Java
Les instructions

Préparé par Larbi Hassouni

Instructions

Les instructions sont l'équivalents des phrases dans un langage naturel

Une instruction est une unité complète d'exécution.

Les expressions ci-dessous peuvent se transformer en instructions en les terminant par un point virgule (;).

- Expressions d'affectation
- N'importe quelle utilisation de ++ et --
- Appels de méthodes
- Expressions de création d'objets

De telles expressions sont appelées des instruction-expressions

Exemples d'instruction-expressions.

//instruction d'affectation

uneValeur = 14.5 ;

//instruction d'incrémentation

uneValeur++ ;

//Appel d'une méthode

System.out.println(uneValeur) ;

//Création d'un objet

Integer integerObject = new Integer(4);

En plus des instruction-expressions, il ya deux autres types d'instructions.

-Instruction de déclaration de variables.

Exemple :

```
double uneValeur = 14.5;
```

- Instruction de contrôle qui détermine l'ordre d'exécution des instructions.

Les instructions if et for sont des exemples d'instructions de contrôle.

Exemple :

```
if (a>b) {
        Max = a;
    }
```

Bloc

Un bloc est un groupe d'instructions placées entre crochet {} et peut être utilisé

n'importe où une instruction simple est autorisée.

Le code qui suit présente deux blocs qui contient chacun une seule instruction

```
if (Character.isUpperCase(aChar)) {
        System.out.println("The character " + aChar  + " is uppercase.");
}
else {
        System.out.println("The character " + aChar  + " is lowercase.");
}
```

Instructions de contrôle d'exécution

Lorsque vous écrivez un programme, vous écrivez une suite d'instructions que vous stockez dans un fichier.

Sans les instructions de contrôle d'exécution, l'intérpréteur exécutera vos instructions dans l'ordre ou elles apparaissent dans le fichier de gauche à droite et du haut en bas.

Instructions de contrôle d'exécution

Les instructions de contrôle permettent de changer l'ordre séquentiel d'exécution des instructions du programme en permettant de:

1. Exécuter des instructions en fonction de l'évaluation d'une condition;
2. Répéter l'exécution d'un même groupe d'instructions.
   Le nombre de répétition dépend du résultat de l'évaluation d'une condition.

Par exemple, dans la partie du code ci-dessous, l'instruction if exécute l'instruction

System.out.println uniquement dans le cas ou la valeur retournée par Character.isUpperCase(aChar) est égale à true.

```
char aChar;
 ...
if (Character.isUpperCase(aChar)) {
      System.out.println("The character " + aChar + " is uppercase.");

}
```

Instructions de contrôle d'exécution

Le langage Java fournit plusieurs instructions de contrôle d'exécution. Elles sont fournies dans le tableau ci-dessous.

Type d'instruction	Mot clé
boucle	while, do-while, for
sélection	if-else, switch-case
Gestion des exceptions	try-catch-finally, throw
saut	break, continue, label:, return

Format général d'une instruction de contrôle

Le format général d'une instruction de contrôle d'exécution est :

```
Instruction de contrôle {
    Instruction(s)
}
```

Techniquement, les accolades — { and } — ne sont pas obligatoires si le bloc contient une seule instruction.

Cependant, il est recommandé de toujours utiliser les accolades parce qu'elles rendent le code plus facile à lire, et aident à éviter de faire des erreurs lorsqu'on modifie le programme.

Instruction de sélection « if …else »

if (*expressionBooléenne*)

bloc-instructions ou instruction

else

bloc-instructions ou instruction

```
if (x % 2 == 0) {
   type = 0;
   x++;
}
else
   type = 1;
```

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

Expression conditionnelle (Opérateur ternaire)

expressionBooléenne **?** *expression1* **:** *expression2*

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

est équivalent à

```
int y;
if (x % 2 == 0)
   y = x + 1
else
   y = x;
```

Parenthèses pas indispensables

Pour plus de détail, voir votre cours du C

Instruction de sélecion <switch ….case…>

```
switch(expression) {
   case val1: instructions;
              break;
   ...
   case valn: instructions;
              break;
default: instructions;
}
```

Attention, sans **break**, les instructions du cas suivant sont exécutées !

