UNIVERSITÉ Grenoble Alpes

Initiation à l'informatique et à l'algorithmique (LICENCE MIASHS 1)

6. Les itérations

Les traitements répétitifs sont très fréquents en informatique. Plutôt que de répéter les mêmes instructions plusieurs fois, on va recourir à des instructions spéciales qui permettent de revenir en arrière (ou boucler).

Par exemple, si on veut afficher tous les nombres premiers entre 1 et 1000, on ne va pas les afficher un par un ! Ou encore si on veut lire une valeur entre 1 et 10 et que l'utilisateur se trompe et saisisse une valeur trop grande, il faut lui redemander une nouvelle valeur. On pourrait utiliser deux instructions de lecture et un test pour effectuer la deuxième instruction en cas d'erreur. Mais s'il se trompe de nouveau, notre programme ne fonctionnera pas, à moins de prévoir ce cas et d'écrire trois instructions de lecture et deux tests.... Il est bien plus simple d'utiliser une boucle.

L'instruction while

La première instruction de boucle que l'on va étudier suit ce schéma :

while condition {

instructions

}

while

Tant que la condition est vraie, les instructions entre les deux accolades sont exécutées. Plus précisément, l'ordinateur teste la condition (rappelez-vous, une condition n'est pas une instruction, elle doit être soit vraie, soit fausse). Si elle est fausse, il ignore cette instruction et continue le programme après l'accolade fermante. Si elle est vraie, il exécute les instructions entre les accolades. Puis, il remonte pour tester de nouveau la condition. Si elle est toujours vraie, il recommence à exécuter les mêmes instructions entre accolades, sinon il continue après l'accolade fermante, etc.

Revenons sur l'exemple de la lecture d'un nombre entre 1 et 10. On peut l'écrire simplement comme ceci :

```
int nb  ;   // un nombre entre 1 et 10
Scanner s = new Scanner(System.in);
System.out.println("Veuillez entre un nombre entre 1 et 10");
nb = s.nextInt();
while ((nb<0) || (nb >10)) {
        System.out.println("Le nombre doit être entre 1 et 10. Recommencez.");
        nb = s.nextInt();
}
```

Tout comme pour l'instruction if, on peut se dispenser de mettre des accolades s'il n'y a qu'une seule instruction.

 Il faut que les instructions dans la boucle modifient d'une manière ou d'une autre les variables utilisées dans la condition. Sinon, la condition ne sera jamais fausse et la boucle se poursuivra indéfiniment ! Par exemple, ce programme ne s'arrête jamais :

```
// Programme qui boucle indéfiniment
x=1  ;
while (x<10)
        System.out.print(x)  ;
```

Écrivons maintenant un programme qui lit des valeurs au clavier et qui en fait la somme. On suppose qu'il y a 10 valeurs à entrer.

Il faut que notre boucle contienne l'instruction de lecture ainsi qu'une instruction pour totaliser les valeurs et une instruction pour compter jusque 10.

Pour utiliser un compteur, il faut:

- une variable initialisée (à 0 ou 1 généralement);
- une condition d'arrêt qui peut dépendre de cette variable ;
- une incrémentation de cette variable dans la boucle, c'est-à-dire une instruction de la forme `cpt = cpt + 1`. Vous trouverez parfois cette instruction sous la forme `cpt++` ou encore `cpt+=1`.

Voici une boucle qui compte jusque 100 :

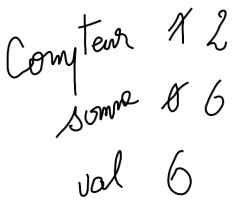
```
int compteur = 0;
while (compteur <= 100) {
        System.out.println(compteur);
        compteur = compteur + 1;
}
```

Attention à ne pas oublier d'augmenter le compteur de 1, sinon, comme précédemment, le programme ne terminera jamais !

Pour calculer une somme, il nous faut une autre variable, initialisée à 0 et une instruction pour lui ajouter une valeur à chaque passage dans la boucle :
`somme = somme + valeur` (ou encore `somme+=valeur`) On obtient donc :

```
int compteur = 1  ;
int somme = 0  ;
int val  ;
System.out.println("Veuillez entrer 10 valeurs");
while (compteur <= 10) {
        System.out.println("Veuillez entrer la valeur suivante");
        val=s.nextInt();
        somme = somme + val;
        compteur = compteur + 1;
}
System.out.println("La somme de ces valeurs est "+somme);
```

Prenez le temps de simuler l'exécution de ces instructions, l'une après l'autre, en inscrivant sur une feuille de papier les valeurs successives des trois variables comme ci-contre.



