

# Chapitre 3: Instructions conditionnelles et Boucles

## (NFA031 - Jour)

V. Aponte

Cnam

26 octobre 2020

# Les blocs d'instructions

## Bloc d'instructions

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

- délimiter les instructions d'une méthode,
- regrouper des instructions derrière if, else, for, etc. :

Exemples :

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

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

# Instruction conditionnelle

# Structure d'une conditionnelle

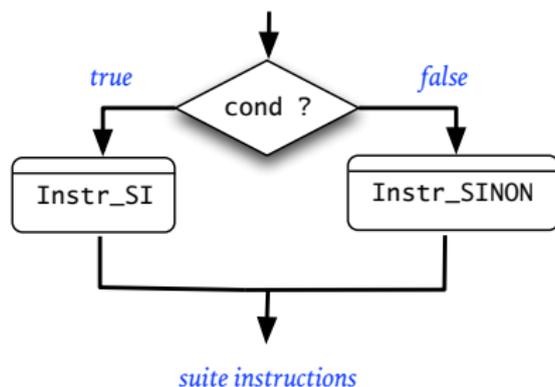
## Sert à spécifier :

- les différents cas d'un problème (« conditions »)
- et ce qu'il correspond d'exécuter dans chaque cas (« branches »).

## Est composé de :

- 1 une condition à tester (expression booléenne)
- 2 + 2 branches (blocs d'instructions), dont une seule est exécutée

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



# Exemple de conditionnelle (1)

```
int x = Terminal.lireInt();  
if (x >= 0) {  
    Terminal.ecrireStringln("positif_ou_nul.");  
}  
else {  
    Terminal.ecrireStringln("strictement_negatif.");  
}  
Terminal.ecrireString("on_passe_toujours_ici");
```

- 1 évaluation de la condition ( $x \geq 0$ )
  - 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.afficheStringln("positif ou nul.");
}
else {
    Terminal.afficheStringln("strictement_negatif.");
}
Terminal.afficheString("on_passe_toujours_ici");
```

- 1 évaluation de la condition ( $x \geq 0$ )
  - 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 negatif.");
}
Terminal.ecrireString("on_passe_toujours_ici");
```

- 1 évaluation de la condition ( $x \geq 0$ )
  - 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_negatif.");
}
Terminal.ecrireString("on passe toujours ici");
```

- 1 évaluation de la condition ( $x \geq 0$ )
  - 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 :  $p_{HT}$ , double ; taux :  $t$ , char ('n' ou 'r').
- sorties : prix TTC :  $p_{TTC}$ , double

## Algorithme

- 1 afficher un message demandant de saisir une somme HT.
- 2 recueillir la réponse dans  $p_{HT}$
- 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  $p_{TTC} = p_{HT} + (p_{HT} * 0.196)$
  - 2 Cas 2 : le taux réduit  $p_{TTC} = p_{HT} + (p_{HT} * 0.05)$
- 6 afficher  $p_{TTC}$

# 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 );
    }
}
```

# if sans else

Si l'on veut dire :

si condition est vraie alors faire ceci, sinon continuer le programme

on omettra le `else`.

Exemple :

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

Notez : si la condition est fausse, on passe à l'instruction suivante.

# Règles de style pour l'écriture de conditionnelles

- Les instructions d'un bloc (une par ligne) sont alignées quelques colonnes à droite par rapport au `if` ou `else` dont elles dépendent,
- Alignez chaque `else` sur la même colonne que le `if` auquel il correspond.
- Alignez les instructions des différents blocs sur la même colonne
- Mettez toujours des accolades pour les branches, même s'il n'y a qu'une seule instruction (Java autorise à ne pas en mettre dans ce cas);

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

## if suivi d'un autre if

Si deux `if` se suivent, à quel `if` correspond le `else` ?

- 1 Cas 1 : des accolades les séparent, le `else` correspond au premier `if`

```
if ( x > 0 ) {  
    if (y > 0) System.out.println("Cas_1");  
} else  
    System.out.println("Cas_2");
```

- 2 Cas 2 : sans accolades, le `else` correspond toujours au dernier `if` qui le précède.

```
if ( x > 0 )  
    if (y > 0)  
        System.out.println("Cas_1");  
else  
    System.out.println("Cas_2");
```

## Tests à la suite : `if-else-if`

Construction pour réaliser des tests de type « si-sinon-si » :

```
if (condition1){
    bloc1
}
else if (condition2) {
    bloc2
...
}
else if (conditionN) {
    blocN
}
else {
    blocN+1
}
```