S'il n'y a pas de clause **default**, rien n'est exécuté si *expression* ne correspond à aucun **case**

Val1, val2, …valn sont des constantes

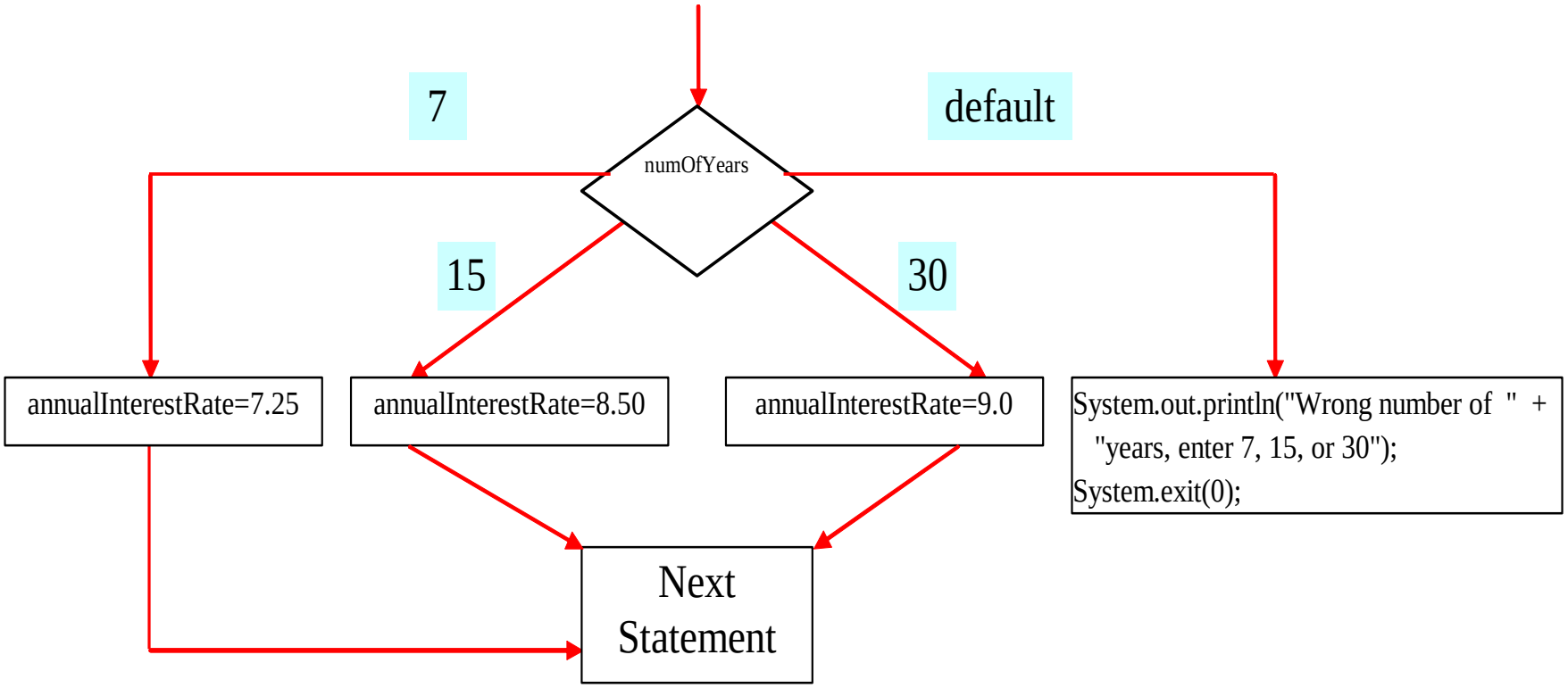
***expression* est de type char, byte, short, int, String ou de type énumération.**

Attention: le type long n'est pas autorisé, seuls les types convertibles en int sont autorisés.

