Using For-Loops for Repetition

A **loop** is a program structure for executing a block of code repeatedly. The code for a loop usually comes with a specification of when to stop the repetition. Java has four loop structures: the for-loop, the while-loop, the do-while loop, and the for-each loop.

Suppose we want to increase the number of repetitions from two to a much larger number, for example, ten. We can accomplish this by adding eight more lines of `oneInteraction()`:

```
1   public static void main( String[] args )
2   {
3     oneInteraction();
4     oneInteraction();
5     oneInteraction();
6     oneInteraction();
7     oneInteraction();
8     oneInteraction();
9     oneInteraction();
10    oneInteraction();
11    oneInteraction();
12    oneInteraction();
13  }
```

Now what should we do if we wanted to increase the number of repetitions to 20? Should we add ten more lines of the same `oneInteraction()`? Using a loop, it is possible to state the 20-time repetitions in just a few lines.

```
1  public static void main( String[] args )
2  {
3    int i;
4    for ( i = 1; i <= 10; i = i + 1 )
5    {
6      oneInteraction();
7    }
8  }
```

Line 4 of the code,

```
for ( i = 1; i <= 10; i = i + 1 )
```

is the for-loop. It means

"repeat the following as long as `i <= 10` by first assigning the value of 1 to `i` and then adding 1 to `i` each time."

The "following" refers to the block of code between Lines 5–7. We call this block the **loop-body**.

The actions that take place in the above for-loop are as follows:

- The value of `1` is stored in `i`.
- As long as the value of `i` is less than or equal to `10`,
  - execute `oneInteraction` and
  - increase the value of `i` by 1.

The use of a for-loop in stating the repetition makes it easy to change the number of repetitions. Furthermore, if the name of the method changes, we only have to replace just one call, which appears in the body of the loop.

The variable `i` that refers to the "round" in the repetition can be used in the body of the loop. For example, before calling `oneInteraction` we can announce the round:

```
1  public static void main( String[] args )
2  {
3    int i;
4    for ( i = 1; i <= 10; i = i + 1 )
5    {
6      System.out.println( "This is round " + i + "." );
7      oneInteraction();
8    }
9  }
```

The code with the round announcement appears next:

```
1  import java.util.Scanner;
2  public class BMIRepetitive
3  {
4    public static final double BMI_SCALE = 703.0;
5    public static final int FEET_TO_INCHES = 12;
6
7    public static double bmiFormula( double weight, double height )
8    {
9      return BMI_SCALE * weight / (height * height);
10   }
11
12   public static void oneInteraction()
13   {
14     Scanner keyboard = new Scanner( System.in );
15     System.out.print( "Enter weight: " );
16     double weight = keyboard.nextDouble();
17     System.out.print( "Enter height in feet and inches: " );
18     double feet = keyboard.nextDouble();
19     double inches = keyboard.nextDouble();
20     double height = FEET_TO_INCHES * feet + inches;
21     double bmi = bmiFormula( weight, height );
22     System.out.println( "Your BMI is " + bmi + "." );
23   }
24   public static void main( String[] args )
25   {
26     int i;
27     for ( i = 1; i <= 10; i = i + 1 )
28     {
29       System.out.println( "This is round " + i + "." );
30       oneInteraction();
31     }
32   }
33 }
```

Listing 7.1 A program that repeatedly computes BMI using a for-loop. The program announces each round

The execution of the code, with some input from the user, produces the following:

```
1  This is round 1.
2  Enter weight: 150
3  Enter height in feet and inches: 5 6
4  Your BMI is 24.207988980716255.
5  This is round 2.
6  Enter weight: 150 5 7
7  Enter height in feet and inches: Your BMI is 23.490755179327245.
8  This is round 3.
9  Enter weight: 160 5 7
10 Enter height in feet and inches: Your BMI is 25.056805524615726.
11 This is round 4.
12 ...
```

As we have seen in the above, the header part of a for-loop has three components with a semicolon in between:

