Chapitre 3: Instructions conditionnelles et Boucles

(NFA031 - Jour)

V. Aponte

Cnam

9 octobre 2017

Blocs d'instructions

Suite d'instructions placées entre accolades `{`, `}`. Ils servent à :

- délimiter les instructions d'une méthode :

```
public static void main(){
    System.out.println ("Hello, world");
}
```

- regrouper des instructions derrière if, else, for, etc. :

```
if (valeur < 0) {
    valeur=-valeur;
    System.out.println("Débit  : "+valeur);
}else
    System.out.println("Crédit : "+valeur);
```

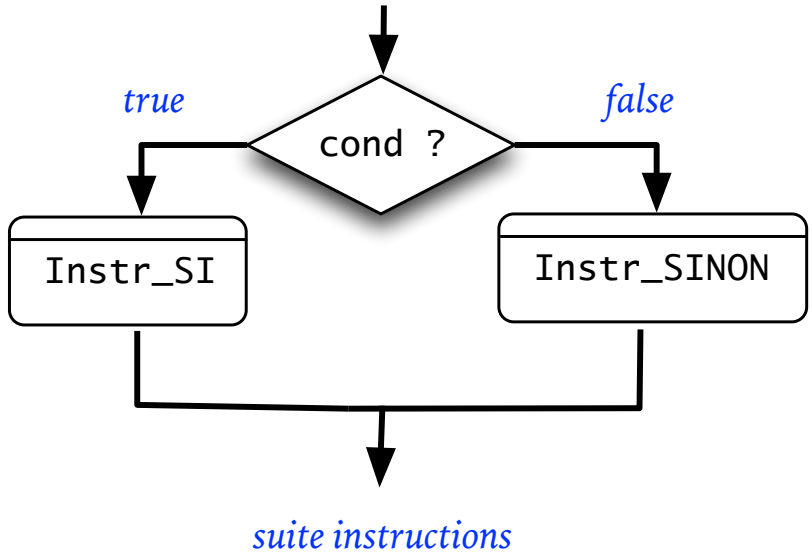
Instruction conditionnelle

Conditionnelle

Problème :

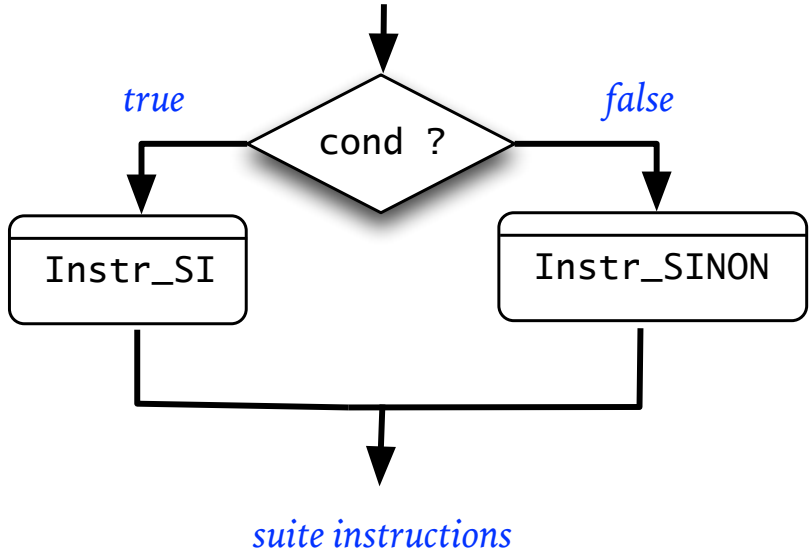
- spécifier différents cas possibles à l'exécution (« conditions »)
- et des comportements (« branches ») pour chaque cas
  - tester une condition
  - + 2 séquences d'instructions différentes (branches) en fonction du résultat du test

```
if (cond){
   Instructions_SI;
} else {
   Instructions_SINON;
}
```



Conditionnelle (bis)

```
if (cond){
   Instructions_SI;
} else {
   Instructions_SINON;
}
```




```
if (cond) {
  ... // instructions si test vrai (branche "si")
}
else {
  ... // instructions si test faux (branche "sinon")
}
```

Exemple de conditionnelle (1)

```
int x = Terminal.lireInt();
if (x >= 0) {
    Terminal.ecrireStringln("positif ou nul.");
}
else {
    Terminal.ecrireStringln("strictement négatif.");
}
Terminal.ecrireString("toujours affiché");
```

1. évaluation de la condition `(x>=0)`
   1. si condition vraie 1ère séquence d'instructions
   2. sinon, 2ème séquence
2. après 1.1 ou 1.2 on exécute de toute façon la suite

Exemple de conditionnelle (condition vraie)

```
int x = Terminal.lireInt();
if (x >= 0) {
    Terminal.ecrireStringln("positif ou nul.");
}
else {
    Terminal.ecrireStringln("strictement négatif.");
}
Terminal.ecrireString("toujours affiché");
```

1. évaluation de la condition `(x>=0)`
   1. si condition vraie 1ère séquence d'instructions
   2. sinon deuxième sinon 2ème séquence
2. après 1.1 ou 1.2 on exécute de toute façon la suite

Exemple de conditionnelle (condition fausse)

```
int x = Terminal.lireInt();
if (x >= 0) {
    Terminal.ecrireStringln("positif ou nul.");
}
else {
    Terminal.ecrireStringln("strictement négatif.");
}
Terminal.ecrireString("toujours affiché");
```