Exemple de switch

```
switch (year) {
  case 7:  annualInterestRate = 7.25;
           break;
  case 15: annualInterestRate = 8.50;
           break;
  case 30: annualInterestRate = 9.0;
           break;
  default: System.out.println(
   "Wrong number of years, enter 7, 15, or 30");
}
```

Organigramme de switch



Exemple de switch

```
char lettre;
int nbVoyelles = 0, nbA = 0,
nbT = 0, nbAutre = 0;
Lettre = 'b';
switch (lettre) {
   case 'a' : nbA++;
   case 'e' :                          // pas d'instruction !
   case 'i' : nbVoyelles++;
                 break;
    case 't' : nbT++;
                  break;
default : nbAutre++;
}
```

Pour plus de détail, voir votre cours du C

Répétitions « tant que »

Il existe deux formes de répétions <<tant que>>:

1-

while(*expressionBooléenne*)
bloc-instructions ou instruction

Si expressionBooleenne est false dés le départ, le bloc d'instruction ne sera exécutée aucune fois

2-

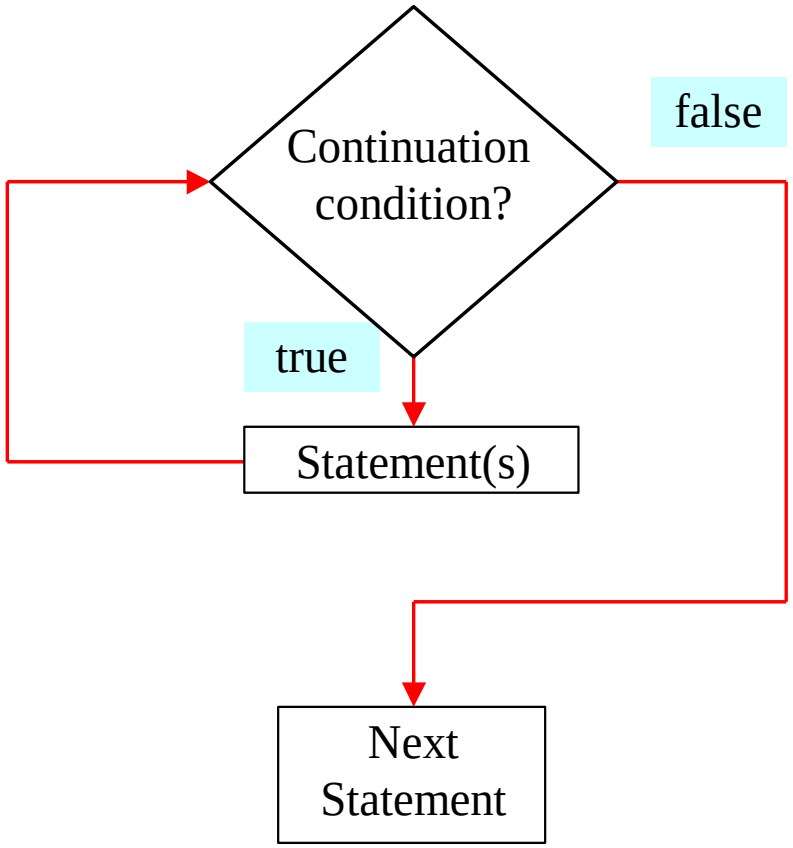
do
bloc-instructions ou instruction
while(expressionBooléenne)

Le bloc d'instructions est exécuté au moins une fois

On répéte l'exécution de *<<bloc-instructions ou instruction>>*
tant que <<expressionBooléenne>> est true.
On sort de la boucle lorsque <<expressionBooléenne>> devient false

