

Cours de Programmation Orientée Objet JAVA

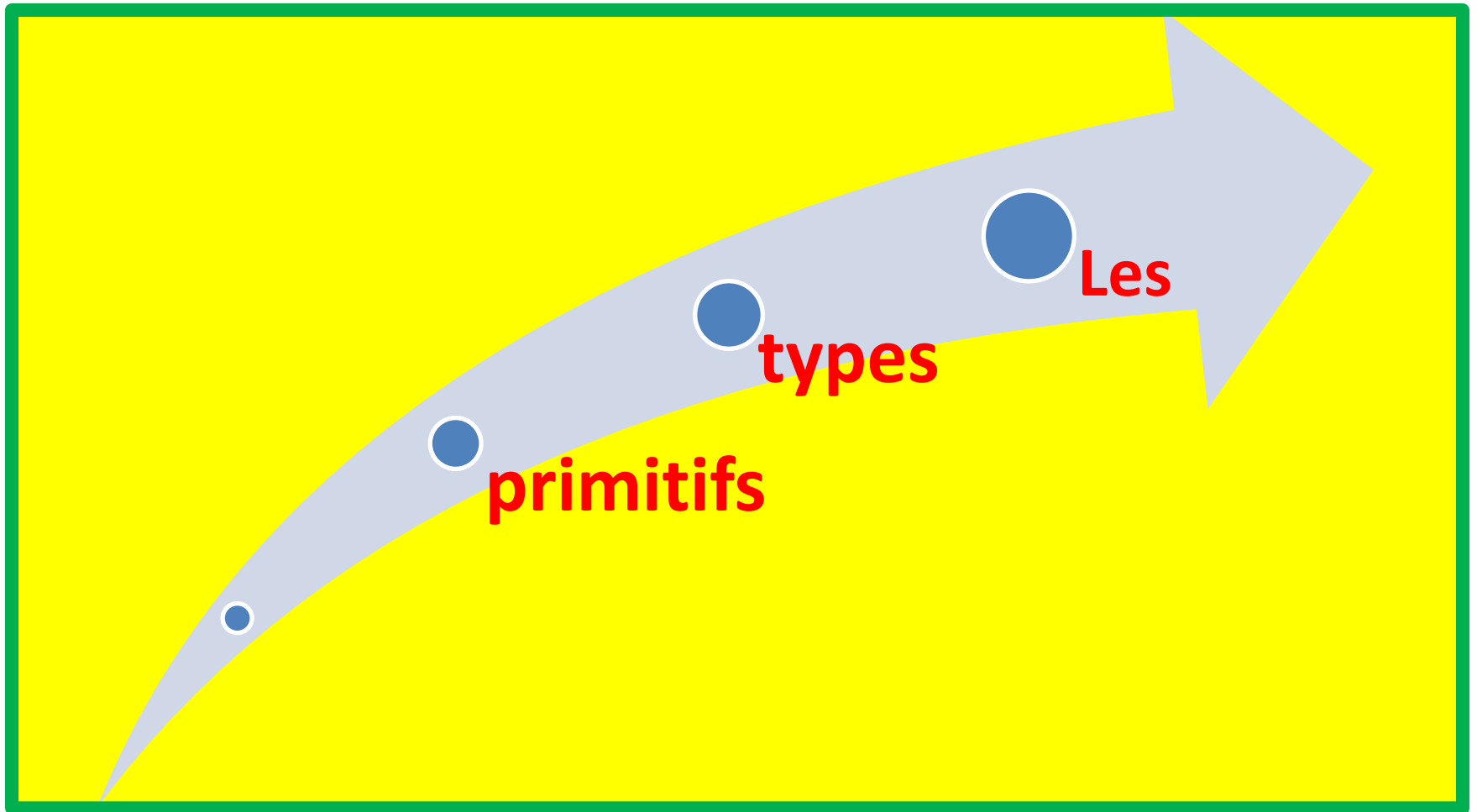
Demba SOW

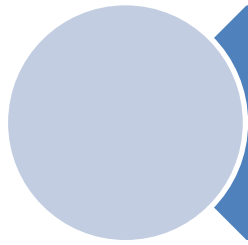
*Docteur en
Mathématiques et en
Cryptologie*

L.A.C.G.A.A.

F.S.T. / U.C.A.D.

