CONTROL STRUCTURES (REPETITION)

while Looping (Repetition) Structure

In the previous section, you saw that sometimes it is necessary to repeat a set of statements several times. One way to do this is to type the set of statements in the program over and over. For example, if you want to repeat a set of statements 100 times, you type the set of statements 100 times in the program. However, this way of repeating a set of statements is impractical, if not impossible. Fortunately, there is a simpler approach. As noted earlier, Java has three repetition, or looping, structures that allow you to repeat a set of statements until certain conditions are met. This section discusses the first looping structure, a while loop.

The general form of the while statement is:

```
while (logical expression)
     statement
```

In Java, while is a reserved word. The logical expression is called a loop condition or simply a condition. The statement is called the body of the loop. Moreover, the statement can be either a simple or compound statement. Also, note that the parentheses around the logical expression are part of the syntax. Figure 1.1 shows the flow of execution of a while loop.

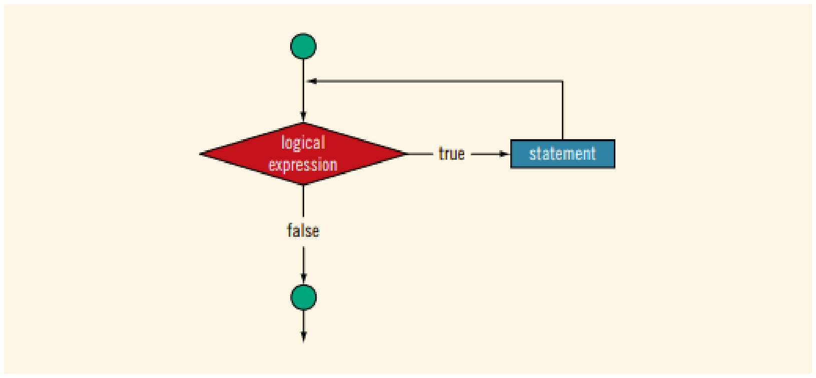


Figure 1.1 while Loop

The **logical expression** provides an entry condition. If it initially evaluates to true, the **statement** executes. The loop condition—the **logical expression**—is then reevaluated. If it again evaluates to true, the **statement** executes again. The statement (body of the loop) continues to execute until the **logical expression** is no longer true. A loop that continues to execute endlessly is called an **infinite loop**. To avoid an infinite loop, make sure that the loop's body contains one or more statements that ensure that the loop condition—the logical expression in the while statement—will eventually be false.

Example

Example:

```
//this loop prints whole numbers from 0 to 10
int num = 0;
while ( num <= 10) {
      System.out.println(num);
      num++;      //num = num + 1;
}
```

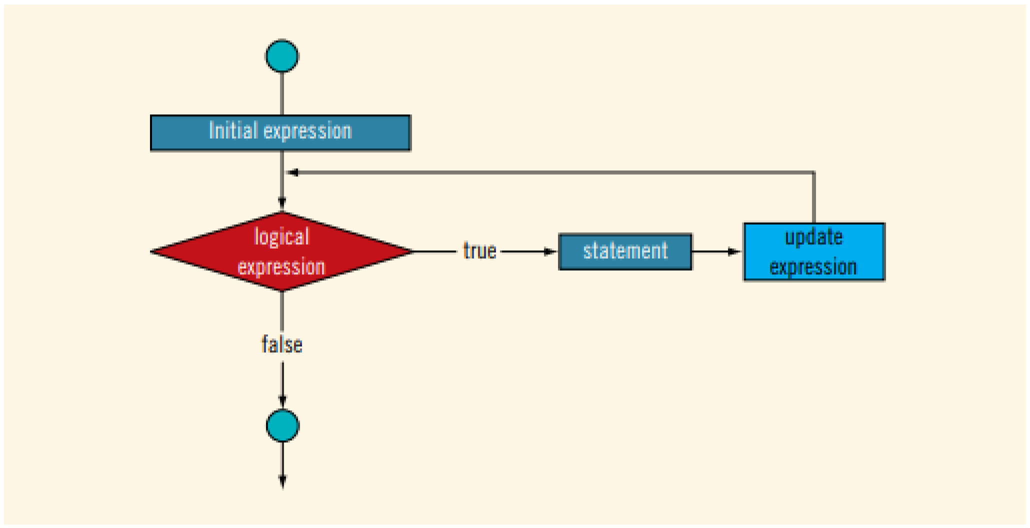
In the example, the variable, num is **a loop control variable**, a variable whose value determines whether loop execution continues. Within the loop body, the statement num++; is used to update the loop control variable until the loop condition evaluates to false.

for Looping (Repetition) Structure

The while loop discussed in the previous section is general enough to implement all forms of repetitions. As noted earlier, Java provides three looping structures. The previous section discussed while loops in detail. This section explains how to use Java's for loop. The general form of the for statement is:

```
for (initial expression; logical expression; update expression)
    statement
```

In Java, for is a reserved word. **The logical expression** is called **the loop condition**. The **initial expression, logical expression, and update expression (called for loop control expressions)** are enclosed within parentheses and control the body (statement) of the for statement. Note that the for loop control expressions are separated by semicolons, and that the body of a for loop can have either a simple or compound statement. Figure 1.2 shows the flow of execution of a for loop.



The for loop executes as follows:

1. The **initial expression** executes.

2. The **logical expression** is evaluated. If the loop condition evaluates to true:

a. Execute the body of the for loop.

b. Execute the update statement (the third expression in the parentheses).