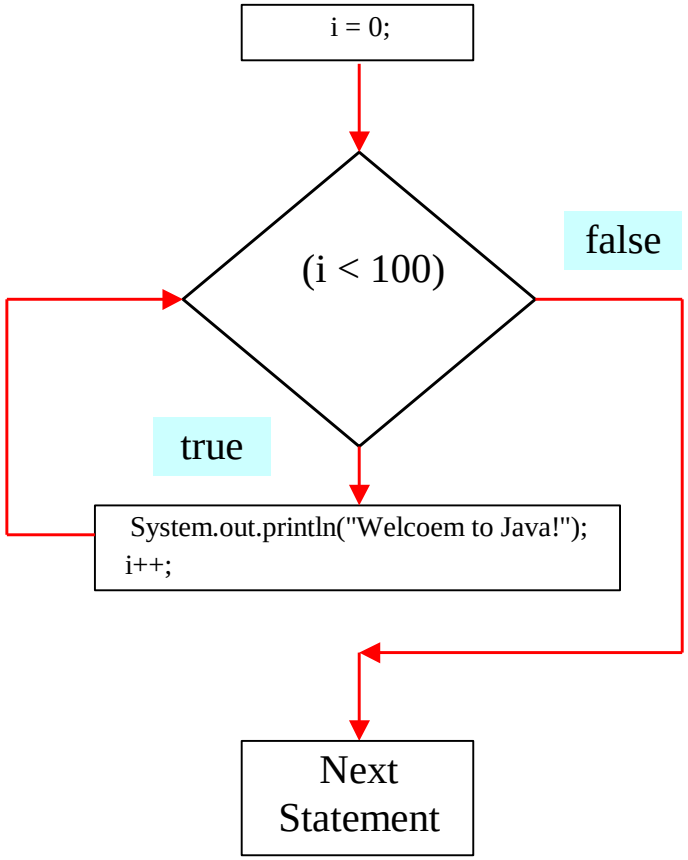
Organigramme de la boucle while

```
while (continuation-condition) {
   loop-body;
}
Next statement
```



Exemple de boucle while

```
int i = 0;
while (i < 100) {
  System.out.println(
    "Welcome to Java!");
  i++;
}
```



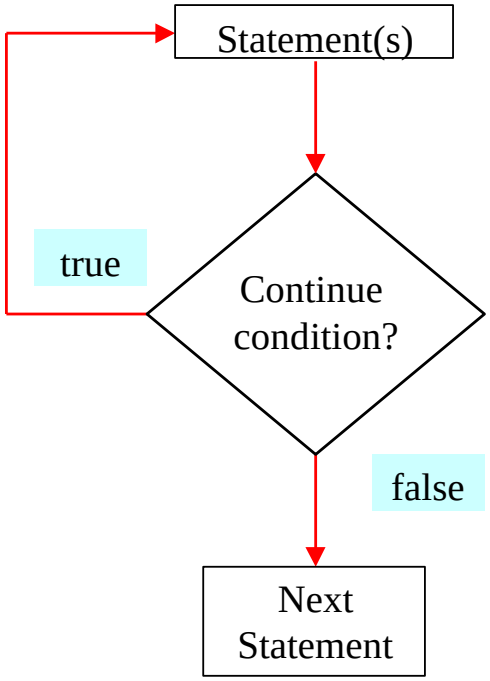
Exemple : diviseur d'un nombre

```
public class Diviseur {
   public static void main(String[ ] args) {
      int i = Integer.parseInt(args[0]);
      int j = 2;
       while (i % j != 0) {
            j++;
       }
      System.out.println("PPD de " + i + " : " + j);
   }
}
```

Pour plus de détail, voir votre cours du C

Organigramme de la boucle do-while

```
do {
    <Statements>
} while (continue-condition);
<Next Statement>;
```



Répétition **for**

```
For (init ;  test ;  incrément ) {
   instructions;
}
```