1. évaluation de la condition `(x>=0)`
   1. si condition vraie 1ère séquence d'instructions
   2. sinon, 2ème séquence
2. après 1.1 ou 1.2 on exécute de toute façon la suite

Exemple de conditionnelle (la suite)

```
int x = Terminal.lireInt();
if (x >= 0) {
    Terminal.ecrireStringln("positif ou nul.");
}
else {
    Terminal.ecrireStringln("strictement négatif.");
}
Terminal.ecrireString("toujours affiché");
```

1. évaluation de la condition `(x>=0)`
   1. si condition vraie 1ère séquence d'instructions
   2. sinon, 2ème séquence
2. après 1.1 ou 1.2 on exécute de toute façon la suite

Exemple

Problème : écrire un programme qui, étant donné un prix hors taxe saisi par l'utilisateur, calcule et affiche le prix correspondant TTC. Il y a 2 taux de TVA possibles :

- la TVA normale à 19.6%
- et le taux réduit à 5.5%.

On demandera également de saisir la catégorie du taux qu'il faut appliquer.

Exemple

Données

- entrées : prix HT : `pHT`, `double` ; taux : `t`, `char` ('n' ou 'r').
- sorties : prix TTC : `pTTC`, `double`

Algorithme

1. afficher un message demandant de saisir une somme HT.
2. recueillir la réponse dans `pHT`
3. afficher un message demandant le taux ('n' ou 'r').
4. recueillir la réponse dans `t`
5. 2 cas :
   1. Cas 1 :le taux normal `pTTC= pHT+ (pHT*0.196)`
   2. Cas 2 :le taux réduit `pTTC= pHT+ (pHT*0.05)`
6. afficher `pTTC`

Codage en Java

```
public class PrixTTC {
 public static void main (String[] args) {
    double pHT,pTTC;
    char t;
    Terminal.ecrireString("Entrer le prix HT: ");
    pHT = Terminal.lireDouble();
    Terminal.ecrireString("Taux? (normal->n, reduit ->r) ")
    t = Terminal.lireChar();
    if (t=='n'){
         pTTC=pHT + (pHT*0.196);
    } else {
         pTTC=pHT + (pHT*0.05);
    }
    Terminal.ecrireStringln("La somme TTC: "+ pTTC );
  }
}
```

Mettez toujours les accolades

S'il y a une seule instruction par branche, on peut ommettre les accolades :

Exemple :

```
if (mois < 12)
  mois = mois + 1;
else {
  mois = 1;
  annee = annee + 1;
}
```

Conseil : mettez-les toujours !

`if` sans `else`

Si l'on veut dire :

si condition est vraie alors faire ceci, sinon ne rien faire

on peut omettre le `else`.

Exemple :

```
if (x != 0)
     {y = y + x;}
```

si la condition est fausse, on passe à la prochaine instruction.

Règles de style pour le `if`

- Chaque `else` est aligné sur la même colonne que le `if` auquel il correspond.
- Chaque nouveau `if` est indenté à droite.
- Un `else` correspond toujours au dernier `if` qui le precède.

```
if ( x > 0 )
  if (y > 0)
     System.out.println("Cas 1");
  else
     System.out.println("Cas 2");
```

Quels sont les affichages ?

A qui correspond le `else` ?

Si l'on veut forcer la correspondance du `else` avec le premier if :

```
if ( x > 0 ) {
  if (y > 0) System.out.println("Cas 1");
}
else
  System.out.println("Cas 2");
```

le comportement change :

$\Rightarrow$ affiche "Cas 2" si $x \leq 0$, et n'affiche rien si $y \leq 0$

Tests à la suite : `if ... else if`

```
if (condition1){
    s1
}
else if (condition2) {
    s2
...
}
else if (conditionN) {
    sN
}
else {
    sN+1
}
```

Chaque `else` correspond au `if` qui le précède.

`if ... else if`

- On peut mettre autant de `else if` que l'on veut, avec ou sans un `else` final.
- Les conditions sont testées dans l'ordre. La première qui est vraie entraîne l'exécution du bloc associé.
- Si aucune condition n'est vraie, c'est le bloc du `else` qui est exécuté.
- Cette instruction est une sorte de «conditionnelle» à N+1 branches. Seule 1 des branches est exécutée.

Exemple de `if ... else if`

```
if (temperature < 15)
   System.out.println("Il␣fait␣froid.");
else if (temperature < 27)
   System.out.println("Il␣fait␣bon.");
else
   System.out.println("Il␣fait␣chaud.");
```

Quelles sont les valeurs de `temperature` dans chaque cas ?

- "Il fait froid." $\Rightarrow$ `temperature` ≤ 14
- "Il fait bon." $\Rightarrow 15 \leq$ `temperature` ≤ 26
- "Il fait chaud." $\Rightarrow$ `temperature` ≥ 27

Portée, localité des variables

Variables locales à un bloc

- Les variables déclarées dans un bloc ne sont visibles que dans ce bloc.
- Elles sont locales au bloc.

```
{
  int v = 7;
  System.out.println("Débit␣␣:␣"+v);
}
System.out.println("Crédit␣:␣"+v);
   // Erreur: v inconnue ici
```

- Interdit : déclarer plusieurs fois la même variable dans un bloc.

