

LES STRUCTURES CONDITIONNELLES ET LES STRUCTURES ITERATIVES

Nous abordons ici l'un des chapitres les plus importants : les conditions sont une autre notion fondamentale de la programmation. En effet, ce qui va être développé ici s'applique à énormément de langages de programmation, et pas seulement à Java.

Dans une classe, la lecture et l'exécution se font de façon séquentielle, c'est-à-dire ligne par ligne. Avec les conditions, nous allons pouvoir gérer différents cas de figure sans pour autant lire tout le code. Vous vous rendrez vite compte que tous vos projets ne sont que des enchaînements et des imbrications de conditions et de boucles

Le rôle des boucles est de répéter un certain nombre de fois les mêmes opérations. Tous les programmes, ou presque, ont besoin de ce type de fonctionnalité. Nous utiliserons les boucles pour permettre à un programme de recommencer depuis le début, pour attendre une action précise de l'utilisateur, parcourir une série de données, etc. Une boucle s'exécute tant qu'une condition est remplie.

1. Les opérateurs logiques

Avant de pouvoir créer et évaluer des conditions, vous devez savoir que pour y parvenir, nous allons utiliser ce qu'on appelle des « opérateurs logiques ». Ceux-ci sont surtout utilisés lors de conditions pour évaluer différents cas possibles. Voici les différents opérateurs à connaître :

- « == » : permet de tester l'égalité.
- « != » : permet de tester l'inégalité.
- « < » : strictement inférieur.
- « <= » : inférieur ou égal.
- « > » : strictement supérieur.
- « >= » : supérieur ou égal.
- « && » : l'opérateur ET. Il permet de préciser une condition
- « || » : le OU. Même combat que le précédent.
- « ? : » : l'opérateur ternaire. Pour celui-ci, vous comprendrez mieux avec un exemple

2. La structure if... else

```
if( /*condition*/ ) {  
    //Exécution des instructions si la condition est remplie  
} else {  
    //Exécution des instructions si la condition n'est pas remplie  
}
```

Java code

✚ Cela peut se traduire par « si...sinon... ».

Le résultat de l'expression évaluée par l'instruction **if** sera un **boolean**, donc soit **true**, soit **false**. La portion de code du bloc if ne sera exécutée que si la condition est remplie. Dans le cas contraire, c'est le bloc de l'instruction **else** qui le sera.

Mettons notre petit exemple en pratique :

```
int i = 10;  
if( i < 0 )  
    System.out.println( "Le nombre est négatif" );  
else  
    System.out.println( "Le nombre est positif" );
```

Java code

- ✚ Lorsque le bloc d'instruction d'une structure est constitué d'une seule instruction, vous pouvez ignorer les « { » et « } »
- ✚ Après un **else**, on peut mettre une nouvelle condition **if...**
- ✚ Les conditions peuvent être multiples, juste il faut les séparer par l'un des opérateurs logiques : « && » ou « || »

Exemple vérifier si un nombre est dans l'un des intervalles [0, 50[ou [50, 100]

```
int i = 10;  
if( 0 <= i && i < 50 )  
    System.out.println( "Le nombre est dans l'intervalle [0, 50[ " );  
else if( 50 <= i && i <= 100 )  
    System.out.println( "Le nombre est dans l'intervalle [50, 100]" );  
else  
    System.out.println( "Le nombre n'est dans aucun intervalle." );
```

Java code

3. La structure switch

Pour éviter trop de répétition de **else... if...** la structure **switch** est l'idéal. Elle permet de tester les valeurs d'une même variable, afin d'exécuter tel ou tel autre bloc d'instruction. On a la syntaxe :

```
switch ( /*Variable*/ ) {  
    case /*valeur 1*/ :  
        /*Action à exécuter si variable vaut valeur 1*/ ;  
        break;  
    case /*valeur n*/ :  
        /*Action à exécuter si variable vaut valeur n*/ ;  
        break;  
    default:  
        /*Action à exécuter si variable vaut aucune valeur si haut */ ;  
}
```

Java code

✚ L'instruction **break**; permet d'interrompre la boucle. Il peut arriver qu'on n'ait pas besoin de lui.

Exemple : pour un examen ou on peut obtenir uniquement 0 ou 10 points, on va écrire un programme qui permet d'apprécier une note :

```
int note = 10; //On imagine que la note maximale est 20  
switch (note) {  
    case 0:  
        System.out.println( "Ouch !" );  
        break;  
    case 10 :  
        System.out.println( "Vous avez juste la moyenne." );  
        break;  
    case 20 :  
        System.out.println( "Parfait !" );  
        break;  
    default:  
        System.out.println( "Il faut davantage travailler." );  
}
```

Java code

4. La condition ternaire

Cette structure est une sorte de forme réduite de la structure **if... else...** la structure permet de tester si une condition est remplie puis effectuer une instruction, sinon, exécuter une autre instruction.