est équivalent à

```
init;
while (test) {
    instructions;
   incrément;
}
```

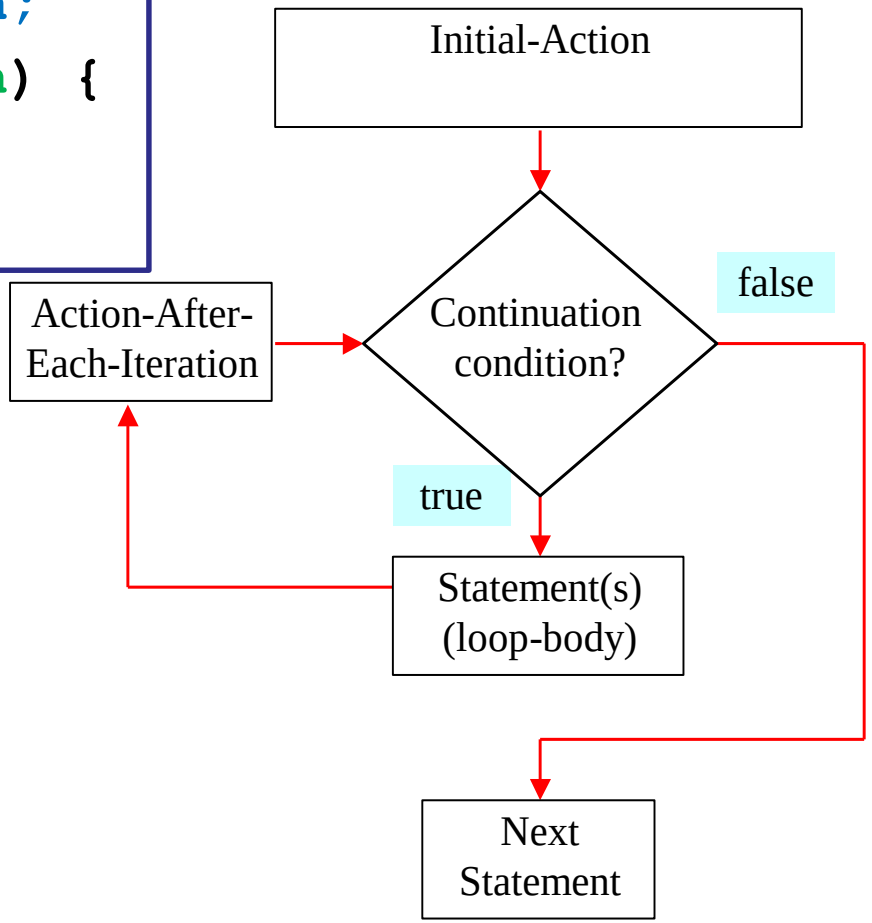
Exemple de **for**

```
int somme = 0;
for (int i=0; i < 100; i++) {
   somme += i;
}
System.out.println(somme);
```

Pour plus de détail, voir votre cours du C

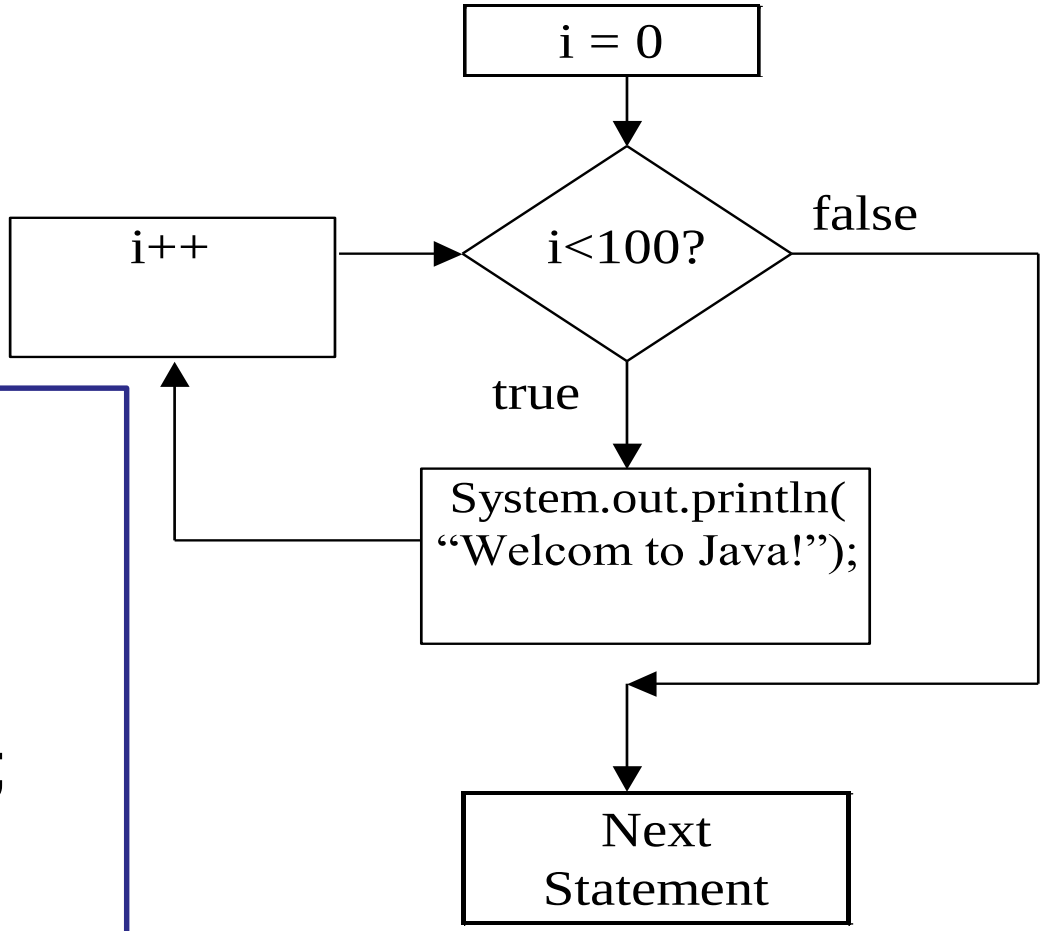
Organigramme de la boucle for

```
for (initial-action;
   loop-continuation-condition;
   action-after-each-iteration) {
    //loop body;
}
```



