Programmation orientée objet : Java

OPÉRATEURS D'INCRÉMENTATION ET DE DÉCRÉMENTATION

- ++ i : expression qui incrémente de 1 la valeur de i, et sa valeur est celle de i après incrémentation

- si la valeur de i est 5, l'expression : n = ++i - 5 affectera à i la valeur 6 et à n la valeur 1.
- n = i++ - 5 (n==0 , i++ vaut 5, i vaut 6)

- On dit que ++ est:

  -un opérateur de **pré incrémentation** lorsqu'il est placé à gauche

  -un opérateur de **post incrémentation** lorsqu'il est placé à droite

OPÉRATEURS D'INCRÉMENTATION ET DE DÉCRÉMENTATION

- Par exemple, le code suivant attribue la valeur 2 autant à i qu'à j :
  - i = 1;
  - j = ++i;
- Mais les lignes suivantes attribuent la valeur 2 à i et la valeur 1 à j:
  - i = 1;
  - j = i++;

OPÉRATEURS D'INCRÉMENTATION ET DE DÉCRÉMENTATION

- De manière similaire, l'opérateur - - décrémente d'une unité son seul opérande numérique

- On dit que -- est:

  -un opérateur de **pré décrémentation** lorsqu'il est placé à gauche

  -un opérateur de **post décrémentation** lorsqu'il est placé à droite

CONVERSIONS DE TYPE

- Java autorise la conversion entre des valeurs entières et des valeurs à virgule flottante.
- Les types char peuvent être convertis en types entiers et à virgule flottante et vice versa.
- Boolean représente le seul type primitif qui ne peut pas être converti en un autre type primitif Java, et vice versa.

CONVERSIONS DE TYPE

- Il existe deux types de conversion de base:
  - Une conversion élargissante se produit lorsqu'une valeur d'un type est convertie vers un type plus large.
  - Une conversion restrictive se produit lorsqu'une valeur est convertie vers un type qui est représenté avec moins de bits.
- Java exécute automatiquement les conversions élargissantes:
  - short a =5 ;
  - int i = a;

CONVERSIONS DE TYPE

- Cependant, les conversions restrictives ne sont pas toujours sans danger.
  - int i = 13;
  - byte b = i; // le compilateur n'autorise pas ceci

- Vous pouvez effectuer un transtypage (cast) en plaçant le nom du type désiré entre parenthèses devant la valeur à convertir.

CONVERSIONS DE TYPE

- Exemple:
  - Int i ; byte b = (byte) i; //oblige la valaur int à etre convertie en une valeur byte
  - i = (int) 13.456; // transforme cette valeur littérale de type double en valeur 13 entière

CONVERSIONS DE TYPE

- Le type char agit comme un type entier dans la plupart des situations
- Exemple:
  - char c = 'f';
  - int i2 = c; // la conversion d'une valeur char en une valeur int

CONVERSIONS DE TYPE

- Le tableau 1 indique quels types primitifs peuvent être convertis vers d'autres types et comment la conversion a lieu.
  - N: Dans le tableau signifie que la conversion ne peut pas avoir lieu.
  - Y: Signifie que la conversion correspond à une conversion élargissante, exécutée par conséquent automatiquement et implicitement par Java.
  - C : Signifie que la conversion est restrictive et nécessite un transtypage explicite.
  - Y*: Signifie qu'il s'agit d'une conversion élargissante automatique mais qu'au moins quelques chiffres significatifs de la valeur pourraient être perdus dans la conversion.

TABLEAU 1 : CONVERSIONS DE TYPES PRIMITIFS

Conversion	Conversions vers							
de	boolean	byte	short	char	int	long	float	double
boolean	-	N	N	N	N	N	N	N
byte	N	-	Y	C	Y	Y	Y	Y
short	N	C	-	C	Y	Y	Y	Y
char	N	C	C	-	Y	Y	Y	Y
int	N	C	C	C	-	Y	Y*	Y
long	N	C	C	C	C	-	Y*	Y*
float	N	C	C	C	C	C	-	Y
double	N	C	C	C	C	C	C	-

L'OPÉRATEUR CONDITIONNEL

syntaxe *: condition* **?** *Valeur si vrai* **:** *valeur si faux*

z = (x==y) ? a : b ; on utilise la valeur de l'expression

a>b ? i++ : i-- ; la valeur de l'expression n'est pas utilisée

- x>=o ? x : -x; correspond à la valeur absolue
- m = ((a>b) ? a : b); affecte à m le maximum de a et b

EXERCICES

INSTRUCTIONS ET STRUCTURES DE CONTRÔLE

INSTRUCTIONS DE BRANCHEMENT CONDITIONNEL

- **if ----- else**

  if (expression1)

  Instruction1

  else if (expression2)

  Instruction2

  …

- **Le else est facultatif**

  If (expression)

  instruction

INSTRUCTIONS DE BRANCHEMENT CONDITIONNEL

- **Switch**

```
switch (expression)
{ case constante1 :
                liste d'instructions1;
                break;
  case constante2 :
                liste d'instructions2;
                break;
      ...
  default :
                liste d'instructionsn;
                                        }
```

LES BOUCLES

- **while :**

while (expression)

instruction;

Tant que expression est vraie, instruction est exécutée.

Si expression est faux instruction ne sera jamais exécutée

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

LES BOUCLES

- **do ---- while :**

```
do
instruction;
while (expression);
```

Ici l'instruction est exécutée tant que expression est vraie.
Instruction est toujours exécutée au moins une fois.

```
int i=0;
do
{
        System.out.println(" i = "+i);
        i++;
}
while ((i <=0) || (i > 10));
```

LA BOUCLE FOR:

- for :

```
for ( expr1; expr2; expr3)
instruction;
```

équivalent à :

```
expr1;
while (expr2);
{
        instruction;
        expr3;
}
```

COMMENTAIRES : FOR

- Chacune des trois expressions est facultative. Ainsi ces constructions sont équivalentes à l'instruction for de notre premier exemple de programme :

```
for (i=1 ; i<=5 ; i++ )
 {
System.out.println(" i = "+i);
}
```

```
i = 1 ;
for ( ; i<=5 ; )
{
System.out.println(" i = "+i);
i++ ;
}
```

COMMENTAIRES : FOR

- Lorsque l'expr2 est absente, elle est considérée comme vraie.
- On peut grouper plusieurs actions dans une expression. Ainsi :

```
for ( i=0, j=1, k=5 ;  ;  )
```

est équivalent à :

```
j=1 ; k=5 ;
for ( i=0 ;  ;  )
```

ou encore à :

```
i=0 ; j=1 ; k=5 ;
for ( ;  ; )
```

INSTRUCTIONS DE BRANCHEMENT NON CONDITIONNEL

- ***break*** : vu en switch, en général permet d'interrompre le déroulement d'une boucle, et passe à la première instruction qui suit la boucle.

```
 int i ;
for (i = 0; i < 5; i++)
{
    System.out.println(" i = “+i);
    if (i==3)
        break;
}
    System.out.println(" i = “+i);
```

INSTRUCTIONS DE BRANCHEMENT NON CONDITIONNEL

- ***continue*** : permet de passer directement de la boucle suivante sans exécuter les autres instructions de la boucle.

```
 int i;
for (i = 0; i < 5; i++)
{
    if (i==3)
      continue;
    System.out.println(" i = "+i);
}
System.out.println(" i = "+i);
```