Bloc : environnement local

- Un bloc est un environnement local de déclaration : toute variable n'est connue qu'à l'intérieur du bloc où elle est déclarée.
- La vie d'une variable déclarée dans un bloc se termine en franchissant l'accolade fermante de ce bloc.

```
for (int i=0; i<5; i=i+1){
    Terminal.ecrireIntln(i);      // Correcte
}
Terminal.ecrireIntln(i);         // Erreur
```

Bloc imbriqués

```
public static void main(String[] arg){/*debut bloc 1*/
  int a=2;
  Terminal.ecrireStringln("valeur de a: "+ a );
  if (a==0){
      /* debut bloc 2 */
      int b=3+a;
      Terminal.ecrireStringln("valeur de b: "+ b );
  }   /* fin bloc 2*/
  else     {
      /* debut bloc 3*/
      int c=3+a;
      Terminal.ecrireStringln("valeur de c: "+ c);
  }    /* fin bloc 3*/
  Terminal.ecrireStringln("valeur de a: "+ a );
 }       /* fin bloc 1*/
```

Bloc imbriqués (2)

```
public static void main(String[] arg) {/*debut bloc 1*/
  int a=2;
  Terminal.ecrireStringln("valeur de a: "+ a );
  if (a==0){    /* debut bloc 2 */
      int b=3+a;
      Terminal.ecrireStringln("valeur de b: "+ b );
  }    /* fin bloc 2*/
  else      {    /* debut bloc 3*/
      int c=3+a;
      Terminal.ecrireStringln("valeur de c: "+ c);
  }    /* fin bloc 3*/
  Terminal.ecrireStringln("valeur de a: "+ a );
 }  /* fin bloc 1*/
```

- Les blocs 2 et 3 sont à l'intérieur du bloc 1.
- `a` est connue de 2–13 $\Rightarrow$ connue aussi dans blocs 2 et 3.
- `b` est connue de 5–7 ; `c` est connue de 9–11.

Bloc imbriqués : portée des variables

- La portée d'une variable est la partie d'un programme où elle est connue.
- La **portée** de `a` va de 2–13.
- La **portée** de `b` va de 5–7 ;
- La **portée** de `c` va de 9–11.

Boucles en Java.

Répéter des instructions

Problème : Écrire un programme qui affiche un rectangle de 4 lignes avec 4 étoiles. Nous réalisons 4 appels consécutifs à la méthode `Terminal.ecrireStringln`

```
public class Rectangle {
    public static void main (String[] args) {
            Terminal.ecrireStringln("****");
            Terminal.ecrireStringln("****");
            Terminal.ecrireStringln("****");
            Terminal.ecrireStringln("****");
    }
}
```

C'est possible, mais ce n'est pas très élégant !

Boucle `while`

Syntaxe :

```
while (<cond>)  {
    suiteInstructions
}
```

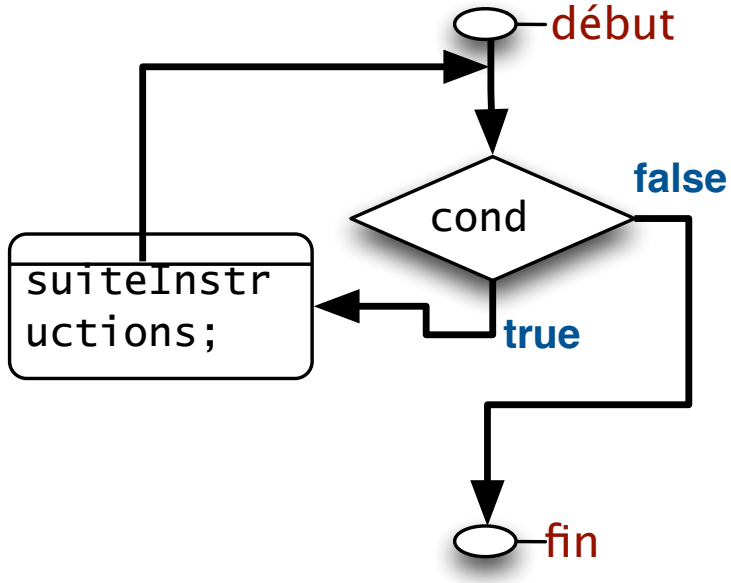
où cond est une expression booléenne.

On lit : « tant que **cond** est vrai, faire **suiteInstructions** »

1. cond est évalué avant chaque itération.
2. Si elle est vraie, on exécute suiteInstructions, puis le contrôle revient au point du test (point 1).
3. Sinon, le contrôle passe à l'instruction immédiatement après la boucle.

