Chapitre 3

Conditionnelle et boucles

3.1 Introduction

Les instructions que nous avons vu jusqu'ici modifient la mémoire ou l'écran. Dans ce chapitre, nous allons voir trois instructions qui ne modifient directement ni la mémoire ni l'écran mais qui déterminent quelles instructions du programme vont s'exécuter.

On les appelle des instructions de contrôle et elles ont la particularité de contenir d'autres instructions. On va avoir des instructions imbriquées dans d'autres instructions. Ces instructions de contrôle ont un rôle déterminant dans l'exécution des programmes et font que ceux-ci font plus que d'exécuter une suite d'instructions invariable.

3.2 Conditionnelle

Nous voudrions maintenant écrire un programme qui, étant donné un prix Hors Taxe (HT) donné par l'utilisateur, calcule et affiche le prix correspondant TTC. Il y a 2 taux possible de TVA : la TVA normale à 20.0% et le taux réduit à 5.5%. On va demander aussi à l'utilisateur la catégorie du taux qu'il faut appliquer.

données

- entrée : un prix HT de type `double` (`pHT`), un taux de type `int` (`t`) (0 pour normal et 1 pour réduit)
- sortie : un prix TTC de type `double` (`pTTC`)

algorithme

1. afficher un message demandant une somme HT à l'utilisateur.
2. recueillir la réponse dans `pHT`
3. afficher un message demandant le taux (0 ou 1).
4. recueillir la réponse dans `t`
5. 2 cas :
   (a) Cas 1 :le taux demandé est normal `pTTC= pHT+ (pHT*0.2)`
   (b) Cas 2 :le taux demandé est réduit `pTTC= pHT+ (pHT*0.055)`
6. afficher `pTTC`

Avec ce que nous connaissons déjà en Java, nous ne pouvons pas coder cet algorithme. La ligne 5 pose problème : elle dit qu'il faut dans un certain cas exécuter une tâche, et dans l'autre exécuter une autre tâche. Nous ne pouvons pas exprimer cela en Java pour l'instant, car nous ne savons qu'exécuter, les unes après les autres de façon inconditionnelle la suite d'instructions qui constitue notre programme.

Pour faire cela, il y a une instruction particulière en Java, comme dans tout autre langage de programmation : l'instruction conditionnelle, qui à la forme suivante :

```
if (condition) {
    instructions1
}else{
    instructions2
}
```

et s'exécute de la façon suivante : si la condition est vraie c'est la suite d'instructions `instructions1` qui est exécutée ; si la condition est fausse, c'est la suite d'instructions `instructions2` qui est exécutée.

La condition doit être une expression booléenne c'est à dire une expression dont la valeur est soit `true` soit `false`.

Voilà le programme Java utilisant une conditionnelle qui calcule le prix TTC :

```
public class PrixTTC {
    public static void main (String[] args) {

        double pHT,pTTC;
        int t;
        Terminal.ecrireString("Entrer le prix HT: ");
        pHT = Terminal.lireDouble();

        Terminal.ecrireString("Entrer taux (normal->0 reduit ->1) ");
        t = Terminal.lireInt();
        if (t==0){
           pTTC=pHT + (pHT*0.2);
        }else {
           pTTC=pHT + (pHT*0.055);
        }
        Terminal.ecrireStringln("La somme TTC: "+ pTTC );
    }
}
```

Ce programme est constitué d'une suite de 8 instructions. Il s'exécutera en exécutant séquentiellement chacune de ces instructions :

1. déclaration de `pHT` et `pTTC`
2. déclaration de `t`
3. affichage du message `"Entrer le prix HT:"`
4. la variable `pHT` reçoit la valeur entrée au clavier.
5. affichage du message `"Entrer taux (normal->0 reduit ->1)"`
6. la variable `t` reçoit la valeur entrée au clavier (0 ou 1)

7. Java reconnaît le mot clé `if` et fait donc les choses suivantes :
   (a) il calcule l'expression qui est entre les parenthèses `t==0`. Le résultat de `t==0` dépend de ce qu'a entré l'utilisateur. S'il a entré `0` le résultat sera `true`, sinon il sera `false`.
   (b) Si le résultat est `true`, les instructions entre les accolades sont exécutées. Ici, il n'y en a qu'une :`pTTC=pHT + (pHT*0.2);` qui a pour effet de donner a `pTTC` le prix TTC avec taux normal. Si le résultat est `false`, ce sont les instructions dans les accolades figurant après le `else` :`pTTC=pHT + (PHT*0.055);`
8. la valeur de `pTTC` est affichée. Cette dernière instruction **est toujours exécutée** : elle n'est pas à l'intérieur des accolades du `if` ou du `else`. La conditionnelle a servi à mettre une valeur différente dans la variable `pTTC`, mais il faut dans les deux cas afficher cette valeur.