- **initialization,**
- **continuation (termination) condition,**
- **update**

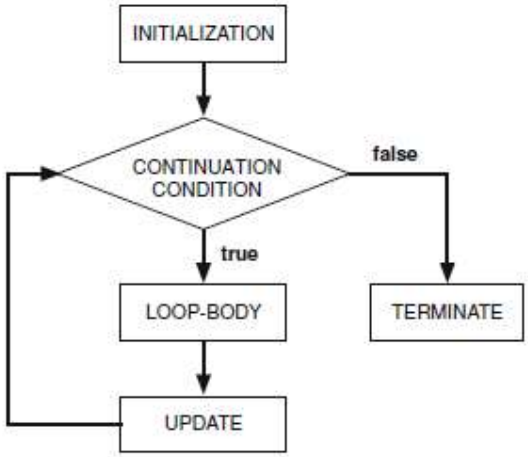

Fig. 7.1 A generic flow chart of for-loops

In other words, the header part of a for-loop takes the form of:

```
for ( INITIALIZATION; CONTINUATION CONDITION; UPDATE ) { ... }
```

The roles of these components are as follows:

- The initialization of a for-loop is a statement that is executed prior to entering the repetition.
- The continuation condition is one that must hold for the loop-body to execute. Before executing the loop-body, this condition is tested. If the condition does not hold, the loop is terminated immediately.
- The update is a statement that is executed after each execution of the loop-body.[1]

The roles of the three components are summarized in Fig. 7.1. Most typically, in a for-loop, the initialization is an assignment to a variable (usually an integer), the termination condition is a comparison involving the variable, and the update is a modification to the variable. We call such a variable an **iteration variable**.

The next program contains a for-loop with an interaction that changes its value from 0 to 7.

```
1  public class ForExample
2  {
3    public static void main( String[] args )
4    {
5      int count;
6      for ( count = 0; count < 8; count ++ )
7      {
8        System.out.println( "The value of count is " + count );
9      }
10   }
11 }
```

Listing 7.2 An iteration over the sequence `0, ..., 7`

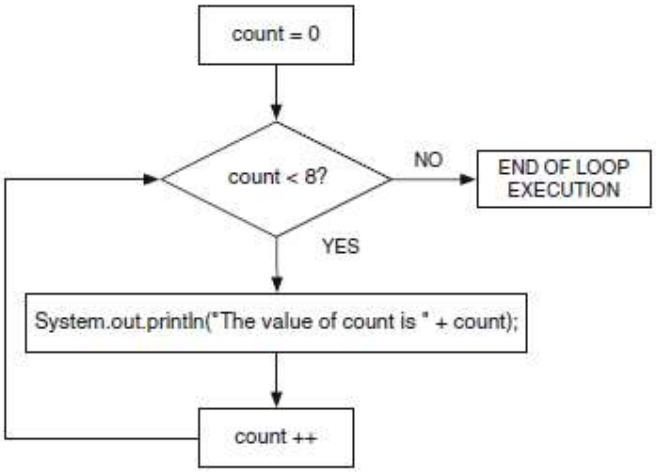

Fig. 7.2 The code execution diagram of `ForExample`

The iteration variable of the for-loop is `count`. Since the initial value of the iteration variable is 0, the termination condition is `count < 8`, and the update is `count ++`, the last value of `count` for which the loop-body is executed is 7. Figure 7.2 summarizes the above observation.

Running the code produces the following:

```
1 The value of count is 0
2 The value of count is 1
3 The value of count is 2
4 The value of count is 3
5 The value of count is 4
6 The value of count is 5
7 The value of count is 6
8 The value of count is 7
```

Using While-Loops

The Syntax of While-Loops

The while-loop is a loop that only requires a continuation condition. The structure of a while-loop is simple:

```
1 while ( CONDITION )
2 {
3   STATEMENTS;
4 }
```

The meaning of this while-loop is "as long as `CONDITION` has the value of `true`, execute `STATEMENTS`". The diagram in Fig. 11.1 shows how a while-loop works. The for-loop and while-loop can simulate each other. First,

```
1 while ( CONDITION )
2 {
3   STATEMENTS;
4 }
```

is equivalent to the for-loop without initialization and update, as shown next:

```
1 for ( ; CONDITION; )
2 {
3   STATEMENTS;
4 }
```