3. Repeat Step 2 until the loop condition evaluates to false.

The **initial statement** usually initializes a variable (called the for loop control, or indexed, variable).

The following for loop prints the first 10 non-negative integers: (Assume that i is an int variable.)

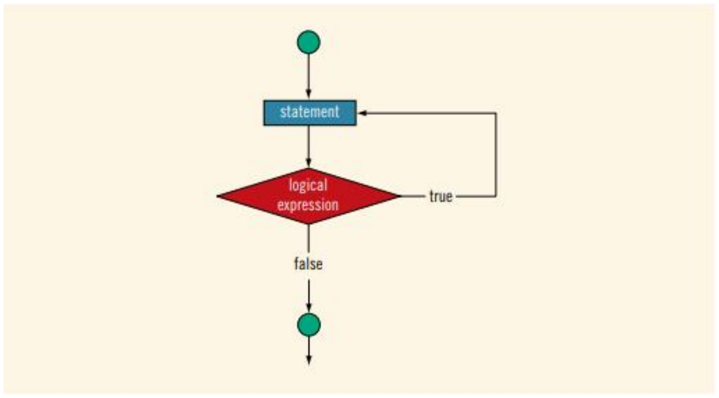
```
for (i = 0; i < 10; i++)
    System.out.print(i + " " );

System.out.println();
```

do...while Looping (Repetition) Structure

This section describes the third type of looping or repetition structure—a do...while loop. The general form of a do...while statement is:

```
do
    statement
while (logical expression);
```

In Java, do is a reserved word. As with the other repetition structures, the do...while statement can be either a simple or compound statement. If it is a compound statement, enclose it between braces. The logical expression is called the loop condition. Figure 1-3 shows the flow of execution of a do...while loop.



The **statement** executes first, and then the **logical expression** is evaluated. If the logical expression evaluates to true, the statement executes again. As long as **the logical expression** in a do...while statement is true, the **statement** executes. To avoid an infinite loop, you must, as before, make sure that the body of the loop contains a statement that ultimately makes the logical expression evaluate to false and assures that it exits properly.

Example

```
i = 0;
do
{
    System.out.print(i + " ");
    i = i + 5;
}
while (i <= 20);
```

The output of this code is:

0 5 10 15 20

After the value 20 is output,

the statement: i = i + 5;

changes the value of i to 25, so i <= 20 becomes false, which halts the loop.

break and continue Statements

The break and continue statements alter the flow of control in a program. As you have seen, the break statement, when executed in a switch structure, provides an immediate exit from the switch structure. Similarly, you can use the break statement in while, for, and do...while loops to immediately exit from these structures. The break statement is typically used for two purposes:

- To exit early from a loop
- To skip the remainder of the switch structure

After the break statement executes, the program continues to execute starting at the first statement after the structure.

Example of implementing the *break* statement:

```
int sum = 0, num;
while (true) {
      System.out.println("Type 0 to end program. A negative number is invalid.");
      System.out.print("Enter a number to add: ");
      num = console.nextInt();

      if (num == 0) {
            System.out.println("You entered 0. End of program. Thank you!");
            break;
      }

      sum = sum + num;
      System.out.println("The sum of entered numbers is: " + sum);
}
```

The example is an infinite loop, but with the use of a break statement, the loop can be terminated. In the example, when the user enters a 0, the expression in the if statement evaluates to true, then the appropriate message is printed, and the break statement terminates the loop.

The break statement terminates only the innermost loop or switch statement that contains the break statement. If a loop containing a break statement is inside another loop, the break statement ends only the innermost loop. If a break statement is within a switch statement that is inside a loop, the break statement terminates the switch statement but not the loop.

The **continue statement** is used in while, for, and do...while structures. When the continue statement is executed in a loop, it skips the remaining statements in the loop and proceeds with the next iteration of the loop. In a while or do...while structure, the logical expression is evaluated immediately after the continue statement. In a for structure, the update statement is executed after the continue statement, and then the logical expression executes.

If the previous program segment encounters a negative number, the while loop terminates. If you want to ignore the negative number and read the next number rather than terminate the loop, replace the break statement with the continue statement, as shown in the following example.

```
sum = 0;

while (console.hasNext())
{
    num = console.nextInt();

    if (num < 0) //if the number is negative, go to the
                 //next iteration
    {
        System.out.println("Negative number found in the data.");

        continue;
    }

    sum = sum + num;
}
```

Nested Control Structures

Nested control structures are control statements that are placed within another. Control structures in Java can be nested in many levels. When nesting control structures, you must make sure that their start and end braces are placed correctly within its control structure. The following are some examples of nested control structures with different levels:

Example #1: *for* loop nested in *if…else* statement

```
System.out.print("Enter a range of loop greater than 0: ");
int range = console.nextInt();
if (range > 0) {
      System.out.println("The output of the loop is: ");
      for (int i = 1; i <= range; i++) {
            System.out.println(i);
      } //end of for loop
} else {
      System.out.println("The entered number is invalid.");
} //end of if…else statement
```

Example #2: Nested *for* loop

```
int x, y;
System.out.println("Multiplication Table of 1-10:");
for (x = 1; x <= 10; x++) {
      for (y = 1; y <= 10; y++) {
            System.out.print(x * y + "\t");
      } //end of inner for loop
      System.out.println();
} //end of outer for loop
```