Attention à ne pas oublier d'initialiser les variables. Par exemple, si vous oubliez d'initialiser la variable somme à 0, vous obtiendrez ce message :

```
variable somme might not have been initialized
```

L'instruction do-while

Il existe une autre structure de boucle, dans laquelle on passe au moins une fois puisque la condition est à la fin et non au début. C'est l'instruction do-while. Le schéma est le suivant :

do { instructions } while (condition)

Examinons les différentes entre le while et le do-while. Avec le while, on peut ne pas passer du tout dans la boucle si la condition est fausse dès le début. Avec le do-while, on y passe au moins une fois. L'exemple précédent sur la vérification de la validité d'un entier saisi au clavier, peut se réécrire comme suit :

```
int nb  ;   // un nombre entre 1 et 10
Scanner s = new Scanner(System.in);
do {
        System.out.println("Veuillez entre un nombre entre 1 et 10");
        nb = s.nextInt();
} while ((nb<0) || (nb >10));
...
```

Prenons un autre exemple. On veut écrire un programme qui simule un lancer de dé. Pour cela, on va utiliser une fonction existante en Java qui produit un nombre aléatoire : `Math.random()`. Voici le programme :

```
import java.util.Scanner;
public class LancerDe {
   public static void main(String[] args) {
        int nbAleatoire;
        Scanner s=new Scanner(System.in);
        do {
           nbAleatoire=1+(int)(Math.random()*6);
           System.out.println("Lancer du dé... c'est un "+nbAleatoire);
           System.out.println("Un autre lancer ? (o/n)");

        } while (s.nextLine().equals("o"));
   }
}
```

`Math.random()` retourne un réel dans l'intervalle [0,1[. Pour avoir un entier entre 1 et 6, il faut simplement multiplier ce réel par 6, prendre sa partie entière avec un cast et lui ajouter 1.

La boucle `do-while` s'arrête quand l'utilisateur ne répond pas o (oui) quand l'ordinateur lui demande s'il veut continuer.

Changeons un peu la nature de notre programme. On veut maintenant lui faire lancer 10.000 fois un dé et compter le nombre de six, pour vérifier si le générateur de nombres aléatoires fonctionne bien. Normalement, on devrait avoir une valeur pas trop éloignée de 10000/6=1667.

constante

Pour améliorer la lisibilité du programme, on va définir cette valeur 10.000 une fois pour toutes au début du programme, ce qui permettra aussi de la changer plus facilement. On va donc déclarer une *constante*. Une constante, comme son nom l'indique, ne peut pas changer de valeur. Elle est définie avec le mot-clé *final static*. Comme une constante doit être connue de toutes les parties du programme, elle sera même définie avant le *main*.

Si vous essayez de changer une constante dans un programme, vous aurez ce message :

```
cannot assign a value to final variable
```

Voici notre programme :

```
import java.util.Scanner;
public class LancerDe10000Fois {
   final static int nbLancers=10000;
   public static void main(String[] args) {
         int nbAleatoire;  // représente le nombre aléatoire
         int cpt=0;   // représente le nombre de lancers
         int nb6=0;  // représente le nombre de six déjà lancés
         Scanner s=new Scanner(System.in);
         do {
            nbAleatoire=1+(int)(Math.random()*6);   // on tire une valeur entre 1 et 6
            System.out.println("Lancer du dé... c'est un "+nbAleatoire);
            if (nbAleatoire == 6)  // si c'est un six...
                  nb6++;            // on compte un six de plus
            cpt++;
         } while (cpt < nbLancers);
         System.out.println("Sur les "+nbLancers+" lancers, il y a eu "+nb6+" six.");
   }
}
```