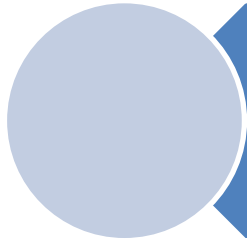






Les types primitifs

- Java dispose d'un certain nombre de types de base dits *primitifs*, permettant de manipuler des entiers, des flottants, des caractères et des booléens.
 - Ce sont les seuls types du langage qui ne sont pas des classes.
- Les types primitifs (au nombre de 8) se répartissent en quatre grandes catégories selon la nature des informations qu'ils permettent de manipuler:
 - **Nombres entiers,**
 - **Nombres flottants,**
 - **caractères**
 - **booléens**



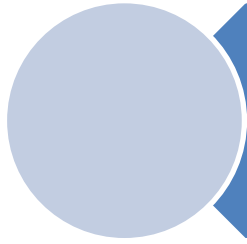
Le type entier

Ils servent à représenter les nombres entiers relatifs, ils sont au nombre de 4.

BYTE (1 octet) **SHORT** (2 octets) **INT** (4 octets) **LONG** (8 octets)

- **byte 8 bits** : -128 à 127
- **short 16 bits** : -32768 à 32767
- **int 32 bits** : -2147483648 à 2147483647
- **long 64 bits** : -9223372036854775808 à 9223372036854775807

Les types élémentaires ont une taille identique quelque soit la plate-forme d'exécution.



Le type flottant

Ils permettent de représenter , de manière approchée, une partie des nombres réels. Java prévoit deux types de flottants correspondant chacun à des emplacements mémoire de tailles différentes: *float* et *double*.

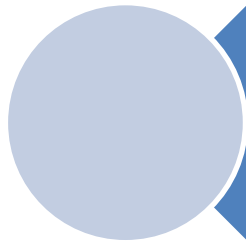
Double : 64 bits

Float : 32 bits

Exemple :

float pi = 3.141**f**; // nécessaire de suffixer par la lettre f sinon erreur de compilation

double v = 3**d** ; // suffixe d non nécessaire 3d = 3



Le type caractère

■ **CHAR** (2 octets) et **STRING**

■ **char** : caractère isolé

- codage unicode sur **16 bits** non signé
- expression littéral char entre apostrophes

'+', 'a', '\t', '\u???'

- Une variable de type caractère se déclare:

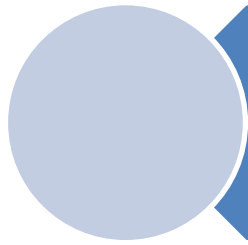
`char c1; char c1,c2 ;` // c1 et c2 sont 2 variables caractère

■ **String** : chaîne de caractères

- une expression littérale de type *String*

`String s1 = "bonjour", s2 = new String();`

■ **ATTENTION** : String est une classe du paquetage **java.lang** .



Le type booléen

Ce type sert à représenter une valeur logique du type *vrai/faux*

■ BOOLEAN

■ Deux états: **true** / **false**

■ Exemples:

boolean ok=false;

if (n<p) ... //n<p est expression booléenne valant vrai ou faux

boolean ordonne ; // déclaration d'une variable de type booléenne

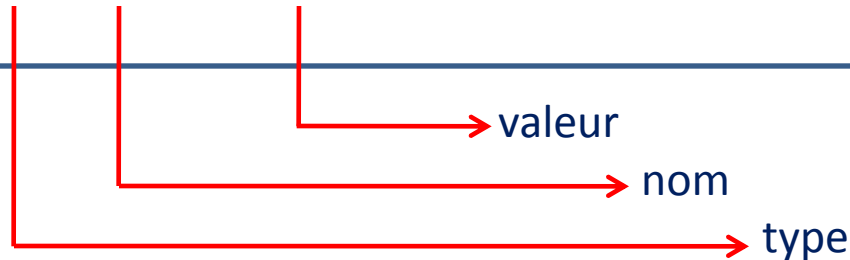
.....

ordonne =n<p; //ordonne reçoit la valeur de l'expression $n < p$

Initialisation des variables (1/2)

Exemple :

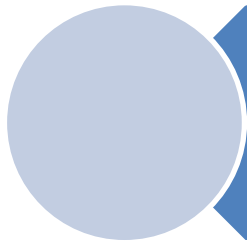
```
int nombre;           // déclaration de la variable nombre
nombre = 100;         //initialisation de la variable nombre
int nombre = 100;    //déclaration et initialisation de la variable nombre
```



= opérateur d'affectation
(associatif de droite à gauche)

■ Remarque :

- Une variable manipulée **dans une méthode** (variable locale) ou **un bloc** devra **toujours** être initialisée avant toute utilisation.
- La déclaration d'une variable **réserve de la mémoire** pour stocker sa valeur.



