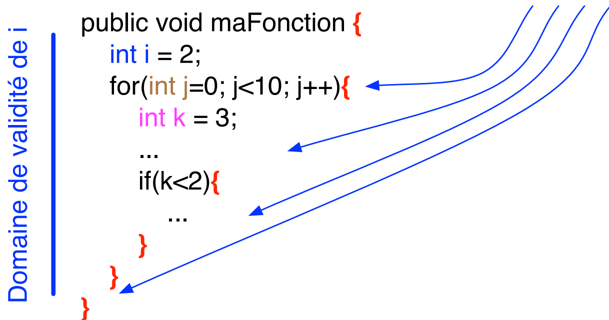
2 choses à retenir sur les String

1. **Les chaines sont immutables**: modifier une chaine existante est impossible, il faut créer une nouvelle chaine qui est une modification de l'ancienne. Cela rend la classe peu efficiente dans certain cas... Et il faut alors se tourner vers des objets plus évolués (`StringBuffer` notamment)
2. **Ne pas utiliser == avec les** `String` mais toujours la méthode `.equals`. Les deux versions compilent mais la première donnera régulièrement des résultats faux (que nous expliquerons plus tard).

```java
String s1 = "toto";
String s2 = "titi";
if( s1.equals(s2) )
    System.out.println("les chaines sont identiques");
else
    System.out.println("les chaines sont differentes");
```

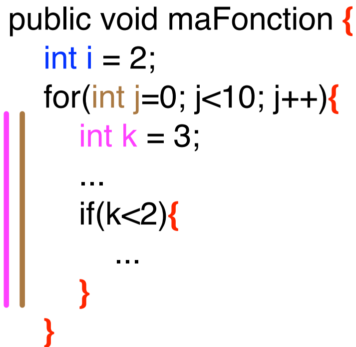
- une fonction est un bloc,
- une boucle ou une conditionnelle forme également un bloc,
- les blocs sont repérés par des accolades: **{...}**

Les variables **déclarée** dans un bloc sont détruites en sortant du bloc.



- une fonction est un bloc,
- une boucle ou une conditionnelle forme également un bloc,
- les blocs sont repérés par des accolades: **{...}**

Les variables **déclarée** dans un bloc sont détruites en sortant du bloc.



JAVA, un langage typé

Les types sont très importants en JAVA: le compilateur vérifie toujours les types des différentes variables

- Certaines conversions sont **implicites**:

```java
double d = 1; double d2 = i; // avec i un int existant
```

Il est possible de transformer n'importe quel type de base en `String` (l'affichage est donc facile)

```java
String s = "mon message "+1.5+" "+d;
```

SCIENCES SORBONNE UNIVERSITÉ

JAVA, un langage typé

Les types sont très importants en JAVA: le compilateur vérifie toujours les types des différentes variables

- Certaines conversions sont **implicites**:

```java
double d = 1; double d2 = i; // avec i un int existant
```

Il est possible de transformer n'importe quel type de base en `String` (l'affichage est donc facile)

```java
String s = "mon␣message␣"+1.5+"␣"+d;
```

- Certaines conversions doivent être **explicites**

```java
int i = (int) 2.4;
```

il y a une perte d'information liée à la conversion; JAVA ne tolère pas la conversion implicitement, il faut que le programmeur la demande explicitement (pour être sûr que la perte d'information est souhaitée).

SCIENCES SORBONNE UNIVERSITÉ

JAVA, un langage typé

Les types sont très importants en JAVA: le compilateur vérifie toujours les types des différentes variables

- Certaines conversions sont **implicites**:

```java
double d = 1; double d2 = i; // avec i un int existant
```

Il est possible de transformer n'importe quel type de base en `String` (l'affichage est donc facile)

```java
String s = "mon message "+1.5+" "+d;
```

- Certaines conversions doivent être **explicites**

```java
int i = (int) 2.4;
```

- Conversions **impossibles**

```java
int i = (int) true; // conversion impossible des booleens
                    // vers le domaine numerique
                    // => ERREUR de compilation
```

SCIENCES SORBONNE UNIVERSITÉ

- Syntaxe directement inspirée du C/C++
- Déclaration:

visibilité	type de retour	nom de la fonction	arguments
`public`	`double`	`monCalcul`	`(int arg1, String arg2)`

```
1   public double monCalcul(int i, double d){
```

- Syntaxe directement inspirée du C/C++
- Déclaration:

visibilité	type de retour	nom de la fonction	arguments
`public`	`double`	`monCalcul`	`(int arg1, String arg2)`

- Calculs divers

```
1  public double monCalcul(int i, double d){
2    double resultat = 100. + i * d;
```

- Syntaxe directement inspirée du C/C++
- Déclaration:

visibilité	type de retour	nom de la fonction	arguments
`public`	`double`	`monCalcul`	`(int arg1, String arg2)`

- Calculs divers
- Retour (obligatoire si autre que `void`)

```java
public double monCalcul(int i, double d){
    double resultat = 100. + i * d;
    return resultat;
}
```

SCIENCES SORBONNE UNIVERSITÉ

- Syntaxe directement inspirée du C/C++
- Déclaration:

visibilité	type de retour	nom de la fonction	arguments
`public`	`double`	`monCalcul`	`(int arg1, String arg2)`