3.2.1 `if` sans `else`

Lorsqu'on veut dire : si `condition` est vraie alors faire ceci, sinon ne rien faire, on peut omettre le `else` ainsi que les accolades qui suivent ce mot-clé.

3.2.2 tests à la suite

Lorsque le problème à résoudre nécessite de distinguer plus de 2 cas, on peut utiliser la forme suivante :

```
if (condition1){s1}
else if (condition2) {s2}
...
else if (conditionp) {sp}
else {sf}
```

On peut mettre autant de `else if` que l'on veut, mais 0 ou 1 `else` (et toujours à la fin). `else if` signifie sinon si. Il en découle que les conditions sont évaluées dans l'ordre. La première qui est vraie donne lieu à l'exécution de la séquence d'instructions qui est y associée. Les conditions qui suivent sont ignorées même si elle sont vraies. Elles ne sont même pas calculées. Si aucune condition n'est vraie, c'est le `else`, s'il y en a un, qui est exécuté. Si aucune condition n'est vraie et qu'il n'y a pas de `else`, l'instruction `if` n'exécute aucune instruction.

3.3 Itération

Ecrivons un programme qui affiche à l'écran un rectangle formé de 5 lignes de 4 étoiles. Ceci est facile : il suffit de faire 5 appels consécutifs à la méthode `Terminal.ecrireStringln`

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

Ceci est possible, mais ce n'est pas très élégant ! Nous avons des instructions qui nous permettent de répéter des tâches. Elles s'appellent les instructions d'*itération*.

3.3.1 La boucle for

La boucle for permet de répéter une tâche un nombre de fois connus à l'avance. Ceci est suffisant pour l'affichage de notre rectangle :

```
public class Rectangle {
    public static void main (String[] args) {
        for (int i=0;i<5;i=i+1){
          Terminal.ecrireStringln("****");
        }
    }
}
```

Répète `5` fois, pour `i` allant de 0 à 4, les instructions qui sont dans les accolades, c'est à dire dans notre cas : afficher une ligne de 4 `*`.
Cela revient bien à dire : répète 5 fois "afficher une ligne de 4 étoiles".

Détaillons cela : La boucle `for` fonctionne avec un compteur du nombre de répétitions qui est géré dans les 3 expressions entre les parenthèses.

- La première :`int i=0` donne un nom à ce compteur (`i`) et lui donne une valeur initiale `0`. Ce compteur ne sera connu qu'à l'intérieur de la boucle for. (il a été declaré dans la boucle for `int i`)
- La troisième : `i=i+1` dit que les valeurs successives de `i` seront 0 puis 0+1 puis 1+1, puis 2+1 etc.
- La seconde (`i<5`) dit quand s'arrête l'énumération des valeurs de `i` : la première fois que `i<5` est faux.

Grâce à ces 3 informations nous savons que `i` va prendre les valeurs successives 0, 1, 2, 3, 4. Pour chacune de ces valeurs successives, on répétera les instructions dans les accolades.

Jouons un peu

Pour être sur d'avoir compris, examinons les programmes suivants :

```
public class Rectangle {
    public static void main (String[] args) {
        for (int i=0;i<5;i=i+2){
          Terminal.ecrireStringln("****");
        }
    }
}
```

`i` va prendre les valeurs successives 0, 2, 4. Il y aura donc 3 répétitions. Ce programme affiche 3 lignes de 4 étoiles.

```
public class Rectangle {
    public static void main (String[] args) {
        for (int i=1;i<5;i=i+2){
          Terminal.ecrireStringln("****");
        }
    }
```

```
}
```

`i` va prendre les valeurs successives 1, 3. Il y aura donc 2 répétitions. Ce programme affiche 2 lignes de 4 étoiles.

```
public class Rectangle {
    public static void main (String[] args) {
        for (int i=1;i<=5;i=i+2){
          Terminal.ecrireStringln("****");
        }
    }
}
```

`i` va prendre les valeurs successives 1, 3, 5. Il y aura donc 3 répétitions. Ce programme affiche 3 lignes de 4 étoiles.

```
public class Rectangle {
    public static void main (String[] args) {
        for (int i=1;i==5;i=i+2){
          Terminal.ecrireStringln("****");
        }
    }
}
```

`i` ne prendra aucune valeur car sa première valeur (`1`) n'est pas égale à `5`. Les instructions entre accolades ne sont jamais exécutées. Ce programme n'affiche rien.