Boucle `while`

```
while (cond){
    suiteInstructions;
}
```

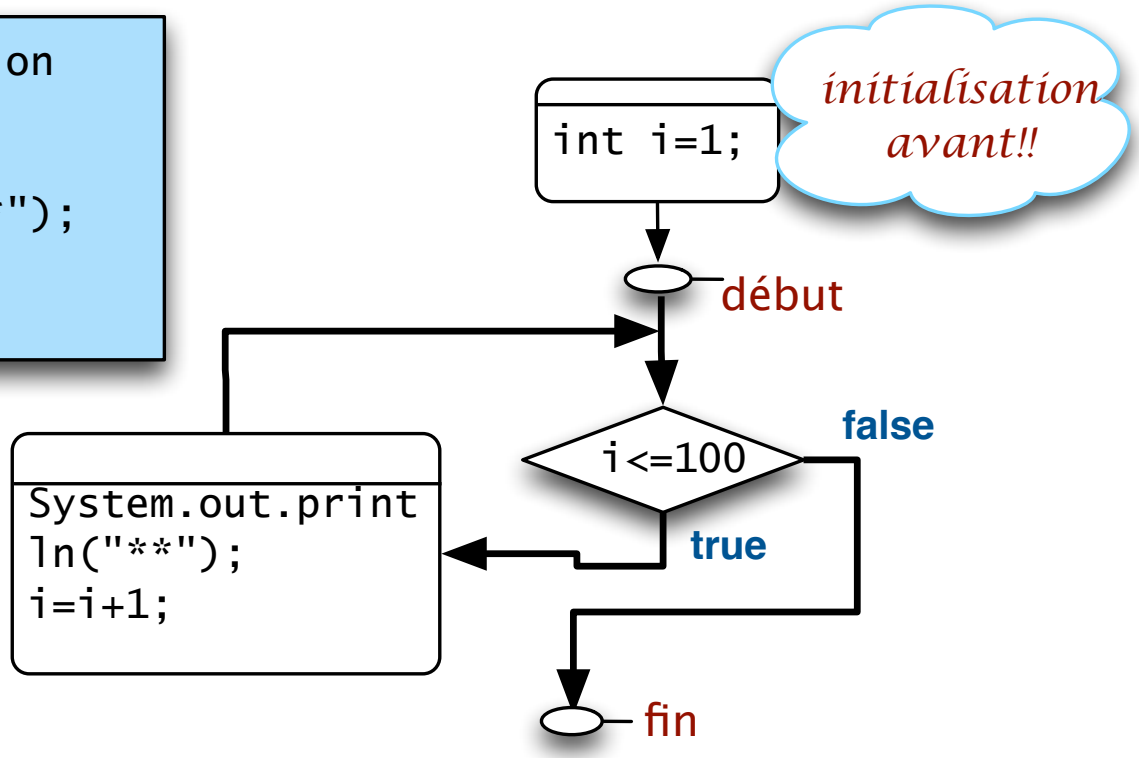


1. cond est la **condition** testée avant chaque tour de boucle
2. le bloc d'instructions est son **corps** ;

Exemple de `while` (1)

```
int i=1; // initialisation

while (i<=100){
  System.out.println("**");
  i=i+1;
}
```



Exemple de `while` (2)

```
int i=1;
while (i<=4)  {
    Terminal.ecrireStringln("****");
    i=i+1;
}
```

1. Tester (`1<= 4`) $\Rightarrow$ exécuter corps : (affichage 1ère ligne ; et `i` $\leftarrow$ 2)
2. Retour au point de test : (`2<= 4`). Affiche 2ème ligne, et `i` $\leftarrow$ 3
3. Tester (`3<= 4`). Affiche 3ème ligne, et `i` $\leftarrow$ 4
4. Tester (`4<= 4`). Affiche 4ème ligne, et `i` $\leftarrow$ 5
5. Le prochain test `5<= 4`, donne false.
6. Arrêt. L'exécution continue avec l'instruction après la boucle.

L'ordre des instructions est important

1. Problème : afficher tous les chiffres de 0 à 9.

```
int i=0;
while (i<=9)   {
    Terminal.ecrireInt(i);
    i=i+1;
}
```

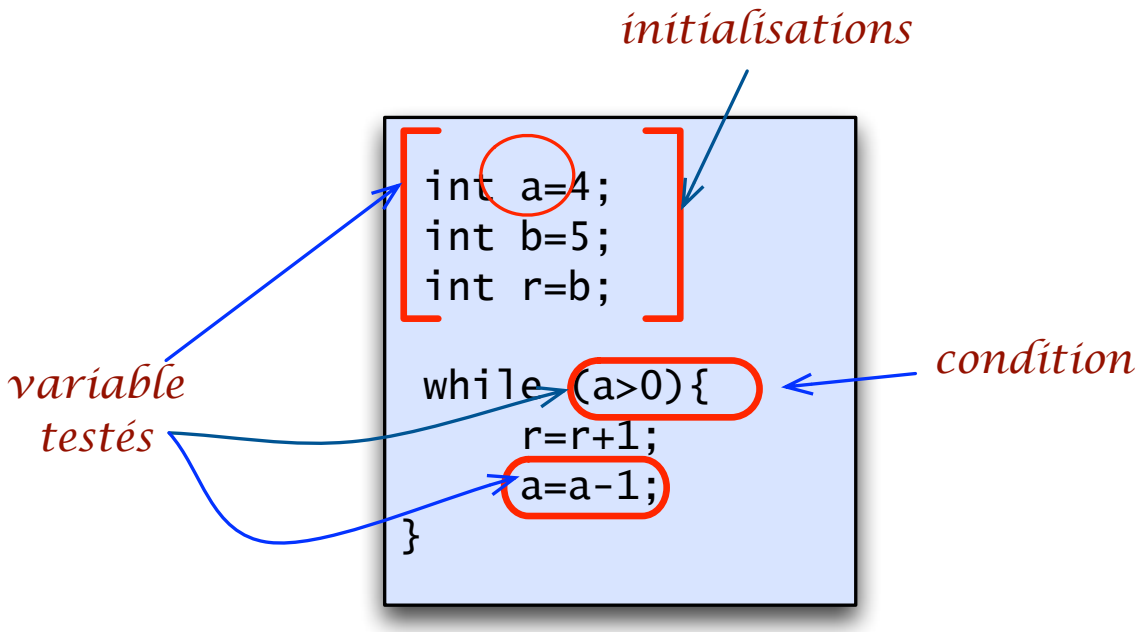
2. Qu'affiche celui-ci ?

```
int i=0;
while (i<=9)   {
    i=i+1;
    Terminal.ecrireInt(i);
}
```

Pourquoi se comportent-elles différemment ?

