Algorithmie et Programmation

Chapitre 4 : les instructions de répétition

Sommaire :

---

1. De quoi parle-t-on ?

Il est courant, dans les programmes informatiques (comme dans la vie de tous les jours), que certaines actions doivent se répéter plusieurs fois. Pour coder ce genre de comportements, nous utilisons des « instructions de répétition » (ou « boucles »).

Nous allons diviser ces boucles en deux familles :

- 1ère famille : on ne connaît pas à l'avance le nombre de fois qu'une opération sera répétée (la boucle **while**)

- 2ème famille : on connaît à l'avance le nombre d'itérations (la boucle **for**)

Prenons le cas d'une saisie au clavier (une lecture). On pose une question à laquelle l'utilisateur doit répondre avec le caractère "O" pour oui ou le caractère "N" pour non. Il est clair que l'utilisateur peut se tromper (il tape par exemple un "o" minuscule).

C'est pourquoi il est nécessaire de mettre en place ce qu'on appelle un « contrôle de saisie » pour vérifier que les données entrées au clavier correspondent bien à celles attendues par l'algorithme.

A priori, on pourrait tester la saisie avec un **IF** :

```
static char lire(){
        Scanner lectureClavier = new Scanner(System.in);
        System.out.println("Voulez vous un café ? (O/N) ");
        return lectureClavier.next().charAt(0);
}
public static void main(String[] args) {
    char choix = lire();
    if (choix!='O' && choix!='N'){
        System.out.println("Réponse fausse, répondez par O ou N")
        choix = lire();
    }
}
```

Ceci est parfait si l'utilisateur ne se trompe qu'une seule fois. Si l'on veut également rattraper une deuxième erreur éventuelle, il faudrait rajouter un autre **IF**. Et ainsi de suite, on peut rajouter des centaines de **IF** et écrire un algorithme interminable. Il y aura toujours moyen que l'utilisateur face échouer la saisie. C'est donc une impasse.

Dans notre cas, nous avons à faire avec une boucle de la 1ère famille, puisque nous ne pouvons pas savoir à l'avance combien de fois l'utilisateur se trompera lors de la saisie. Nous allons donc utiliser la boucles **while** présentées dans ce chapitre.

2. La boucle while

Sémantiquement, la boucle **while** signifie « Fais quelque chose tant qu'une condition est remplie ». Voila son code en Java:

```
while (condition){
      bloc d'instructions
}
```

Nous allons illustrer cette boucle avec notre problème de contrôle de saisie. A priori, cette structure permet de mettre en œuvre le contrôle de saisie par la répétition « tant que la réponse est erronée, recommencez la saisie ». On obtient donc:

```
static char lire(){
    Scanner lectureClavier = new Scanner(System.in);
    System.out.println("Voulez vous un café ? (O/N) ");
    return lectureClavier.next().charAt(0);
}

public static void main(String[] args) {
    char choix = lire();
    while (choix!='O' && choix!='N'){
        System.out.println("Réponse fausse, répondez par O ou N")
        choix = lire();
    }
}
```

3. La boucle for

Il ne nous reste plus qu'à voir la boucle qui nous permet d'effectuer un traitement lorsqu'on connaît d'avance le nombre d'itérations. On utilise alors une variable d'itération qui nous permettrat de contrôler le nombre de répétitions de la boucle. Par convention, si cette variable ne désigne rien de précis on la nomme **i** (pour itérateur). Voilà un exemple de syntaxe d'une boucle **while** où l'on maitrise le nombre d'itérations (10 itérations dans cet exemple):

```
int i = 0;
while (i<10){
    bloc d'instruction
    i = i + 1;
}
```

On remarque dans cette syntaxe trois éléments important et nécessaires au bon fonctionnement d'une boucle (en gras dans l'exemple). Les voici dans l'ordre:

- L'initialisation de la condition
- La condition de boucle
- La mise à jour de la condition

Si l'un de ces éléments venaient à manquer, on pourrait obtenir des bugs bien embêtant comme une boucle infinie ! En voila un exemple:

```
int i = 0;
while (i<10){
    bloc d'instruction
}
```

La condition étant tout le temps vraie et jamais mise à jour, le programmme ne sort jamais de la boucle !

Les boucles où l'on connaît d'avance le nombre d'itérations étant fréquentes, il existe un raccourci d'écriture en Java: la boucle **for**. Pour plus de clarté la boucle **for** réunit sur la même ligne les trois éléments que nous venons de voir. Voilà le même exemple réalisé avec une boucle **for**:

```
for (int i = 0; i<10; i++){
    /* bloc d'instruction */
}
```

En plus de réunir sur la même ligne les trois éléments importants d'une boucle, le **for** a un autre avantage. On peut définir la variable de boucle (**l'itérateur i**) directement dans la boucle. Ceci permet d'écourter la durée de vie de la variable de boucle, afin de ne pas gaspillller de la mémoire inutilement.

Remarque: **i++** est un raccourci d'écriture pour i = i + 1. Il existe aussi un raccourci d'écriture pour i = i – 1, qui est **i--** (voir le document sur la syntaxe java de base disponible sur le site du cours)