Les boucles sont nécessaires

Dans notre premier exemple, nous avons utilisé une boucle pour abréger notre code. Pour certains problèmes, l'utilisation des boucles est absolument nécessaire. Essayons par exemple d'écrire un programme qui affiche un rectangle d'étoiles dont la longueur est donnée par l'utilisateur (la largeur restera 4).

Ceci ne peut se faire sans boucle car le nombre de fois où il faut appeler l'instruction d'affichage dépend de la valeur donnée par l'utilisateur.

En revanche, cela s'écrit très bien avec une boucle `for` :

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

La variable `ligne` est déclarée dans la première instruction. Elle a une valeur à l'issue de la troisième instruction. On peut donc tout à fait consulter sa valeur dans la quatrième instruction (le `for`). Si l'utilisateur entre 5 au clavier, il y aura 5 étapes et notre programme affichera 5 lignes de 4 étoiles. Si l'utilisateur entre 8 au clavier, il y aura 8 étapes et notre programme affichera 8 lignes de 4 étoiles.

le compteur d'étapes peut intervenir dans la boucle

Le compteur d'étapes est connu dans la boucle. On peut tout à fait consulter son contenu dans la boucle.

```
public class Rectangle3 {
  public static void main (String[] args) {
    int lignes;
    Terminal.ecrireString("combien␣de␣lignes␣d'etoiles␣?:");
    lignes=Terminal.lireInt();
    for (int i=1;i<=lignes;i=i+1){
      Terminal.ecrireInt(i);
      Terminal.ecrireStringln("****");
    }
  }
}
```

Ce programme affiche les lignes d'étoiles précédées du numéro de ligne.

C'est pour cela que l'on a parfois besoin que la valeur du compteur d'étapes ne soit pas son numéro dans l'ordre des étapes. Par exemple, voici un programme qui affiche les n premiers entiers pairs avec n demandé à l'utilisateur.

```
public class Combien {
  public static void main (String[] args) {
    int n;
    Terminal.ecrireString("combien␣d'entiers␣pairs␣?:");
    n=Terminal.lireInt();
    for (int i=0;i<n*2;i=i+2){
      Terminal.ecrireInt(i);
      Terminal.ecrireString("␣,␣");
    }
  }
}
```

Et voici un programme qui affiche les 10 premiers entiers en partant du plus grand vers le plus petit :

```
public class PremEntiers {
    public static void main (String[] args) {
        for (int i=10;i>0;i=i-1){
            Terminal.ecrireString(i + ",");
        }
    }
}
```

Pour en savoir plus

La syntaxe de la boucle `for` en Java est beaucoup plus permissive que ce qui à été exposé ici. Nous nous sommes contentés de décrire son utilisation usuelle. Nous reviendrons plus en détail sur les boucles dans un chapitre ultérieur.

3.3.2 la boucle while

Certaines fois, la boucle `for` ne suffit pas. Ceci arrive lorsqu'au moment où on écrit la boucle, on ne peut pas déterminer le nombre d'étapes.

Reprenons notre exemple de calcul de prix TTC. Dans cet exemple, nous demandions à l'utilisateur d'entrer 0 pour que l'on calcule avec le taux normal et 1 pour le taux réduit.
Que se passe-t-il si l'utilisateur entre 4 par exemple ? 4 est diffèrent de 0 donc le `else` sera exécuté. Autrement dit : toute autre réponse que 0 est interprétée comme 1.
Ceci n'est pas très satisfaisant. Nous voulons maintenant améliorer notre programme pour qu'il redemande à l'utilisateur une réponse, tant que celle-ci n'est pas 0 ou 1.
Nous sentons bien qu'il faut une boucle, mais la boucle `for` est inadaptée : on ne peut dire a priori combien il y aura d'étapes, cela dépend de la vivacité de l'utilisateur ! Pour cela, nous avons la boucle `while` qui a la forme suivante :

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

Cette boucle signifie : tant que la condition est vraie, exécuter les instructions entre les accolades (le *corps de la boucle*). Grâce à cette boucle, on peut répéter une tâche tant qu'un **événement dans le corps de la boucle** ne s'est pas produit. C'est exactement ce qu'il nous faut : nous devons répéter la demande du taux, tant que l'utilisateur n'a pas fourni une réponse correcte.

écriture de la boucle

La condition de notre boucle devra être une expression booléenne Java qui exprime le fait que la réponse de l'utilisateur est correcte. Pour faire cela, il suffit d'ajouter une variable à notre programme, que nous appellerons `testReponse`. Notre programme doit s'arranger pour qu'elle ait la valeur `true` dès que la dernière saisie de l'utilisateur est correcte, c'est à dire si `t` vaut 0 ou 1, et `false` sinon.