Éléments à identifier (et à maitriser)

- **Avant** : les initialisations de variables ;
- **Entête** : la condition et les variables testées ;
- **Corps** : les modifications sur les variables testées ;



Éléments à identifier (et à maitriser)

- **Avant** : les initialisations de variables (a,b,r) ;
- **Entête** : la condition `(a>0)` et les variables testées (a) ;
- **Corps** : les modifications sur les variables testées (`a=a-1`) ;

```
int a=4;   initialisations
int b=5;   initialisations
int r=b;   initialisations

while (a>0) {
    r=r+1;
    a=a-1;
}
```

Éléments importants sur la condition

Toutes les boucles testent une condition booléenne.

- Les variables de la condition : doivent être correctement initialisées ;
- la valeur de la condition à chaque tour est true ou false :
  - si true on exécute le corps ;
  - si false on termine la boucle ;
  - si false au départ, on n'exécute jamais le corps
  - si toujours true, on ne termine jamais la boucle !

Conclusion

- Les valeurs d'initialisation doivent permettre au corps de s'exécuter ;
- Le corps doit modifier les variables testées, de sorte que la condition « devienne un jour » false.

Éléments identifiés dans l'exemple

- Le corps est exécuté au moins une fois ?
- La valeur de la condition devient false un jour ? Quand ?
- Combien de tours fait cette boucle ?

```
int a=4;
int b=5;
int r=b;
while (a>0) {
    r=r+1;
    a=a-1;
}
```

Que calcule cette boucle (et dans quelle variable) ?

L'exemple avec une autre initialisation

- Le corps est exécuté au moins une fois ?
- La valeur de la condition devient false un jour ? Quand ?
- Combien de tours fait cette boucle ?

```
int a=0;
int b=5;
int r=b;
while (a>0) {
    r=r+1;
    a=a-1;
}
```

Et cette fois, qu'est-ce qui est calculé ?

Autres exemples

Dans tous les exemples, identifier les éléments importants et répondre aux questions les concernant. . .

Exemple 1 :

```
int i=0;
int n = 0;
while (i<=0){
    n= n + 10;
    i = i-1;
 }
 Terminal.ecrireStringln("Valeur de n:"+n);
 Terminal.ecrireStringln("Valeur de i:"+i);
```

Qu'affiche ce programme ?

Autres exemples (2)

Exemple 2 :

```
int i=0;
while (false){
    i= i+1;
 }
 Terminal.ecrireString("Valeur de i:"+i);
```

Qu'affiche ce programme ?

Autres exemples (3)

Exemple 3 :

```
int i=0;
while (i<0){
    i= i+1;
 }
 Terminal.ecrireString("Valeur de i:"+i);
```

et celui-ci ?

Autres exemples (4)

Exemple 4 :

```
int i=-2;
int n = 0;
while (i!=0){
    n= n + 10;
    i = i-1;
 }
 Terminal.ecrireStringln("Valeur de n:"+n);
 Terminal.ecrireStringln("Valeur de i:"+i);
```

et cet autre ?

Boucles qui ne terminent pas

Exemple 5 : Qu'affiche cette boucle ?

```
int i=4;
int n = 0;
while (i!=0){
    n= n + 10;
    i = i-3;
 }
 Terminal.ecrireStringln("Valeur de n:"+n);
 Terminal.ecrireStringln("Valeur de i:"+i);
```

Dans une boucle qui ne termine pas, soit la condition est inadaptée au calcul, soit dans le corps, nous avons oublié ou mal modifié les variables testées.

Les boucles qui ne terminent pas

Cette nouvelle version termine toujours. Pourquoi ?

```
int i=4;
int n = 0;
while (i>0){
    n= n + 10;
    i = i-3;
 }
 Terminal.ecrireStringln("Valeur de n:"+n);
 Terminal.ecrireStringln("Valeur de i:"+i);
```

Boucles qui ne terminent pas

```
boolean reponseValide=false;
while (!reponseValide){
    Terminal.ecrireString("Entrer taux: ");
    t = Terminal.lireInt();
}
```

Quel est le problème avec cette boucle ?

La valeur de la **variable testée** dans la condition de la boucle **ne change jamais** au cours de l'exécution :

- La première fois la condition testée est vraie, et donc le corps de la boucle est exécuté.
- Ensuite, comme la variable testée n'est jamais modifiée, la condition testée reste toujours vraie $\Rightarrow$ la boucle ne s'arrête jamais.

Exemple pour calculer une somme d'entiers

Problème : Calculer la somme d'une suite de nombres entiers saisis au clavier. Arrêt à la lecture de zéro.

Solution : Répéter la saisie d'un nombre n, puis son ajout à une variable `total` qui accumule la somme de tous les nombres saisis.

Algorithme : somme d'entiers

Entrées : *n* un entier d'une suite à saisir
Sortie : *total* entier

Algorithme :

1. Initialiser *n* par une saisie,
2. Initialiser *total* avec 0,
3. Tant que $n \not= 0$ faire :
    1. $total \leftarrow total + n$
    2. Saisir *n*
4. Afficher la valeur de *total*.