On a la syntaxe générale :

```
variable = « condition » ? /*action si condition vrai*/ : /*action si condition faux*/ ;
```

Java code

- ✚ Se raccourcir est très pratique pour les gros programmes lorsqu'on évite la saturation. Plus tôt très simple à utiliser.
- ✚ Il est possible que l'action d'un ternaire soit un autre ternaire

Exemple :

```
int x = 10, y = 20;  
int max = (x < y) ? y : x ; //Maintenant, max vaut 20
```

Java code

Exemple avec plusieurs ternaire :

```
int x = 10, y = 20;  
int max = (x < y) ? ((y < 10) ? y % 10 : y) : x ; //Maintenant, max vaut 20
```

Java code

5. La boucle while

Pour décortiquer précisément ce qui se passe dans une boucle, nous allons voir comment elle se construit ! Une boucle commence par une déclaration : ici **while**. Cela veut dire, à peu de chose près, « tant que ». Puis nous avons une condition : c'est elle qui permet à la boucle de s'arrêter. Une boucle n'est utile que lorsque nous pouvons la contrôler, et donc lui faire répéter une instruction un certain nombre de fois. C'est à ça que servent les conditions. Ensuite nous avons une ou plusieurs instructions : c'est ce que va répéter notre boucle (il peut même y avoir des boucles dans une boucle !). Syntaxe :

```
while ( /* Condition */ ) {  
    //Instructions à répéter  
}
```

Java code

Nous allons travailler sur un exemple concret mais d'abord, réfléchissons à « comment notre boucle va travailler ». Pour cela, il faut déterminer notre exemple. Nous allons afficher « Bonjour », prénom qu'il faudra taper au clavier ; puis nous demanderons si l'on veut recommencer. Pour cela, il nous faut une variable qui va recevoir le prénom, donc dont le type sera String, ainsi qu'une variable pour récupérer la réponse. Et là, plusieurs choix s'offrent à nous : soit un caractère, soit une chaîne de caractères, soit un entier. Ici, nous prendrons une variable de type char. C'est parti !

```
String prenom;  
char reponse = 'O';  
Scanner sc = new Scanner(System.in);  
while (reponse == 'O') {  
    System.out.println( "Donnez un prénom : " );  
    prenom = sc.nextLine();  
    System.out.println( "Bonjour " + prenom + ", comment vas-tu ?" );  
    //Sans ça, nous n'entrerions pas dans la deuxième boucle  
    reponse = ' ';  
    //Tant que la réponse n'est pas O ou N, on repose la question  
    while(reponse != 'O' && reponse != 'N') {  
        //On demande si la personne veut faire un autre essai  
        System.out.println( "Voulez-vous réessayer ? (O/N)" );  
        reponse = sc.nextLine().charAt(0);  
    }  
}  
System.out.println( "Au revoir..." );
```

Java code

6. La boucle **do... while**

Son fonctionnement est identique à celui de la boucle **while** à deux détails près :

- La boucle **do... while** s'exécutera au moins une fois, contrairement à sa sœur. C'est-à-dire que la phase de test de la condition se fait à la fin, car la condition se met après le **while**.
- C'est une différence de syntaxe, qui se situe après la condition du **while**. Vous voyez la différence ? Oui ? Non ? Il y a un « ; » après le **while**. C'est tout ! Ne l'oubliez cependant pas, sinon le programme ne compilera pas.

Syntaxe :

```
do{  
    //Instructions à répéter  
} while ( /* Condition */ );
```

Java code

Mis à part ces deux éléments, ces boucles fonctionnent exactement de la même manière. D'ailleurs, refaisons notre programme précédent avec une boucle **do... while**.

```
String prenom = new String();  
//Pas besoin d'initialiser : on entre au moins une fois dans la boucle !  
char reponse = "  
  
Scanner sc = new Scanner(System.in);  
  
do{  
    System.out.println( "Donnez un prénom : " );  
    prenom = sc.nextLine();  
    System.out.println( "Bonjour " + prenom + ", comment vas-tu ?" );  
  
    do{  
        System.out.println( "Voulez-vous réessayer ? (O/N)" );  
        reponse = sc.nextLine().charAt(0);  
    } while(reponse != 'O' && reponse != 'N' );  
} while (reponse == 'O' );  
  
System.out.println( "Au revoir..." );
```

Java code

7. La boucle for

Cette boucle est un peu particulière puisqu'elle prend tous ses attributs dans sa condition et agit en conséquence. Je m'explique : jusqu'ici, nous avons fait des boucles avec :

- Déclaration d'une variable avant la boucle ;
- Initialisation de cette variable ;
- Incrémentation de celle-ci dans la boucle

Contrairement aux deux dernières boucles, la boucle for est utilisée lorsque le nombre d'itération est connu.

Syntaxe :

```
for( /*initialisation*/ ; /*condition*/ ; /*Incrémentation*/ ) {  
    //instructions à itérer  
}
```

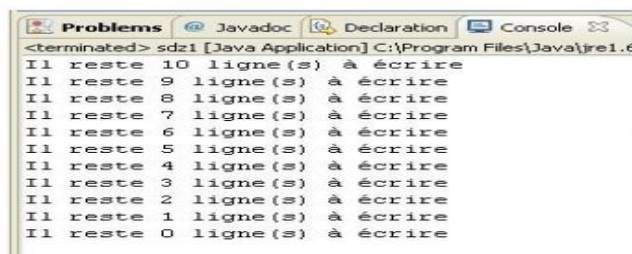
Syntaxe 2 à utiliser pour parcourir des listes de données (tableaux, listes chaînées...). Cette syntaxe sera mieux étudiée plus tard :

```
for( type_de_données_de_la_liste element : liste ) {  
    //instructions à itérer  
}
```

Exemple :

```
for(int i = 10; i >= 0; i--)  
    System.out.println("Il reste "+i+" ligne(s) à écrire");
```

Java code



The screenshot shows a Java IDE window with a console tab. The console output displays the result of a for loop that counts down from 10 to 0. Each iteration prints a line: "Il reste 10 ligne(s) à écrire", "Il reste 9 ligne(s) à écrire", ..., "Il reste 1 ligne(s) à écrire", and finally "Il reste 0 ligne(s) à écrire". The text "Boucle for avec décrémentation" is visible in the background of the console output.