Notre boucle pourra s'écrire :

```
while (testReponse==false){
  Terminal.ecrireStringln("Entrer taux (normal->0 reduit ->1) ");
  t = Terminal.lireInt();
  if (t==0 || t==1){
    testReponse=true;
  }
  else {
    testReponse=false;
  }
}
```

Il faudra, bien entendu, avoir déclaré `testReponse` avant la boucle.

comportement de la boucle

La condition de la boucle est testée **avant** chaque exécution du corps de la boucle. On commence donc par tester la condition ; si elle est vraie le corps est exécuté une fois, puis on teste à nouveau la

condition et ainsi de suite. L'exécution de la boucle se termine la première fois que la condition est fausse.

initialisation de la boucle

Puisque `testReponse==false` est la première chose exécutée lorsque la boucle est exécutée, il faut donc que `testReponse` ait une valeur **avant** l'entrée dans la boucle. C'est ce qu'on appelle *l'initialisation de la boucle*. ici, puisque l'on veut entrer au moins une fois dans la boucle, il faut initialiser `testReponse` avec `false`.

état de sortie de la boucle

Puisqu'on sort d'une boucle `while` la première fois que la condition est fausse, nous sommes sûrs que dans notre exemple, en sortie de boucle, nous avons dans `t` une réponse correcte : 0 ou 1. Les instructions qui suivent la boucle sont donc le calcul du prix TTC selon des 2 taux possibles.

Voici le code Java :

```
public class PrixTTC2 {
    public static void main (String[] args) {

        double pHT,pTTC;
        int t=0;
        boolean testReponse=false;
        Terminal.ecrireString("Entrer␣le␣prix␣HT:␣");
        pHT = Terminal.lireDouble();
        while (testReponse==false){
          Terminal.ecrireString("Entrer␣taux␣(normal->0␣reduit␣->1)␣");
          t = Terminal.lireInt();
          if (t==0 || t==1){
            testReponse=true;
          }
          else {
            testReponse=false;
          }
        }
        if (t==0){
          pTTC=pHT + (pHT*0.2);
        }
        else {
          pTTC=pHT + (pHT*0.055);
        }
        Terminal.ecrireStringln("La␣somme␣TTC:␣"+ pTTC );
    }
}
```

Terminaison des boucles `while`

On peut écrire avec les boucles while des programmes qui ne s'arrêtent jamais. C'est presque le cas de notre exemple : si l'utilisateur n'entre jamais une bonne réponse, la boucle s'exécutera à l'infini.

Dans ce cours, les seuls programmes qui ne terminent pas toujours et que vous aurez le droit d'écrire

seront de cette catégorie : ceux qui contrôlent les saisies utilisateurs.

Pour qu'une boucle `while` termine, il faut s'assurer que le corps de la boucle contient des instructions qui mettent à jour la condition de boucle. Autrement dit, le corps de la boucle est toujours constitué de deux morceaux :

— le morceau décrivant la tâche à effectuer à chaque étape
— le morceau décrivant la mise à jour de la condition de sortie

Pour qu'une boucle termine **toujours**, il faut que **chaque** mise à jour de la condition de sortie, nous rapproche du moment où la condition sera fausse.

C'est bien le cas dans l'exemple suivant :

```
public class b1 {
    public static void main (String[] args) {

        int i=1;

        while (i!=10){
          Terminal.ecrireString(i + ",");
          i=i+1;
        }
    }
}
```

Au début de la première exécution du corps de boucle `i` vaut 1, Au début de la deuxième exécution du corps de boucle `i` vaut 2, ... A chaque fois, on progresse vers le moment où `i` vaut 10. Ce cas est atteint au bout de 10 étapes.

Attention, il ne suffit de se rapprocher du moment où la condition est fausse, il faut l'atteindre un jour, ne pas la rater.

Le programme suivant, par exemple, ne termine jamais :

```
public class b3 {
    public static void main (String[] args) {

        int i=1;

        while (i!=10){
          Terminal.ecrireString(i + ",");
          i=i+2;
        }
    }
}
```

Car `i` progresse de 1 à 3, puis à 5, à 7, à 9, à 11 et 10 n'est jamais atteint.

Quelle boucle choisir ?

La boucle while et la boucle for sont d'une certaine façon équivalentes : on peut toujours transformer une boucle for en un while équivalent et réciproquement. Mais dans certain cas, c'est l'une qui est plus facile à écrire et dans d'autres, c'est l'autre.

Nous vous proposons le critère suivant pour déterminer quelle boucle choisir : si au moment où la boucle commence, on connait le nombre de tours à effectuer, choisir un for. Dans les autres cas, choisir un while.