Dérouler l'algorithme

Supposons qu'on applique l'algorithme avec saisie de 5,3,7,0.

1.	$n \leftarrow 5$	Initialisations
2.	$total \leftarrow 0$	
3.	Test $(n = 5 \not= 0)? \Rightarrow$ *vrai*	$total \leftarrow 0 + 5\,;\, n \leftarrow 3$
3.	Test $(n = 3 \not= 0)? \Rightarrow$ *vrai*	$total \leftarrow 5 + 3\,;\, n \leftarrow 7$
3.	Test $(n = 7 \not= 0)? \Rightarrow$ *vrai*	$total \leftarrow 5 + 3 + 7\,;\, n \leftarrow 0$
3.	Test $(n = 0 \not= 0)? \Rightarrow$ *faux*	Arrêt
4.	Afficher *total*	15

A chaque itération, *total* accumule la somme de sa valeur initiale (0) et des valeurs prises par $n = 5, 3, 7$.

Version avec `while`

Tant que $n \not= 0$, faire :
- (a) $total \leftarrow total + n$
- (b) $n \leftarrow$ nouvelle saisie

} corps

condition d'entrée

```
total = 0;
while ( n !=0 )  {
   total = total + n;
   Terminal.ecrireString("Un entier? (fin avec 0)");
   n = Terminal.lireInt();
}
```

Le programme complet

```
public class Somme {
 public static void main (String[] args) {
   int n, total;
   Terminal.ecrireString("Entrez un entier (fin avec 0): ");
   n = Terminal.lireInt();
   total = 0;
   while ( n !=0 )  {
     total = total + n;
     Terminal.ecrireString("Un entier? (fin avec 0) ");
     n = Terminal.lireInt();
   }
   Terminal.ecrireStringln("Le total est: " + total);
}}
```

La boucle for

C'est un raccourci pour la forme suivante de boucle `while` :

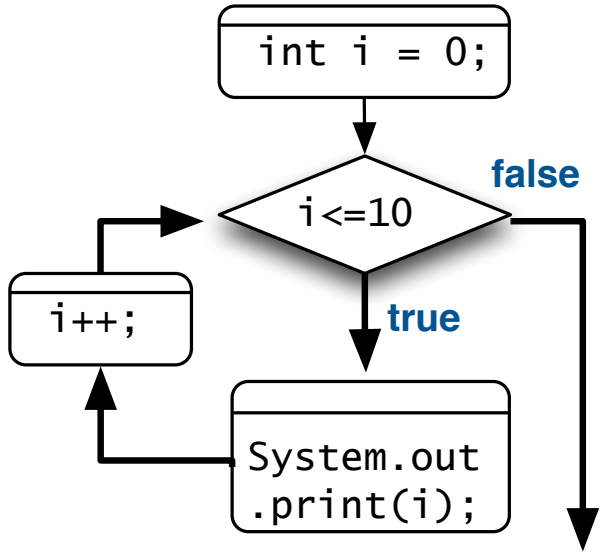
```
initialisation
while ( condition ) {
    instructions
    avancement
}
```

Une boucle for regroupe toujours ces 4 éléments.

La boucle for (schéma)

Boucle <<for>>

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



Exemples de boucle for (1)

On peut reformuler la boucle `while`

```
int i=1;
while (i<=4)  {
   Terminal.ecrireStringln("****");
   i=i+1;
}
```

avec la boucle `for` suivante :

```
for (int i=1;i<=4;i=i+1){
   Terminal.ecrireStringln("****");
}
```

Exemple de boucle for (2)

```
for (int i=1;i<=4;i=i+1){
    Terminal.ecrireStringln("****");
}
```

On lit : Répéter pour `i=1`, jusqu'à $i \leq 4$, l'instruction `Terminal.ecrireStringln("****")`.

Une boucle for possède :

- **une entête** : `for (int i=1;i<=4;i=i+1)`
- un **corps** : les instructions entre accolades.
- souvent, mais pas toujours : un **compteur** : ici, la variable `i`.

Exemple de boucle for (3)

```
for (int i=1;i<=4;i=i+1){
    Terminal.ecrireStringln("****");
}
```

Dans l'entête de cette boucle on retrouve :

- `int i=1` : **déclaration et initialisation** du compteur.
- `i<=4` : **condition d'entrée**. Testée avant chaque tour : si elle est vraie on exécute le corps, sinon, on arrête.
- `i=i+1` : **pas d'avancement**. Exécuté en dernier après chaque tour de boucle. Modifie la valeur du compteur.

Déroulement d'une boucle for

```
for (<initialisation>;<condition>;<avancement>){
    <instructions>
}
```

1. On exécute <initialisation> **une seule fois**.
2. On teste <condition>
    1. si elle est fausse, on arrête la boucle,
    2. si elle est vraie, on exécute le corps : <instructions>
    3. puis, on exécute <avancement>
3. On revient au point de test <condition>

Déroulement d'une boucle for (2)

```
for (int i=1;i<=4;i=i+1){
    Terminal.ecrireStringln("****");
}
```

- **On exécute une fois** : `i=1`. Puis, on teste `i<=4` $\Rightarrow$ true. Donc, on peut exécuter le corps,
- **1er** tour : on exécute `Terminal.ecrireStringln("****")`, puis `i = i+1`, donc `i` $\leftarrow$ 2
- **2ème tour** : `i=2`. On affiche une ligne puis `i` $\leftarrow$ 3
- **3ème** : `i=3`. Comment avant et `i` $\leftarrow$ 4
- **4ème** : `i=4`. Comment avant et `i` $\leftarrow$ 5
  Avant d'initier le prochain tour, on teste `i<=4` $\Rightarrow$ false. Donc, on arrête.

