Chapitre 5

Faire des répétitions

On retrouve les même types de boucles qu'en C++ :

- La boucle **`répéter`** … **`tant que`**
  `do` … **`while ()`**

- La boucle **`répéter pour`**…
  `for ()` …

- La boucle **`tant que`** … **`faire`**
  `while () do` …

1. La boucle `do…While`

La boucle **`do…while`** est une structure répétitive, dont les instructions sont exécutées avant même de tester la condition d'exécution de la boucle.

Syntaxe :

```
do
   Une seule instruction
while (expression conditionnelle);
```

S'il y a plusieurs instructions, il faut utiliser les accolades { } :

```
do
{
   Plusieurs instructions
} while (expression conditionnelle);
```

1.2 Principes de fonctionnement

La boucle **`do`**…**`while`** s'exécute selon les principes suivants :

- Les instructions situées à l'intérieur de la boucle sont exécutées tant que l'expression conditionnelle placée entre parenthèses après **`while`** est vraie.

- Les instructions sont examinées au moins une fois (expression conditionnelle en fin de boucle).

- Si la condition mentionnée entre parenthèses reste toujours vraie, les instructions de la boucle sont répétées à l'infini. On dit que le programme «boucle».

- Une instruction modifiant le résultat du test de sortie de boucle est placée à l'intérieur de la boucle, de façon à stopper les répétitions au moment souhaité.

- Un point-virgule est placé à la fin de l'instruction :

```
do
   ...
while (expression);
```

1.3 Exemple

Demande d'un nombre positif ou nul. Si le nombre est négatif, il faut le saisir de nouveau.

```
public class Positif
{
  public static void main (String [] arg)
  {
     float Nb;
     Scanner clavier = new Scanner(System.in);
     do
     {
        System.out.print("Entrez un nombre >= 0");
        Nb=clavier.nextFloat();
     }
     while (Nb < 0);
     System.out.print("Le nombre positif est:" + Nb);
  }
}
```

2. La boucle `while`

Le langage Java propose une autre structure répétitive, analogue à la boucle **`do`**…**`while`**, mais dont la décision de poursuivre la répétition s'effectue en début de boucle. Il s'agit de la boucle **`while`**.

2.1 Syntaxe

```
while (expression conditionnelle)
   Une seule instruction
```

Si la boucle est composée d'au moins deux instructions, celles-ci sont encadrées par des accolades de façon à déterminer où débute et se termine la boucle :

```
while (expression conditionnelle)
{
   plusieurs instructions
}
```

2.2 Principe de fonctionnement

Le terme **`while`** se traduit par « tant que ». La structure répétitive s'exécute selon les principes suivants :

- Tant que l'expression à l'intérieur des parenthèses reste vraie, la ou les instructions composant la boucle sont exécutées.

- Le programme sort de la boucle dès que l'expression à l'intérieur des parenthèses devient fausse.

– Une instruction est placée à l'intérieur de la boucle pour modifier le résultat du test à l'entrée de la boucle, de façon à stopper les répétitions.

– Si l'expression à l'intérieur des parenthèses est fausse dès le départ, les instructions ne sont jamais exécutées.

– A l'inverse de la boucle **`do..while`**, il n'y a pas de points-virgule à la fin de l'instruction :

```
while (expression)
    ...
```

2.3 Exemple

Écrire un programme **`Devinette`**, qui tire un nombre au hasard entre 0 et 10 et demande à l'utilisateur de trouver ce nombre. Pour ce faire, la méthode est la suivante :

- Tirer au hasard un nombre entre 0 et 10 (avec la fonction **`Math.random()`**)
- Lire un nombre au clavier
- Tant que le nombre lu est différent du nombre tiré au hasard :
  - Lire un nombre
- Afficher un message de réussite.

```java
// A taper dans un fichier Devinette.java
public class Devinette
{
  public static void main (String [] arg)
  {
    int nombreHasard, nombreLu;
    Scanner clavier = new Scanner(System.in);

    nombreHasard = (int)(10*Math.random());
    System.out.print("Devinez un nombre entre 0 et 10");
    System.out.print("Votre choix : ");
    nombreLu = clavier.nextInt();
    while (nombreLu != nombreHasard)
    {
      System.out.print("Votre choix : ");
      nombreLu = clavier.nextInt();
    }
    System.out.print("Bravo! C'était " + nombreLu);
  } // Fin du main ()
}   // Fin de la classe Devinette
```

3. La boucle `for`

L'instruction **`for`** permet d'écrire des boucles dont on connaît à l'avance le nombre d'itérations (de tours) à exécuter.