Initialisation des variables (2/2)

- En java, toute variable appartenant à un objet (définie comme étant un attribut de l'objet) est initialisée avec une **valeur par défaut** en accord avec son **type** au moment de la création.
- Cette initialisation ne s'applique pas aux **variables locales** des méthodes de la classe (cf: remarque précédente).

TYPE	VALEUR PAR DÉFAUT
boolean	false
byte, short, int, long	0
float, double	0.0
char	u\000
classe	null

Utilisation des variables

```
package essai ;
```

```
public class UtilVariable
```

```
{ String chaine ; // variable de type (class ) String, valeur par défaut null
```

```
double solde ; // valeur par défaut 0.0 assignée à solde
```

```
public static void main(String [ ] args)
```

```
{
```

```
    System.out.println(" valeur de solde =" +solde);
```

```
}
```

```
public void affiche( )
```

```
{ chaine = new String(" bonjour" ); // objet de type String initialisée
```

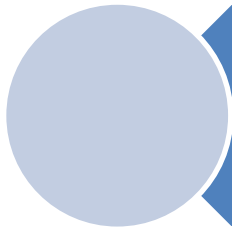
```
long nombre ; // erreur de compilation car nombre non initialisée
```

```
System.out.println(" valeur de nombre= "+nombre);
```

```
}
```

```
}
```

new pour
créer un objet



variables finales (le mot clé final)

Java permet de déclarer que la valeur d'une variable ne doit pas être modifiée pendant l'exécution du programme

```
package essai ;
```

```
public class VariableFinale
```

```
{ final long NOMBRE ;
```

```
  final double MAX = 100 ; // variable finale = constante
```

```
  public static void main(String [ ] args)
```

```
  {
```

```
    System.out.println(" utilisation de variable constante" );
```

```
  }
```

```
  public void affiche( )
```

```
  { NOMBRE = 1000 ; // initialisation différée de la variable NOMBRE
```

```
    System.out.println(" valeur de MAX= "+MAX);
```

```
  }
```

```
}
```

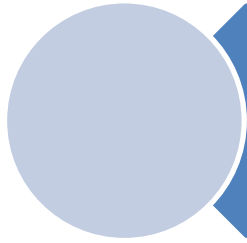
Une fois convenablement initialisée une variable finale ne peut pas voir sa valeur évoluée .

Les Opérateurs (arithmétiques binaires)

```
package essai ;  
public class Op_Arith_Binaire  
{   public static void main(String [ ] args)  
    { int a =100, b = 50 ;  
      int addit  = a + b ;  
      int soustr = a - b ;  
      int div    = a / b ;  
      int multi  = a * b ;  
      System.out.println(" addit =" +addit );  
      System.out.println(" soustr =" +soustr );  
      System.out.println(" div =" +div );  
      System.out.println(" multi =" +multi );  
    }  
}
```

+ **-** ***** **/** sont des
opérateurs arithmétiques
binaires qui portent sur **deux**
opérandes de *même type* et
renvoient un *résultat du même*
type que le type des opérandes

addit	=	150
soustr	=	50
div	=	2
multi	=	5000



Les Opérateurs (unaire et modulo)

```
package essai ;
```

```
public class Op_Unaire_et_Modulo
```

```
{ public static void main(String [ ] args)
```

```
{ int  a =100, b = 50 ;
```

```
int  _modulo = 100 % 50 ;
```

Opérateur modulo:
fournit le reste de la
division de deux
opérandes.

```
System.out.println(" le reste de la division de " +a+" par" +b+" est"+_modulo );
```

```
float  x = + 10.2f ;
```

```
double d= -50.2;
```

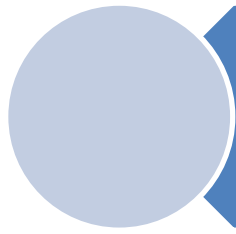
```
System.out.println (" x et d sont des opérateurs unaires portant sur un seul opérande" );
```

```
}
```

```
}
```

le reste de la division de 100 par 50 est 0

x et d sont des opérateurs unaires portant sur un seul opérande



Les Opérateurs (priorité)

Lorsque plusieurs opérateurs apparaissent dans une même expression, il est nécessaire de savoir dans quel ordre ils sont mis en jeu

- Les opérateurs unaires $+$ et $-$ ont la priorité la plus élevée.
- On trouve ensuite au même niveau, les opérateurs $*$, $/$ et $\%$.
- Au dernier niveau, se retrouvent les opérateurs binaires $+$ et $-$.
- En cas de priorité identique, les calculs s'effectuent de gauche à droite. On dit que l'on a une *associativité de gauche à droite*.