Exemple de la boucle for

```
int i;
for (i = 0; i<100; i++) {
  System.out.println(
     "Welcome to Java");
}
<Next Statement>
```



Boucle « for each »

Une nouvelle syntaxe introduite par la version 5 du JDK simplifie le parcours d'un tableau

La syntaxe est plus simple/lisible qu'une boucle *for* ordinaire

Attention, on ne dispose pas de la position dans le tableau (pas de « variable de boucle »)

On verra par la suite que cette syntaxe est encore plus utile pour le parcours d'une « collection »

Parcours d'un tableau

```
public class ForEach{
        public static void main(String[] args){
                String[ ] noms = {"Larbi", "Adil", "Tarik", "Mohammed"};
                // Lire « pour chaque nom dans noms »
                // « : » se lit « dans »
                for (String nom : noms) {
                        System.out.println(nom);
                }
        }
}
```

Attention!! : Cette boucle ne permet de modifier les éléments du tableau. Au cas où vous devez modifier lés élément du tableau, il faut utiliser la boucle for classique

Instructions liées aux boucles

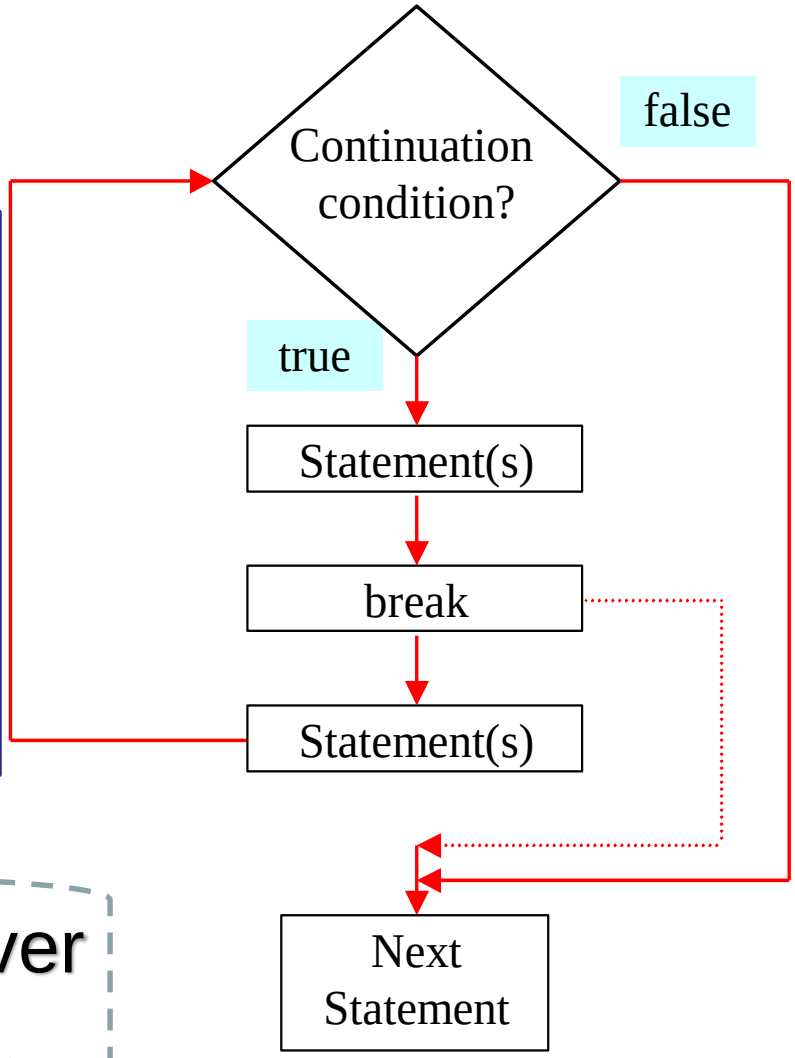
break sort de la boucle et continue après la boucle