# Structure et comportement d'un `if-else-if`

- Autant de `else if` que l'on veut, avec ou sans un `else` final,
- Les conditions sont testées dans l'ordre. La 1ère qui est vraie entraîne l'exécution du bloc associé et on va à la suite du programme
- Une condition n'est testée que si toutes les précédentes donnent false,
- Si aucune condition n'est vraie, c'est le bloc du `else` qui est exécuté.
- Cette instruction est un 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`  $\leq$  14
- "Il fait bon."  $\Rightarrow$   $15 \leq$  `temperature`  $\leq$  26
- "Il fait chaud."  $\Rightarrow$  `temperature`  $\geq$  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("Debit_:_:" + v);  
}  
System.out.println("Credit_:_:" + 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);    // Correct  
}  
Terminal.ecrireIntln(i);    // Erreur
```

# Bloc imbriqués

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

# Boucle `while`

## Syntaxe :

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

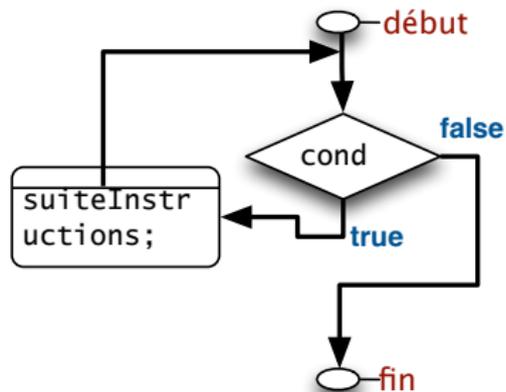
où cond est une expression booléenne.

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

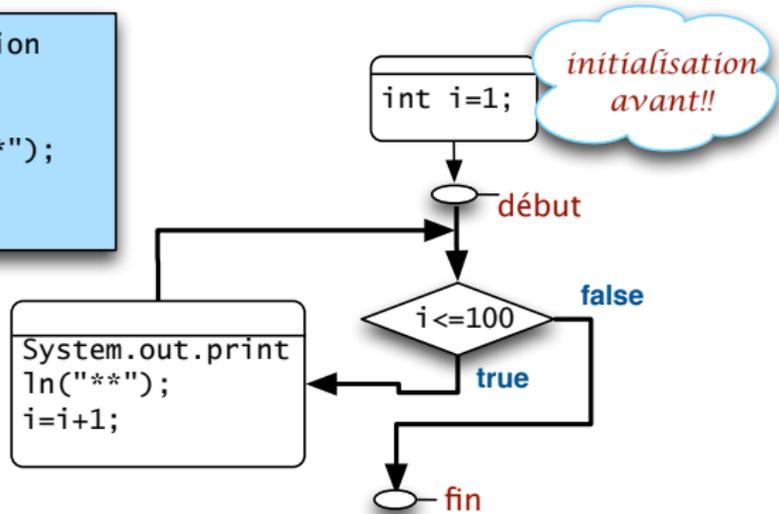
```
while (cond){  
    suiteInstructions;  
}
```



- 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 ( $1 \leq 4$ )  $\Rightarrow$  exécuter corps : (affichage 1ère ligne ; et  $i \leftarrow 2$ )
- 2 Retour au point de test : ( $2 \leq 4$ ). Affiche 2ème ligne, et  $i \leftarrow 3$
- 3 Tester ( $3 \leq 4$ ). Affiche 3ème ligne, et  $i \leftarrow 4$
- 4 Tester ( $4 \leq 4$ ). Affiche 4ème ligne, et  $i \leftarrow 5$
- 5 Le prochain test  $5 \leq 4$ , 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;
}
```

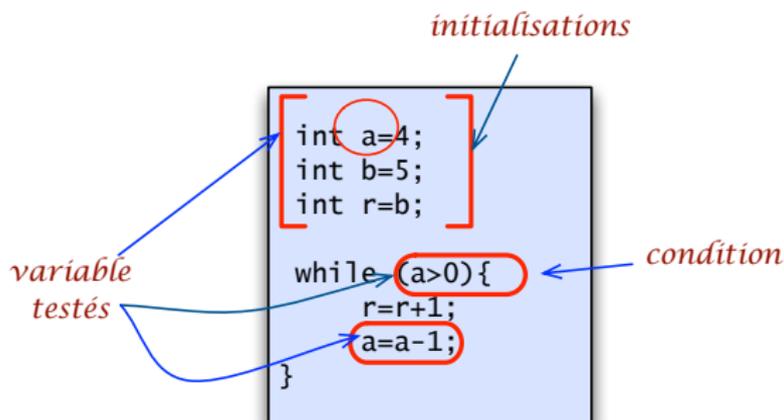
- 2 Qu'affiche celui-ci ?