Second,

```
1 for ( INITIALIZATION ; CONDITION; UPDATE )
2 {
3   STATEMENTS;
4 }
```

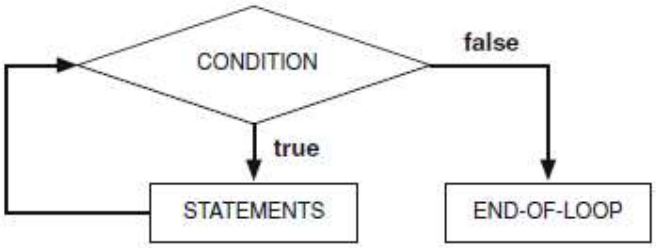

Fig. 11.1 A diagram that represents the while-loop

is equivalent to:

```
1 INITALIZATION;
2 while ( CONDITION )
3 {
4   STATEMENTS;
5   UPDATE;
6 }
```

```
1  import java.util.*;
2  public class UpToLimit
3  {
4    public static void main( String[] args )
5    {
6      Scanner keyboard = new Scanner( System.in );
7      int input, total = 0, goal = 1000;
8      while ( total <= goal )
9      {
10       System.out.print( "Enter input: " );
11       input = keyboard.nextInt();
12       total += input;
13       System.out.printf( "Input=%d, Total=%d%n", input, total );
14     }
15     System.out.printf( "The total has exceeded %d.%n", goal );
16   }
17 }
```

Listing 11.3 A program that receives input numbers until the total exceeds a preset bound

The loop can be quickly finished by entering a large number:

```
1  Enter input: 355555
2  Input=355555, Total=355555
3  The total has exceeded 1000.
```

Using Do-While Loops

The Syntax of Do-While Loops

The do-while loop is a variant of while-loops, where the execution of the loop-body precedes the termination condition evaluation.

The structure of a do-while loop is:

```
do
{
  STATEMENTS
} while ( CONDITION );
```

The semicolon that appears is necessary.
We can rewrite a do-while loop using a while-loop. The following do-while loop

```
do
{
  STATEMENTS;
} while ( CONDITION );
```

is equivalent to

```
STATEMENTS;
while ( CONDITION )
{
  STATEMENTS;
}
```

Since a while-loop is an indefinite loop, we can write the program so that it will run forever using `true` as the termination condition.

```
while ( true )
{
  STATEMENTS;
  if ( CONDITION )
  {
    break;
  }
}
```

Here is a simple program that uses a do-while loop.

Consider receiving a series of tokens from the user until the user enters "Godot", when the execution terminates (where, of course, the "Godot" comes from a play by Samuel Beckett[3] titled *Waiting for Godot*). We store the user input to a `String` variable named `input`, and build a do-while loop using the condition `!input.equals( "Godot" )`. In other words, the program will run until the user enters `"Godot"`.

If the code uses a while-loop, we need to assign some initial value other than the `"Godot"`. Otherwise, the loop terminates immediately without asking the user to enter an input. If the code uses a do-while loop, the initialization is unnecessary.

Here is the version that uses a while-loop:

```
1  import java.util.*;
2  public class Godot
3  {
4    public static void main( String[] args )
5    {
6      Scanner keyboard = new Scanner( System.in );
7      String input = "";
8      while ( !input.equals( "Godot" ) )
9      {
10       System.out.println( "This program is called \"Godot\"." );
11       System.out.print( "Enter input: " );
12       input = keyboard.nextLine();
13     }
14     System.out.println( "Terminating the program." );
15   }
16 }
```

Here is the version that uses a do-while loop:

```
1  import java.util.*;
2  public class DoWhileGodot
3  {
4    public static void main( String[] args )
5    {
6      Scanner keyboard = new Scanner( System.in );
7      String input;
8      do
9      {
10       System.out.println( "This program is called \"Godot\"." );
11       System.out.print( "Enter input: " );
12       input = keyboard.nextLine();
13     } while ( !input.equals( "Godot" ) );
14     System.out.println( "Terminating the program." );
15   }
16 }
```