continue passe à l'itération suivante

break et **continue** peuvent être suivis d'un nom d'étiquette qui désigne une boucle englobant la boucle où elles se trouvent
(une étiquette ne peut se trouver que devant une boucle)

Instruction break

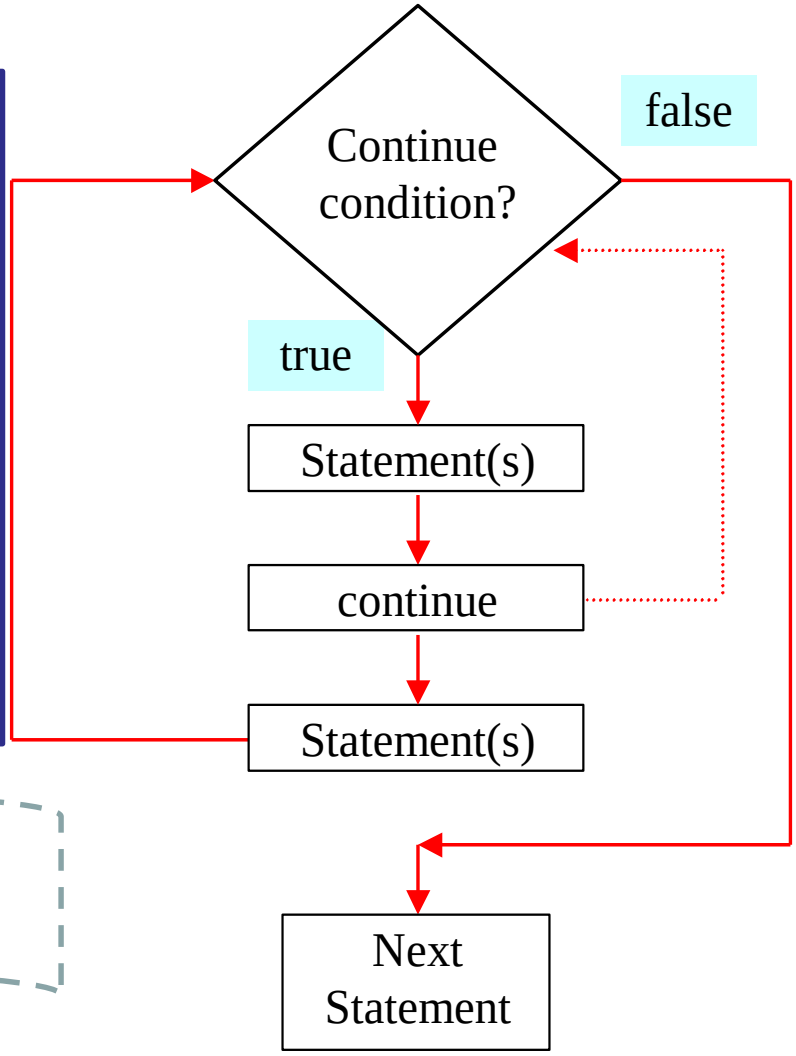
```
While(Continuation-condition){
        Statements
        break;
        Statements
}
```