Remarquez l'incrémentation de la variable nb6 qui compte le nombre de six (nb6++) seulement si on a un six et l'incrémentation systématique de la variable qui compte les lancers. Voici un exemple d'exécution du programme :

```
Lancer du dé... c'est un 6
...
Lancer du dé... c'est un 2
Lancer du dé... c'est un 3
Sur les 10000 lancers, il y a eu 1664 six.
```

7. Le langage machine

L'ordinateur est composé d'un microprocesseur qui ne comprend pas le Java. Ce microprocesseur comprend en revanche des instructions généralement très simples comme additionner deux nombres, copier une valeur dans une case mémoire, etc. On pourrait écrire dans le langage du microprocesseur, le *langage machine*, mais ce serait très fastidieux. C'est pour cela que l'on a créé des langages de plus haut niveau, plus compréhensibles par nous les humains comme C, Pascal, Perl, Python, et... Java. L'inconvénient est qu'il a fallu créer des programmes qui *traduisent*, par exemple, Java en langage machine.

Regardons à quoi pourrait ressembler un programme en langage machine. C'est un exemple fictif pour bien comprendre. Voici un programme Java :

```
public class MonProg {
        public static void main(String [] args) {
                int a,b,c;
                a=1;
                b=2;
                if (a==b)
                        c=a*(2+b);
                else
                        c=0;
                c=c*2
        }
}
```

et sa traduction dans un langage machine imaginaire :

```
 86 chargerValeur 1
 87 stockerEnMemoire 32
 88 chargerValeur 2
 89 stockerEnMemoire 33
 90 chargerMemoire 32
 91 CalculerDifferenceAvecMemoire 33
 92 SiDifferentDeZeroAllerEn 98
 93 ChargerValeur 2
 94 CalculerSommeAvecMemoire 33
 95 CalculerProduitAvecMemoire 32
 96 stockerEnMemoire 34
 97 Aller en 100
 98 chargerValeur 0
 99 stockerMemoire 34
100 chargerValeur 2
101 CalculerProduitAvecMemoire  34
102 stockerEnMemoire 34
```

Il faut envisager la mémoire comme une immense suite de cases, chacune étant repérée par un numéro. Si la mémoire de votre ordinateur fait par exemple 4Go, cela signifie qu'il y a l'équivalent de 4 milliards d'octets. Cette mémoire stocke tout : le programme mais aussi les données (les variables a,b et c de notre programme). Dans l'exemple ci-dessus, on voit que le programme est stocké dans les cases 86 à 102. Les données seront stockées dans les cases 32 à 34.

`chargerValeur` est une instruction qui place une valeur dans une zone intermédiaire. `StockerEnMemoire` est une opération inverse qui place ce qui est dans cette zone intermédiaire dans une case particulière de la mémoire. Ainsi, les deux premières instructions codent le "a=1" du programme Java, la variable "a" étant associée à la

case mémoire 32. Les deux suivants codent le b=2 !

L'instruction de test if (a==b) est traduite en une instruction qui charge la valeur de a, une instruction qui calcule la différence avec le contenu de la mémoire 33 (donc b) et une instruction qui, si cette différence est nulle, poursuit le programme plus loin (à la case 98 dans notre cas).

Vous pouvez suivre pas à pas le déroulement du programme jusqu'à la fin. Vous voyez combien c'est fastidieux de programmer dans ce langage machine puisque il faut gérer les cases de la mémoire, prévoir où la suite du programme va se dérouler, etc.

Il existe plusieurs méthodes pour passer d'un langage de haut niveau à un langage machine. Pour un langage comme le C, le programme est traduit directement en un fichier en langage machine. C'est une opération de *compilation*. Le fichier est dit *exécutable*, c'est par le cas des fichiers .exe sous Windows : leur contenu est donc complètement opaque pour nous humains. Une fois que le fichier exécutable est créé, on n'a plus besoin du compilateur pour pouvoir exécuter le programme en C.

Pour un langage comme Java, c'est un peu différent. Le programme est traduit dans un langage particulier, le *byte code*, pas encore exécutable, mais plus proche des instructions de la machine. C'est le fichier `.class`, illisible aussi pour nous humains. Il y a ensuite un autre programme qui est capable de lire cette traduction pour l'exécuter. Ce programme s'appelle la machine virtuelle java. C'est pour cela que vous devez toujours posséder cette machine virtuelle pour exécuter vos programmes.