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

Pourquoi se comportent-elles différemment ?

# éléments à identifier (et à maîtriser)

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



# éléments à identifier (et à maîtriser)

- **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 à identifier dans l'exemple

- Le corps est exécuté au moins une fois ?
- La valeur de la condition devient false (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 (dans la variable r) ?

## éléments identifiés dans l'exemple

- Le corps est exécuté au moins une fois ? **Oui, car  $a = 4 > 0$**
- La valeur de la condition devient false (quand) ? **Oui, quand  $a = 0$**
- Combien de tours fait cette boucle ? **4 tours,  $a = 4, 3, 2, 1$**

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

Que calcule cette boucle (dans la variable r) ?  **$r = b + a$**

## L'exemple avec une autre initialisation

- Le corps est exécuté au moins une fois ?
- La valeur de la condition devient false (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 (2)

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 : oubli de modification

```
int i=4;
int n = 0;
while (i>0) {
    n= n + 10;
}
Terminal.ecrireStringln("Valeur_de_n:"+n);
Terminal.ecrireStringln("Valeur_de_i:"+i);
}
```

Quel est le problème avec cette boucle ?

La valeur de la **variable testée** dans la condition de la boucle **ne change jamais dans le corps**

- 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 : 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  $n \neq 0$  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.  $n \leftarrow 5$
2.  $total \leftarrow 0$
3. Test ( $n = 5 \neq 0$ )?  $\Rightarrow$  *vrai*
3. Test ( $n = 3 \neq 0$ )?  $\Rightarrow$  *vrai*
3. Test ( $n = 7 \neq 0$ )?  $\Rightarrow$  *vrai*
3. Test ( $n = 0 \neq 0$ )?  $\Rightarrow$  *faux*
4. Afficher  $total$

Initialisations

$total \leftarrow 0 + 5; n \leftarrow 3$
------------------------------------------

$total \leftarrow 5 + 3; n \leftarrow 7$
------------------------------------------

$total \leftarrow 5 + 3 + 7; n \leftarrow 0$
----------------------------------------------

Arrêt

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  $n \neq 0$ , faire :

(a)  $total \leftarrow total + n$

(b)  $n \leftarrow$  nouvelle saisie

condition d'entrée

corps

```
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);
    }
}
```

# Boucle for

# La boucle for

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

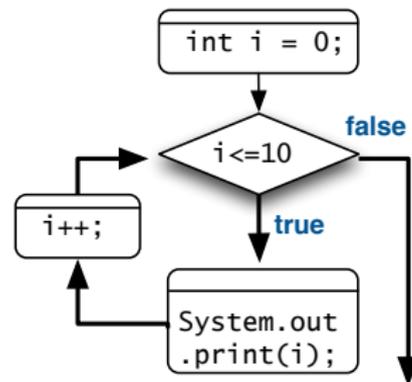
```
initialisation
while ( condition ) {
    instructions
    avancement
}
```

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  $i \leq 4 \Rightarrow \text{true}$ . 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  $i \leq 4 \Rightarrow \text{false}$ . 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 nlines;
        Terminal.ecrireString("Combien_de_lignes?:_");
        nlines=Terminal.lireInt();
        for (int i=0;i<nlines;i=i+1){
            Terminal.ecrireInt(i);
            Terminal.ecrireStringln("****");
        }
    }
}
```

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

# Boucle `do-while`

# 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.afficheString("Un entier?_(fin:_0)_");
n = Terminal.lireInt(); // Initialisation n
total = 0;
while ( n !=0 ) {
    total = total + n;
    Terminal.afficheString("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 saisie 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 );
```

# Compléments sur la boucle for

## Variable « d'itération » d'une boucle `for`

Dans une boucle `for`, il y a souvent une variable dite « d'itération » pour contrôler l'entrée et/ou l'arrêt de la boucle.

- Ces variables 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. Elle n'est alors **visible** que dans la boucle ;
  - ▶ **extérieure à la boucle** : si elle est déclarée avant l'entête. Elle est alors visible même après la fin de la boucle.

# Exemple de localité ou pas de variables d'itération

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 non locale et donc visible en dehors de la boucle.

