Chapitre4:

Les opérateurs et les instructions de contrôle

Opérateurs d'affectation composés

Les plus utilisés sont: +=, - =, *=, and /=

```
y -= 6;          ⇔    y = y- 6;
x += 2 * 5;           x = x+ 2 * 5;
```

Exemple: Laquelle des expressions suivantes est equivalente à l'expression:

x*= 2 + 5;

```
x = (x * 2) + 5;   // incorrect
x = x * (2 + 5) ;  // correct
```

L'expression à droite est toujours placées entre parenthèses

Les opérateurs "==" et "!="

"==" et "!=" pour les primitives

Si les primitives sont égaux, le résultat sera true avec == et false avec !=

Example:

```
'a' == 'a'; // true
5 != 6;  //true
5.0 == 5L; //true
```


Example: Quel est le résultat de ce code?

```
boolean b = false;
 if (b = true) { System.out.println("b is true");}
else { System.out,println("b is false");  }
```

b is true

Le code suivant ne compile pas:

```
int x = 1;
if (x = 0) { }
```

"==" and "!=" pour les variables de référence

Exemple: Quel est le résultat du code suivant?

```
import javax.swing.JButton;
class CompareReference {
 public static void main(String[] args) {
    JButton a = new JButton("Exit");
   JButton b = new JButton("Exit");
    JButton c = a;
    System.out.println("Is reference a == b? " + (a == b));
System.out.println("Is reference a == c? " + (a == c));  }}
```

Résultat:
Is reference a == b? false
Is reference a == c? true

Comparison avec instanceof

Exemple 1: Quel est le résultat du code suivant?

```
public static void main(String[] args) {
  String s = new String("foo");
 if (s instanceof String) {
    System.out.print("s is a String");  }
}
```

Résultat: s is a string

L'opérateur instanceof est utilisé uniquement pour les variables de références d'objets, et il est utilisé pour tester si un objet est d'un type particulier. Type désigne classe ou interface.

Exemple 2: soit:

```
interface Foo { }
class A implements Foo { }
class B extends A { }
...
A a = new A();
B b = new B();
```

Les instructions Ci-dessous sont vraies:

a instanceof Foo
b instanceof A
b instanceof Foo

Exemple 3: Est ce que ce code compile?

```
class Cat { }
class Dog {
 public static void main(String [] args) {
    Dog d = new Dog();
    System.out.println(d instanceof Cat);  }
}
```

Ne compile pas

Il n'est pas possible que d réfère un Cat ou un sous type de Cat.

L' opérateur de Concaténation de String

Exemple:

```
String a = "String";
int b = 3;
int c = 7;
System.out.println(a + b + c);
System.out.println(a + (b + c));
```

Résultat: String37
String10

Il est possible d'utiliser += avec Strings, par exemple:

```
String s = "123";
s += "45";
s += 67;
System.out.println(s);
```

Résultat: 1234567

Exercice Quel est le résultat du code suivant?

```
public class Greek {
int i=1;
public int getI(){
System.out.print("ici getI..");
return i=1;
}
public static void main(String[] args) {
Greek gr= new Greek();
System.out.println(gr.i+3+" "+gr.i+ " "+ gr.getI()+" ");
}
}
```

ici getI..4 1 1

Les opérateurs d'incrémentation et de décrémentation ++ incrémenter (préfix et postfix)
-- décrémenter (préfix et postfix)

Quel est le résultat de chaque opération (b est toujours initialisé à 2)?

1. a=b++;
2. a=++b;
3. a=b--;
4. a=--b;
5. a=1;a+=b++;
6. a=1;a+=++b;
7. a=1;a-=b++;
8. a=1;a-=++b;
9. a=1;a+=b--;
10. a=1;a+=--b;

1. a=2 ; b=3
2. a=3 ; b=3
3. a=2; b=1
4. a=1;b=1
5. a=3;b=3
6. a=4;b=3
7. a=-1 ; b=3
8. a=-2;b=3
9. a=3;b=1
10. a=2 ; b=1

Exercice: Quel est le résultat du code suivant?

```
class MathTest {
  static int players = 0;
  public static void main (String [] args) {
  System.out.println("players : " + players++);
   System.out.println("La valeur de players est "  +
players);
  System.out.println(" La valeur de players est
maintenant "  + ++players);
} }
```

players: 0
La valeur de players est 1
La valeur de players est maintenant 2

Exemple: Le code suivant ne compile pas:

```
final int x = 5;
int y = x++; // la valeur de la variable final x ne peut pas
             changer
```

Les opérateurs logiques court-circuit

Et logique: &&; ou logique: ||

Exemple: quel est le résultat du code suivant?

```
boolean b = false && true;
   System.out.println("boolean b = " + b);
```

boolean b = false

Les opérateurs logiques non court-Circuit

Et logique: &; ou logique: |

Exemple:

```
int z = 5;
if(++z > 5 || ++z > 6) z++; System.out.println("z="+z);
                                                    // z = 7
int z = 5 ;
if (++z > 5 | ++z > 6) z++; System.out.println("z="+z);
                                                    // z = 8
```

Les opérateurs logiques ^ (xor) et ! (not)

Exemple:

```
System.out.println ("xor " + ((2<3) ^ (4>3)));
```

Produit le résultat: xor false

```
boolean t = true;
boolean f = false;
System.out.println ("! " + (t & !f) + " " + f);
```

Produit le résultat: ! true false

Les instructions de contrôle

Expression Conditionnelle

```
if (expression booléenne) {
instructions; // exécutées si expression  boolénne
//retourne true
}
else {
instructions2; // exécutées si expression booléenne
//retourne false
}
```

Remarque: Un bloc serait préférable, même s'il n'y a qu'une seule instruction.