- Calculs divers
- Retour (obligatoire si autre que `void`)
- Bonne pratique: on n'utilise pas les arguments comme variable tampon !! $\Rightarrow$ argument = constante

```
1 public double monCalcul(int i, double d){
2   double resultat = 100. + i * d;
3   return resultat;
4 }
```

Opérateurs classiques (par ordre de priorité)

<table>
<tr><td>opérateurs postfixés</td><td>[ ]   .   expr++   expr--</td></tr>
<tr><td>opérateurs unaires</td><td>++expr   --expr   +expr   -expr   ~   !</td></tr>
<tr><td>création ou cast</td><td>new   (type)   expr</td></tr>
<tr><td>opérateurs multiplicatifs</td><td>*   /   %</td></tr>
<tr><td>opérateurs additifs</td><td>+   -</td></tr>
<tr><td>décalages</td><td><<   >>   >>></td></tr>
<tr><td>opérateurs relationnels</td><td><   >   <=   >=</td></tr>
<tr><td>opérateurs d'égalité</td><td>==   !=</td></tr>
<tr><td>et bit à bit</td><td>&</td></tr>
<tr><td>ou exclusif bit à bit</td><td>^</td></tr>
<tr><td>ou (inclusif) bit à bit</td><td>|</td></tr>
<tr><td>et logique</td><td>&&</td></tr>
<tr><td>ou logique</td><td>||</td></tr>
<tr><td>opérateur conditionnel</td><td>?   :</td></tr>
<tr><td>affectations</td><td>=   +=   -=   *=   /=   %=   &=   ^= |=   <<=   >>=   >>>=</td></tr>
</table>

SCIENCES SORBONNE UNIVERSITÉ

- Syntaxe du *Si, ... Alors:*

```
int i=8;
if(i > 5){
    // code a effectuer dans ce cas
}
else{ // le else est facultatif
    // Code a effectuer sinon
}
```

- Syntaxe du *Si, ... Alors:*

```
1 int i=8;
2 if(i > 5){
3       // code a effectuer dans ce cas
4 }
5 else{ // le else est facultatif
6       // Code a effectuer sinon
7 }
```

- En cas de clauses multiples:

```
1 switch(i){
2 case 1:
3    // Code a effectuer si i == 1
4    break; // sinon le reste du code est AUSSI effectue
5 case 2: //
6     // Code a effectuer si i == 2
7    break;
8 default : // Si on n'est passe nulle part ailleurs
9 }
```

La définition des boucles est identiques au C/C++

- Syntaxes: 2 options (principales)
  Pour i allant de 0 à 9, faire...

```
1 int i;
2 for(i=0; i<10; i++){// i prend les valeurs 0 a 9 =
3                     // 10 iterations
4 // code a effectuer 10 fois
5 }
```

SCIENCES SORBONNE UNIVERSITÉ

La définition des boucles est identiques au C/C++

- Syntaxes: 2 options (principales)

  Pour i allant de 0 à 9, faire...

```
1 int i;
2 for(i=0; i<10; i++){// i prend les valeurs 0 a 9 =
3                     // 10 iterations
4 // code a effectuer 10 fois
5 }
```

  Tant que i inférieur à 10, faire...

```
1 int i = 0;
2 while(i<10){// i prend les valeurs 0 a 9 =
3                     // 10 iterations
4     // code a effectuer 10 fois
5     i++; // ne pas oublier, sinon boucle infinie !
6 }
```

- D'autres syntaxes sont possibles : *do...while* etc...

SCIENCES SORBONNE UNIVERSITÉ

3 types d'interruptions de boucles

- return : l'interruption **la plus forte**. Coupe l'exécution de la méthode (**sort de la fonction**, pas seulement la boucle).

```
// le modulo par 5 peut-il retrouner un entier >=5?
public void maFonction(){
    for(int i=0; i<10; i++){
        if(i%5>4){
            System.out.println("C'est tres etrange");
            return;
        }
    }
    System.out.println("l'operation modulo 5 retourne"+
        "toujours un entier inferieur a 5");
}
```

SCIENCES SORBONNE UNIVERSITÉ

3 types d'interruptions de boucles

- return
- break : l'interruption de boucle

```
1  // 6 fait-il parti des multiples de 2?
2  public void maFonction(){
3      boolean found = true;
4      for(int i=0; i<10; i++){
5          if(i * 2 == 6){
6              found = true;
7              break; // pas besoin d'aller plus loin
8          }
9      }
10     if(found)
11         System.out.println("6 fait parti des multiples de 2");
12 }
```

3 types d'interruptions de boucles

- return
- break
- continue: sauter une itération de boucle

```
1 // afficher 3./i pour i variant de -10 a 10
2 // il faut penser a sauter le cas 0 qui provoque un probleme
3 public void maFonction(){
4     for(int i=-10; i<=10; i++){// -10 et 10 inclus
5         if(i == 0)
6             continue;
7         System.out.println("3./"+i+" = "+(3./i) );
8     }
9 }
```

Ces instructions rendent le code plus lisible en limitant notamment le nombre de blocs imbriqués.