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

# Initialisations dans une boucle (for)

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

- 1 les déclaration/initialisations seront séparées par des virgules ;
- 2 possibilité d'initialisations **multiples** sans déclarations :

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

- ▶ i et j devront être **déclarées au préalable** ;

- 3 possibilité de déclarations multiples + initialisations :

```
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.

# Exemple de boucle for 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 ?

# Un patron de programmation : boucle de validation d'une lecture

# Tester si une entrée est valide

Il est souvent utile de tester la validité d'une entrée **avant d'entamer des calculs sur celle-ci** :

```
System.out.print("Entrez_un_nombre_positif_");  
int n = Terminal.lireInt();  
... calculs avec n ....
```

*Avant d'entamer les calculs, que faire si n est invalide ?*

- **On termine le programme** : évite les erreurs de calcul avec données incohérentes, mais il faudra relancer le programme si on veut donner une nouvelle entrée.
- **Boucle de lecture** : on re-demande une nouvelle entrée, et cela tant que celle donnée est invalide.

# Patron de boucle de validation

- 1 **variable drapeau (booléen initialisé à false)** : on suppose que l'entrée est invalide afin d'entrer dans la boucle ;
- 2 **corps de la boucle** : on re-demande une nouvelle entrée, et cela tant que celle donnée est invalide.

```
boolean entreeValide = false;  
while (!entreeValide) {  
    System.out.print("Entrez_un_nombre_positif_");  
    int n = Terminal.lireInt();  
    if (n > 0) {  
        entreeValide = true; // au prochain tour on termine  
    } else {  
        System.out.print("Entree_invalide,_recommencez.");  
    }  
} // Fin boucle de validation  
// Ici, n a une valeur positive  
... calculs avec n ....
```

## Exemple : Prix TTC avec validation de lecture

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 la boucle.

# Déroulement de la boucle

- **Lecture et modifications** : si la valeur lue 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

# Boucles imbriquées

Une boucle **imbriquée** est une boucle interne à une autre boucle :

```
// Boucle A
for(int i=1; i<=4; i++) {
    A1; A2; // 1ère, 2ème instructions de A
    // Boucle B (3ème instruction de A)
    for (int j=1; j<=2; j++) {
        B1; B2;
    } // Fin B
    A4; // dernière instruction de A
} // Fin A
```

- La Boucle A contient les instructions  $A_1$  ;  $A_2$  ;  $B$  ;  $A_4$
- La Boucle B est la 3ème instruction de A . B est **imbriquée** dans A,

# Comportement d'une boucle imbriquée

```
for(int i=1; i<=4; i++) { // Boucle A
    A1; A2; // 1ère, 2ème instructions de A
    // Boucle B (3ème instruction de A)
    for (int j=1; j<=2; j++) {
        B1; B2;
    } // Fin B
    A4; // dernière instruction de A
} // Fin A
```

Chaque tour de A exécute successivement  $A_1$ ;  $A_2$ ; B et enfin  $A_4$ , où :

- 1  $A_1$ ;  $A_2$  : sont les instructions de A **avant** B;
- 2 B : est une instruction à exécuter en totalité, c.a.d., jusqu'à sa terminaison. On exécute donc **toutes les itérations de B** (donc potentiellement plusieurs fois  $B_1$ ;  $B_2$ );
- 3  $A_4$  : ce sont les instructions de A **après terminaison de B**. On exécute  $A_4$  seulement si la condition de B est false!

# Exemple de boucle imbriquée

```
for(int i=1; i<=4; i++) { // Boucle A
    A1 System.out.println("Boucle_A:_ (i="+i+" )");
    B   for (int j=1; j<=2; j++) { // Boucle B
        System.out.print(" Boucle B: (i="+i);
        System.out.println(", j="+j+"");
    } // Fin B
    A3 System.out.println("Fin_A_pour_i="+i);
} // Fin A
```

- **Boucle A** : pour chaque valeur de  $i$  , on exécute  $A_1$  ;  $B$  ;  $A_3$  :
  - 1  $A_1$  : les instructions de A **avant** B ;
  - 2  $B$  : on exécute **toutes les itérations de B** (jusqu'à sa terminaison) ;
  - 3  $A_3$  : les instructions de A **après terminaison de B**
- Qu'affiche ce programme ?
- Combien de fois s'affiche le message "Boucle B .." et "Boucle A" ?

# Affichages

```
Boucle A : (i=1)
  Boucle B: (i=1, j=1)
  Boucle B: (i=1, j=2)
Fin A pour i=1
Boucle A : (i=2)
  Boucle B: (i=2, j=1)
  Boucle B: (i=2, j=2)
Fin A pour i=2
Boucle A : (i=3)
  Boucle B: (i=3, j=1)
  Boucle B: (i=3, j=2)
Fin A pour i=3
Boucle A : (i=4)
  Boucle B: (i=4, j=1)
  Boucle B: (i=4, j=2)
Fin A pour i=4
```