Autre forme

x = (expression booléenne) ? expression1 si true
: expression2 si false

Est équivalent à

```
if (expression booléenne) {
x= expression1;
}
else {
x= expression2;
}
```


Exemple: quel est le résultat du code suivant?

```
int x=6;
int y = (x % 2 == 0) ? x + 1 : x;
```

y=7

Les instruction Switch

```
switch(expression) {
case val1: instructions; // exécutées si expression ==val1
break; // Attention, sans break, les instructions du cas
        // suivant sont exécutées !
. . .
case valn: instructions; // exécutées si expression ==valn
break;
default: instructions; // exécutées si aucune des valeurs
                       //prévues
break;
}
```

- *Expression* est de type **char**, **byte**, **short**, ou **int**, ou de type énumération (ou type **énuméré défini avec** enum).

- À partir de java 7 expression peut être de type String

- S'il n'y a pas de clause **default**, rien n'est exécuté si *expression* ne correspond à aucun **case (**aucune valeur prévue**).**

Exemple: Quel est le résultat du code suivant?

```
int x = 3; // x peut être initialisé à un nombre entre 1 et 10
switch (x) {
  case 2:
  case 4:
  case 6:
 case 8:
  case 10:
    System.out.println("x est un nombre pair");
 break;
 default: System.out.println("x est un nombre impair");

}
```

X est un nombre impair

Exemple: Quel est le résultat du code suivant?

```
int x = 2;
switch (x) {
 case 2:  System.out.println("2");
 default: System.out.println("default");
 case 3: System.out.println("3");
 case 4: System.out.println("4");
}
```

2
default
3
4

Les boucles de répétition

Deux types de boucles :

- Répétitions « tant que »

while (*expressionBooléenne***) {**

instructions; // corps de de la boucle

}

- Répétition « faire tant que »: le corps de la boucle est *exécuté au moins une fois.*

do {

instructions; // corps de de la boucle

} while (*expressionBooléenne***);**

Exemple:

```
int x = 2;
while(x == 2) {
System.out.println(x);
 ++x;
}
```

Les variables utilisées dans l'expression de boucle while doivent être declarées avant l'évaluation de l'expression.

Exemple:

```
while (int x == 2) { }   // ne compile  pas
```


Utiliser la boucle for

```
for (/*Initialization*/ ; /*Condition*/ ; /* Iteration */)
{  /* loop body */}
```

– **initialisation (**initialisation**):** Déclaration et/ou affectations, séparées par des virgules

– **Itération :** expressions séparées par des virgules

– **Condition:** expression booléenne

Exemple:

```
for (int i = 0; i<10; i++) {
 System.out.println("i is " + i);
}
```


Remarque: Dans le cas où le corps de la boucle est formée d'une seule instruction, il n'y a pas besoin de mettre les accolades { } pour marquer le début et la fin du bloc

- Il existe une autre forme de la boucle for (utilisée avec les tableaux) à partir de java 5

Exemple: soit le code suivant:

```
int [] a = {1,2,3,4};
for(int x = 0; x < a.length; x++) // boucle for basique
System.out.print(a[x]);
```



```
for(int n : a)          // autre forme de la boucle for
 System.out.print(n);
```

Ce qui produit le résultat suivant:

12341234

➢ L'itérateur doit être déclaré dans la boucle et doit avoir le même type d'éléments du tableau

Interruption de boucles

Les boucles peuvent être interrompues en utilisant **break**, **return**, **System.exit()**, ou **une exception**, qui peuvent causer une boucle à se terminer.

Exemple:

```
static int doStuff() {
 for (int x = 0; x < 3; x++) {
  System.out.println("in for loop");
    return x;  }
return 0;}
```

Code dans la boucle	Ce qui se passe
break	Exécution saute immédiatement à la première instruction après la boucle.
return	Exécution retourne immédiatement à la méthode appelante
System.exit()	Toutes les exécutions des programmes stoppent. La machine virtuelle se ferme.

Utiliser Break et Continue

L'instruction ***continue*** doit être à l'intérieur de boucle; sinon, il y'aura une erreur de compilation

L'instruction ***break*** doit être utilisée dans soit une boucle soit dans switch

Exemple:

```
boolean problem = true;
while (true) {
if (problem)
 {   System.out.println("il y'avait un problème");
 break;
  }}
 // next line of code
```


- L'instruction *continue* permet de passer à l'itération suivante dans la même boucle,

Exemple: Quel est le résultat du code suivant?

```
for (int i = 0; i < 7; i++) {
 if (i == 5) {    continue;  }
System.out.println("Inside loop"+i);
}
```

```
Inside loop0
Inside loop1
Inside loop2
Inside loop3
Inside loop4
Inside loop6
```


Etiquettes de boucles

Exemple:

```
boolean isTrue = true;
outer:  for(int i=0; i<5; i++) {
  while (isTrue) {
  System.out.println("Hello");
    break outer;    } // fin de la boucle interne while
 System.out.println("Outer loop.");
  }
System.out.println("Good-Bye");
```

Output:
Hello
Good-Bye

=> L'étiquette doit obeir aux règles d'appellation de java.