Conversions implicites(1/5)

Les opérateurs arithmétiques ne sont définis que lorsque leurs deux opérandes sont de même type. Mais, vous pouvez écrire des *expressions mixtes dans lesquelles interviennent des opérandes de types différents*.

```
public class ConversionLegale01 {  
    public static void main(String args [])  
    {  
        int a = 10 ;  
        float p =14 ;  
        double d= p + a;  
        System.out.println("la valeur de l'expression mixte (p+a) est :"+d);  
    }  
}
```

La valeur de l'expression mixte (p+a) est : 24.0

Conversions implicites(2/5)

```
double d= p + a;
```

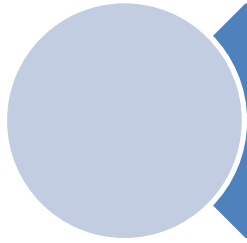
Étape 1: **a** est converti en **float**, **p** reste **float**

Étape 2: on effectue **p+a** qui sera de type **float**

Étape 3: le résultat est converti en **double**

Étape 4: le résultat est stocké (affectation) dans la variable **d**

Ces 4 étapes
sont réalisées
par le
compilateur
d'où
conversions
implicites



Conversions implicites(3/5)

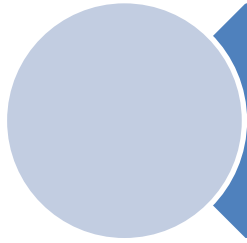
Conversions d'ajustement de type

- Une conversion telle que *int* en *float* est appelée ajustement de type .
- Elle ne peut se faire que suivant une hiérarchie qui permet de ne pas dénaturer la valeur initiale, à savoir :

int -> long -> float -> double

- NB : une conversion de *double* en *float* (par exemple) n'est pas légale . pour l'exemple précédent on ne peut pas faire :

int k = p + a ; // erreur de compilation



Conversions implicites(4/5)

Promotions numériques

- les opérateurs numériques ne sont pas définis pour les types *byte*, *char* et *short*.
- Toute opération qui utilise l'un de ces types nécessite une conversion préalable dans le type *int*
- Cette conversion porte le nom de promotion numérique .

Conversions implicites(5/5)

```
public class ConversionLegale02 {  
    public static void main(String args [])  
    {  
        char c = 'd' ; // le code du caractère 'd' est converti en int  
        short s = 0 ; // la variable s est convertie également en int  
        int n = c + s ; // le résultat de type int est affecté à n  
        System.out.println("la valeur de l'expression mixte (c+s) est:" + n);  
    }  
}
```

La valeur de l'expression mixte (c+s) est : 100

Les opérateurs relationnels(1/2)

Opérateur	Signification
<	inférieur à
<=	inférieur ou égal à
>	supérieur à
>=	supérieur ou égal à
= =	égal à
!=	différent de

Les quatre premiers (<, <=, >, >=) sont de même priorité.
Les deux derniers(= = et !=) possèdent également la même priorité, mais celle-ci est inférieure à celle des précédents

Ces opérateurs sont moins prioritaires que les opérateurs arithmétiques. Ils soumettent eux aussi leurs opérands aux promotions numériques et ajustement de type .

Les opérateurs relationnels(2/2)

Exemple:

```
public class Oper_Relat {  
    public static void main(String args [])  
    {  
        int n = 10 ;  
        short s =10 ;  
        float x = 100;  
        double d= 200;  
        System.out.println("Affichage 1 :"+(n == s) );  
        System.out.println("Affichage 2 :"+(d <= x) );  
    }  
}
```

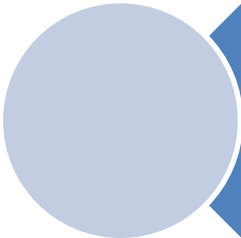
Affichage 1 : true
Affichage 2 : false

Les opérateurs logiques(1/3)

Java dispose d'opérateurs logiques classées par ordre de priorités décroissantes (il n'existe pas deux opérateurs de même priorité).