Break doit toujours se trouver dans un if

Instruction continue

```
While(<Continuation-condition>){
        <Statements>
        continue;
        <Statements>
}
<Next statement>
```



continue doit toujours se trouver dans un if

Exemple de continue et break

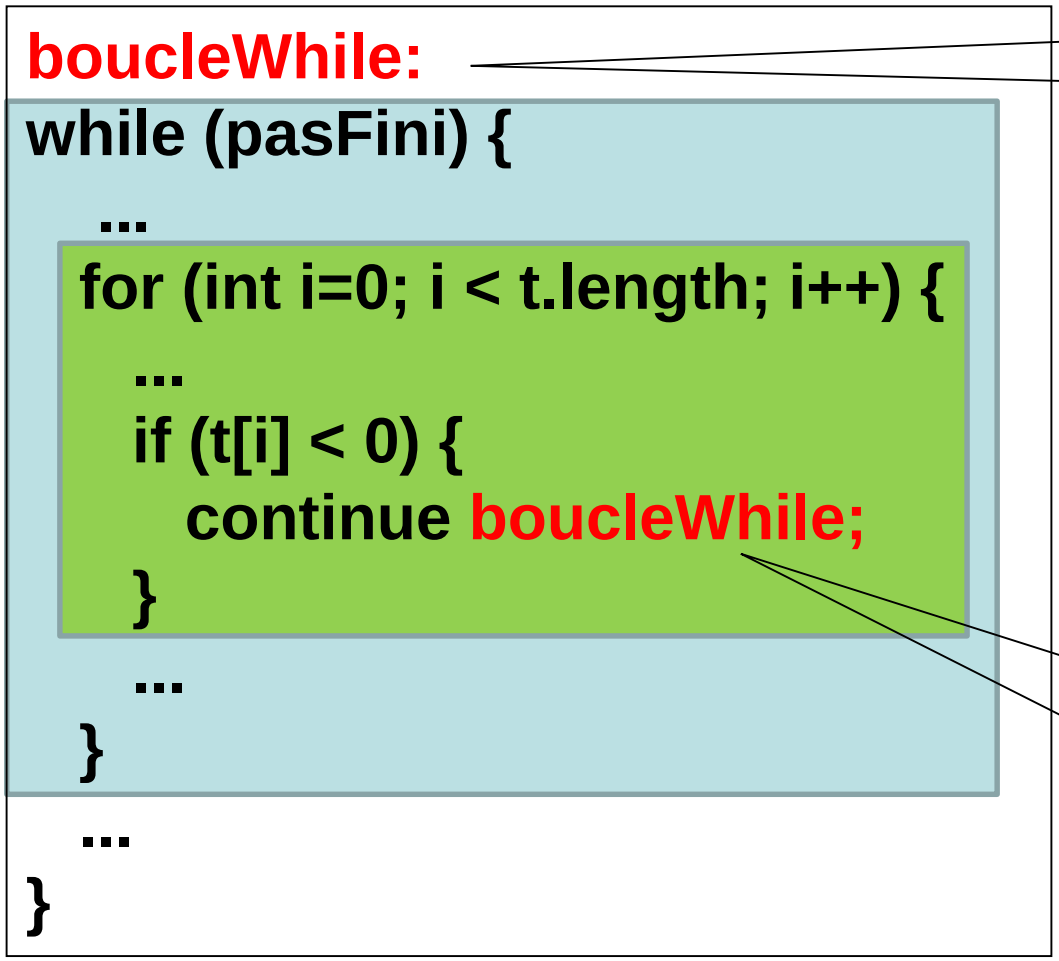
```
int somme = 0;
for (int i=0; i < tab.length; i++) {
   if (tab[i] == 0) {
        break;
   }
   if (tab[i] < 0){
      continue;
   }
   somme += tab[i];
}
System.out.println(somme);
```

Qu'affiche ce code avec le tableau

1
-2
5
-1
0
8
-3
10

Somme = ?

Étiquette de boucles



L'étiquette doit se trouver juste avant l'instruction d'itération

Branchement à l'étiquette boucleWhile : sortie de la boucle for et reprise de nouveau de la boucle while