3.1 Syntaxe

- Si il y a une seule instruction :

```
for (initialisation ; condition ; incrément)
   Une seule instruction
```

- S'il y a plusieurs instructions :

```
for (initialisation ; condition ; incrément)
{
   Plusieurs instructions
}
```

```
for (initialisation ; condition ; incrément)
```

Les termes **initialisation**, **condition** et **incrément** sont des instructions séparées obligatoirement par des points-virgules (;).

Ces instructions définissent une variable (ou indice), qui contrôle le bon déroulement de la boucle :

`initialisation` permet d'initialiser la variable représentant l'indice de la boucle (exemple : **`i=0`**, **`i`** étant l'indice).

`condition` définit la condition à vérifier pour continuer à exécuter la boucle (exemple : **`i< 10`**). Elle est examinée avant chaque tour de boucle, y compris au premier tour de boucle.

`incrément` est l'instruction qui permet d'augmenter ou de diminuer la valeur de la variable d'indice selon un **pas d'incrémentation** (exemple : **`i=i+2`**). Cette instruction est exécutée à la fin de chaque tour de boucle.

3.2 Principes de fonctionnement

Les boucles **`for`** réalisent un nombre précis de boucles dépendant de la valeur initiale, de la valeur finale et du pas d'incrémentation.

```
int  i;
char c;
```

	Valeur initiale	Valeur finale	Pas d'incrémentation	Nbre de boucles	Valeurs prises par i ou c dans la boucle
`for(i=0; i<5; i=i+1)`	0	4	1	5	0,1,2,3,4
`for(i=4; i<=12; i=i+2)`	4	12	2	5	4,6,8,10,12
`for(c='a'; c<'f'; c=c+1)`	'a'	'e'	1	5	a,b,c,d,e
`for(i=5; i>0; i=i-1)`	5	0	-1	5	5,4,3,2,1

Remarques :

`i=i+1`

s'écrit aussi : `i+=1`

ou : `i++`

`i=i-1`

s'écrit aussi : `i-=1`

ou : `i--`

3.3 Exemple

Afficher les carrés des nombres compris entre **`nbDebut`** et **`nbFin`**.

```
// A taper dans un fichier Carre.java
public class Carre
{
  public static void main (String [] arg)
  {
    int i, nbDebut, nbFin;
    Scanner clavier = new Scanner(System.in);

    System.out.print("Premier nombre à mettre au carré: ");
    nbDebut =  clavier.nextInt();

    System.out.print("Dernier nombre à mettre au carré: ");
    nbFin =  clavier.nextInt();

    for (i=nbDebut; i<=nbFin; i++)
    {
      System.out.println("Le carré de " + i + " est :"
                         + (i*i));
    }
  } // Fin du main ()
}   // Fin de la classe Carre
```

Remarques :

- si **`nbDebut > nbFin`**

  alors pas de tour de boucle

- si **`nbDebut == nbFin`**

  un tour de boucle pour **`i = nbDebut`**

- sinon

  des tours de boucles de **`i=nbDebut`** jusqu'à **`i=nbFin`**.

4. Quelle boucle choisir ?

Chacune des trois boucles étudiées dans ce chapitre permet de répéter un ensemble d'instructions.

Cependant, les différentes propriétés de chacune d'entre elles font que le programmeur utilisera un type de boucle plutôt qu'un autre, suivant le problème à résoudre.

4.1 Choisir entre `do…while` et `while`

On utilise l'une ou l'autre quand le nombre de répétition n'est pas connu.

La position du test de sortie est importante :

- **`do…while`** : test à la fin : les instructions à l'intérieur de la boucle sont exécutées au moins une fois.

- **`while`** : test au début : on peut ne pas entrer dans la boucle.

4.2 Choisir entre `for` et `while`

Les boucles **`for`** et **`while`** sont identiques quant au placement du test.

On utilise la boucle for quand on connaît à l'avance le nombre d'itérations à exécuter.