Le résultat est toujours un booléen.

Opérateur	Signification
!	négation
&	et
^	ou exclusif
	ou inclusif
&&	et (avec court-circuit)
	ou inclusif (avec court-circuit)



Les opérateurs logiques (2/3)

■ $(a < b) \ \&\& \ (c < d)$

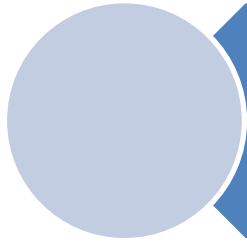
- prend la valeur true (vrai) si les deux expressions $a < b$ et $c < d$ sont toutes les deux vraies (true), la valeur false (faux) dans le cas contraire.

■ $(a < b) \ || \ (c < d)$

- prend la valeur true si l'une au moins des deux conditions $a < b$ et $c < d$ est vraie, la valeur false dans le cas contraire.

■ $!(a < b)$

- prend la valeur true si la condition $a < b$ est fausse, la valeur false dans le cas contraire. Cette expression possède la même valeur que $a \geq b$.



Les opérateurs logiques (3/3)

Les opérateurs de court-circuit **&&** et **||** .

- Ces deux opérateurs recèlent une propriété très importante: leur second opérande (figurant à droite de l'opérateur) n'est évalué que si la connaissance de sa valeur est indispensable pour décider si l'expression correspondante est vraie ou fausse.

Exemple :

```
( 15 < 10 ) && ( 10 > 4)  //on évalue 15 < 10 , le résultat
                           // est faux donc on n' évalue pas
                           // 10 > 4
```


Opérateurs d'incrémentation et de décrémentation

incrémentation

```
int i = 10 ;  
i++ ; // cette expression vaut 10  
      //mais i vaut 11  
  
int j = 10 ;  
++j ; // cette expression vaut 11  
      //et j vaut aussi 11
```

post incrémentation

pré incrémentation

En fait en écrivant :

```
int n = i++ ;
```

on a :

```
n = i ;
```

```
i = i+1 ;
```

Et en écrivant :

```
int p = ++j ;
```

on a :

```
j = j+1 ;
```

```
p = j ;
```

Il existe un opérateur de décrémentation notée --



Opérateurs d'affectation élargie

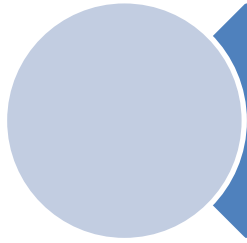
```
int i= 20 ;  
i = i + 1 ; // i vaut 21  
i += 1 ;   // i vaut aussi 21
```

Leur rôle =
condenser les
expressions.
Pas de conversion.

variable = variable opérateur expression ⇔ **variable opérateur = expression**

Liste complète des opérateurs d' affectation élargie :

+= -= *= /= |= ^= &= <<= >>= >>>=



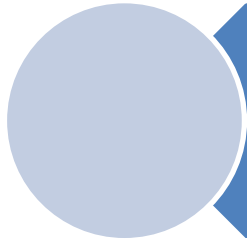
Opérateur Conditionnel

? :

```
if(ciel == bleu)
    temps ="beau"
else
    temps=" mauvais"
```

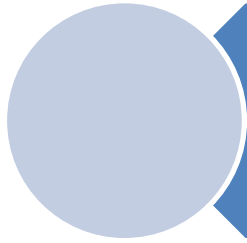


```
temps =ciel==bleu ?"beau" : " mauvais"
```



Priorité des opérateurs (1/2) (ordre de priorité décroissante)

les parenthèses	()
les opérateurs d'incrémentation	++ --
les opérateurs de multiplication, division, et modulo	* / %
les opérateurs d'addition et soustraction	+ -
les opérateurs de décalage	<< >>
les opérateurs de comparaison	< > <= >=



Priorité des opérateurs (2/2) (ordre de priorité décroissante)

les opérateurs d'égalité	== !=
l'opérateur OU exclusif	^
l'opérateur ET	&
l'opérateur OU	
l'opérateur ET logique	&&
l'opérateur OU logique	
les opérateurs d'assignement	= += -=