Exemple

Problème : Écrire un programme qui affiche un rectangle d'étoiles dont la hauteur est saisie (la largeur restera 4).

```
public class Rectangle2 {
  public static void main (String[] args) {
    int nlignes;
    Terminal.ecrireString("Combien␣de␣lignes␣?:␣");
    nlignes=Terminal.lireInt();
    for (int i=0;i<nlignes;i=i+1){
          Terminal.ecrireInt(i);
          Terminal.ecrireStringln("****");
    }
}}
```

- `nlignes` est le nombre de lignes à afficher,
- on utilise cette valeur dans la condition du `for`.

Boucle `do-while`

Il s'agit d'une boucle `while` où les instructions du corps sont exécutées avant de tester la condition de la boucle.

```
do
  {
     suiteInstructions
  }
while (c);
```

où *c* est une expression booléenne.
On lit : "faire suiteInstructions, tant que c est vrai"

1. *suiteInstructions* est exécuté,
2. *c* est évaluée à la fin de chaque itération : s'il est vrai, le contrôle revient à *suiteInstructions* (point 1).
3. Si *c* est faux, le contrôle du programme passe à l'instruction immédiatement après la boucle.

Utilisation

L'intérêt de `do-while` : pouvoir exécuter au moins une fois les instructions du corps avant de tester la condition d'arrêt.

```
Terminal.ecrireString("Un␣entier?␣(fin:␣0)␣");
n = Terminal.lireInt();  // Initialisation n
total = 0;
while ( n !=0 )  {
   total = total + n;
   Terminal.ecrireString("Un␣entier?␣(fin:␣0):␣");
   n = Terminal.lireInt(); // Nouvelle saisie de n
}
```

Dans ce programme : 2 lieux pour gérer la saisie de n.

Réformulation avec do-while

Dans cette version, un seul lieu de sasie pour n :

```
int n;
int total = 0;
do
  {
    Terminal.ecrireString("Un entier? (fin: 0): ");
    n = Terminal.lireInt();   // Saisie de n
    total = total + n;
  }
     while ( n !=0 );
```

Variable déclarée locale à une boucle `for`

- Les variables d'itération n'ont souvent d'intérêt que le temps d'exécuter la boucle.
- Une fois finie, c'est le résultat calculé qu'il est important de récupérer.
- La variable d'itération peut être :
  - **locale** à la boucle : si elle est déclarée dans l'entête de la boucle. Elle n'est alors **visible** que dans la boucle ;
  - **extérieurs** à la boucle : si elle est déclarée avant l'entête de la boucle. Elle est alors visible même après la fin de la boucle.

Exemple

Ici, i est locale à la boucle.

```
somme = 0;
for (int i = 1; i <= n; i++) {
        somme = somme + i;
}
int x = i; // erreur: i n'est pas connue ici
```

Ici, i est visible en dehors de la boucle.

```
somme = 0;
int i;
for (i = 1; i <= n; i++) {
        somme = somme + i;
}
int x = i; // OK
```

Compléments sur les boucles `for`

Structure de la boucle `for` en Java :

```
for (initInstrs ; boolExpr ; avancement) {
        calculInstrs
}
```

- `initInstrs` : initialisations
- `boolExpr` : condition de la boucle (expression booléenne) ;
- `avancement` : modifie les variables testées par la condition.
- `calculInstrs` : instructions à répéter.

Instructions d'initialisation (`initInstrs`)

On peut initialiser et/ou déclarer plusieurs variables dans l'entête d'un `for` :

1. déclaration/initialisations séparéee par des virgules ;
2. initialisations **multiples** sans déclarations :

```
int i, j;      // declaration
for (i=1, j=4; i<j; i++){ ... }
```

$\Rightarrow$ i et j doivent être déclarées auparavant ;

3. déclaration d'une ou plusieurs variables + valeur initiales :

```
for (int i=1, j=4; i<j; i++){ ... }
```

- les variables sont déclarées locales au for ;
- ne peuvent pas exister ailleurs dans le bloc englobant ;
- attention : i et j sont toutes les deux déclarées par le for.

Boucle avec initialisations multiples

Exemple : Calculer la puissance a^b pour deux nombres entiers a, b lus au clavier, et en supposant $b \geq 0$.

```
int p,i;
int a = // lu au clavier
int b = // lu au clavier
for (p = 1, i = b; i >= 1; i--) {
            p = p*a;
}
```

- donner l'évolution des variables en mémoire ;
- pourquoi la valeur initiale de p est 1 ?
- que se passe-t-il si b=0 ? le résultat est il correcte ?

Retour sur calcul du prix TTC

```
public class PrixTTC {
 public static void main (String[] args) {
   double pHT,pTTC;  char t;
   Terminal.ecrireString("Entrer␣le␣prix␣HT:␣");
   pHT = Terminal.lireDouble();
   Terminal.ecrireString("Taux?(normal=n,␣reduit=r)␣");
   t = Terminal.lireChar();
   if (t=='n'){
        pTTC=pHT + (pHT*0.196);
   } else {
        pTTC=pHT + (pHT*0.05);
   }
   Terminal.ecrireStringln("La␣somme␣TTC:␣"+ pTTC );
```

Problème : Pas de validation du taux : si l'on tape autre chose que 'n', le programme se comporte comme si on avait saisit 'r'.

Prix TTC avec validation de saisie

Solution : Répéter la saisie de la variable t tant que la réponse donnée est différente de 'n' et 'r'.

```
boolean reponseValide=false;
while (!reponseValide){
    Terminal.ecrireString("Entrer taux ");
    Terminal.ecrireString("(normal=n, reduit=r) ");
    t = Terminal.lireChar();
    if (t=='n' || t=='r') {reponseValide=true;
    }else{Terminal.ecrireStringln("Taux invalide.")};
 }
 // suite  des calculs pour le prix TTC
```

Déroulement de la boucle

- **Test de la condition** : `!reponseValide` est testée avant chaque itération.
- **Initialisations** : `reponseValide` doit avoir une valeur initiale **avant** la première itération.
- Nous posons : `reponseValide=false` car nous voulons au moins entrer une fois dans le corps de a boucle.

Déroulement de la boucle

- **Saisie et modifications** : si la valeur saisie pour t est valide on change `reponseValide` à true. Sinon, sa valeur reste à false et donc, un nouveau tour de boucle sera exécuté.

- **Etat à la sortie de la boucle** : On sort d'une boucle while dès que sa condition est fausse. Donc, en sortant de celle-ci nous pouvons assurer que :

  - `reponseValide` = true
  - t = 'r' ou t = 'n'

Le programme complet

```
public class PrixTTC {
 public static void main (String[] args) {
   double pHT,pTTC; char t;
   Terminal.ecrireString("Entrer le prix HT: ");
   pHT = Terminal.lireDouble();
   boolean reponseValide=false;
   while (!reponseValide){
      Terminal.ecrireString("Taux? (normal=n reduit=r) ");
      t = Terminal.lireChar();
      if (t=='n' || t=='r') {reponseValide=true;
      }
   }
   // Ici, on sait que t=='n' ou t=='r'
   if (t=='n'){ pTTC=pHT + (pHT*0.196);
   } else {      pTTC=pHT + (pHT*0.05);
   } Terminal.ecrireStringln("La somme TTC: "+ pTTC );
```

Boucles imbriquées

Une boucle peut contenir d'autres boucles :

```
// Boucle 1
for(int i=1; i<=4; i++) {
    instructions début boucle 1
    // Boucle 2
    for (int j=1; j<=2; j++) {
        instructions boucle 2
    }// Fin Boucle 2
  instructions suite boucle 1
} // Fin Boucle 1
```

- Boucle 2 est imbriquée dans Boucle 1.
- Boucle 1 est ici la boucle principale.

Comportement des boucles imbriquées

```
for(int i=1; i<=4; i++) {
    Terminal.ecrireStringln("Boucle␣1-->␣i="+i);
    for (int j=1; j<=2; j++) {
     Terminal.ecrireString("␣␣Boucle␣2-->␣i="+i);
     Terminal.ecrireStringln("␣j="+j);
    }// Fin Boucle 2
    Terminal.ecrireStringln("suite␣boucle␣1␣pour␣i="+i);
} // Fin Boucle 1
```

- Pour chaque valeur de i (boucle principale) :
  - on exécute les instructions avant boucle 2 ;
  - on exécute totalement la boucle 2 ;
  - on continue avec les instructions de boucle 1, après boucle 2.
- Qu'affiche ce programme ?
- Combien de fois s'affiche le message "Boucle 2 .." et "Boucle 1" ?

Affichages

```
for(int i=1; i<=4; i++) {
    Terminal.ecrireStringln("Boucle␣1␣-->␣i="+i);
    for (int j=1; j<=2; j++) {
     Terminal.ecrireString("␣␣Boucle␣2␣-->␣i="+i+"␣j="+j);
    }// Fin Boucle 2
    Terminal.ecrireStringln("suite␣boucle␣1␣pour␣i="+i);
} // Fin Boucle 1
```

```
Boucle 1 --> i=1
  Boucle 2 --> i=1 j=1
  Boucle 2 --> i=1 j=2
fin boucle 1 --> i=1
Boucle 1 --> i=2
  Boucle 2 --> i=2 j=1
  Boucle 2 --> i=2 j=2
fin boucle 1 --> i=2
Boucle 1 --> i=3
  Boucle 2 --> i=3 j=1
  Boucle 2 --> i=3 j=2
fin boucle 1 --> i=3
Boucle 1 --> i=4
  Boucle 2 --> i=4 j=1
  Boucle 2 --> i=4 j=2
fin boucle 1 --> i=4
```

Tous les affichages

```
Boucle 1 --> i=1
  Boucle 2 --> i=1 j=1
  Boucle 2 --> i=1 j=2
fin boucle 1 --> i=1
Boucle 1 --> i=2
  Boucle 2 --> i=2 j=1
  Boucle 2 --> i=2 j=2
fin boucle 1 --> i=2
Boucle 1 --> i=3
  Boucle 2 --> i=3 j=1
  Boucle 2 --> i=3 j=2
fin boucle 1 --> i=3
Boucle 1 --> i=4
  Boucle 2 --> i=4 j=1
  Boucle 2 --> i=4 j=2
fin boucle 1 --> i=4
```