
Control Statements

In this exercise you will:

1. Review the `while` statement
2. Compare the `while` and `do-while` statements
3. Examine the relational and logical operators and how they are used in control statements
4. Investigate `if` and `if-else` statements

The while and do-while Statements

The while statement has the form

```
while(condition)
{
    body
}
```

and directs that the cycle of testing the *condition* and executing the *body* be continued as long as the test confirms that the *condition* is true. This iterative statement is called a looping structure and the while statement is called a while loop.

A slight variation of the while loop is the do-while loop. It has the form

```
do
{
    body
}while(condition);
```

and directs that the process of executing the *body* and then testing the *condition* be continued until the *condition* becomes false. Note that the while loop tests the *condition* before executing the *body*, and the do-while loop tests the *condition* after executing the *body*. Therefore, the *body* of the do-while loop is always executed at least once.

Experiment 3.4

Step 1. Predict the output of this program.

```
class J03E04
{
    public static void main(String[] args)
    {
        int numChips = 1;
        System.out.println("Open the bag and ...");
        while (numchips < 6)
        {
            System.out.println("  can't stop ...");
            numChips = numChips + 1;
        }
        System.out.println("  CAN'T STOP!");
    }
}
```

Step 2. Compile and execute J03E04.java to confirm your answer. If your prediction was incorrect, explain the discrepancy.

Experiment 3.5

Step 1. Predict the output of this program, which uses a while loop.

```
class J03E05A
{
    public static void main(String[] args)
    {
        int x = 5;
        while(x < 5)
        {
            System.out.println(x);
            x++; // the ++ increment operator adds 1 to variable x
        }
        System.out.println("That's all folks!");
    }
}
```

Step 2. Compile and execute program J03E05A.java. Record the results and explain any discrepancies with your prediction.

Step 3. Predict the output of this program, which is similar to the program in Step 1, but uses a do-while loop instead of a while loop.

```
class J03E05B
{
    public static void main(String[] args)
    {
        int x = 5;
        do
        {
            System.out.println(x);
            x++;
        } while(x < 5);
        System.out.println("That's all folks!");
    }
}
```

Step 4. Compile and execute program J03E05B.java. Record the results.

Step 5. Explain the differences in the output between the two programs in Steps 1 and 3.

Boolean Expressions

The *condition* part in a `while` or `do-while` statement is an example of a boolean expression, an expression whose value is either `true` or `false`. In Java, `true` and `false` are reserved words.

We have learned that these symbols, the *relational operators*, are used to construct boolean expressions that compare numerical values.

operator	meaning	Example: <code>int x = 5, y = 7;</code>	
		expression	value
<code>></code>	is greater than	<code>y > 6</code>	<code>true</code>
<code><</code>	is less than	<code>x + 3 < y</code>	<code>false</code>
<code>>=</code>	is greater than or equal to	<code>y - x >= 0</code>	<code>true</code>
<code><=</code>	is less than or equal to	<code>x + 10 <= y + 5</code>	<code>false</code>
<code>==</code>	is equal to	<code>x == y - 2</code>	<code>true</code>
<code>!=</code>	is not equal to	<code>x * y != 35</code>	<code>false</code>

We see that values of type `char` can also be compared using the relational operators. The expression `'a' < 'c'` is evaluated by comparing the underlying Unicode bit patterns interpreted as integers. Since `'a'` is stored as a 97 and `'c'` is stored as a 99, `'a' < 'c'` is true.

The logical operators AND (represented by `&&`) and OR (represented by 2 vertical lines, `||`) can be used to construct more complicated boolean expressions. For example, the *body* of this `while` loop will be executed as long as the value of `x` lies between 5 and 10, inclusive.

```
while(x >= 5 && x <= 10)
{
    body
}
```

If `exp1` and `exp2` are boolean expressions, then `exp1 && exp2` has value `true` if both `exp1` and `exp2` are true. Otherwise, `exp1 && exp2` is false.

In contrast, the body of the following `while` loop will be executed as long as the value of `x` is either less than 5 or greater than 10.

```
while(x < 5 || x > 10)
{
    body
}
```

If `exp1` and `exp2` are boolean expressions, then `exp1 || exp2` has value `false` if both `exp1` and `exp2` are false. Otherwise, `exp1 || exp2` is true.

Java defines a primitive data type `boolean`. A `boolean` variable can store one of two possible values, `true` or `false`. The statements

```
boolean playAgain = false;
boolean outOfBounds = true;
```

declare and initialize two variables of data type `boolean`.

We have now mentioned all eight of the primitive data types in Java: byte, short, int, long, float, double, char and boolean. All other data types in Java are defined by a class.

Experiment 3.7

Step 1. Compile and execute J03E07.java. Test the program by responding with 6. Record the results.

```
import java.util.Scanner;
class J03E07
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);

        int num;
        System.out.print("Enter an integer between 1 and 10: ");
        num = scan.nextInt();

        while(num < 1 || num > 10)
        {
            System.out.print("Error: Enter again: ");
            num = scan.nextInt();
        }
        System.out.println(num + " is my number too!");
    }
}
```

Step 2. Execute the program again. This time respond with numbers less than 1 or greater than 10 such as 15, 0, 23 . . . How many times can the body of the loop be executed? Record your observations and the results.

Step 3. Modify the program by changing the condition. Replace `(num < 1 || num > 10)` with `(num < 1 && num > 10)`. Execute the modified program, testing different input values. For what values of `num` is the body of the loop executed? Record and explain your results.

The if and if-else Statements

Another fundamental control statement in Java is the `if` selection statement. It has the form

```
if(condition)
{
    statementBlock;
}
```

If the *condition* is true, then *statementBlock* is executed; otherwise, *statementBlock* is ignored and execution continues with the statement following the `if` statement.

An expanded form of the `if` statement is the `if-else` statement. It has the form

```
if(condition)
{
    statementBlock_1;
}
else
{
    statementBlock_2;
}
```

This statement allows for one of two separate sets of instructions to be executed depending on whether the specified *condition* is true or false. If the condition is true, then *statementBlock_1* is selected to be executed; if it is false, *statementBlock_2* will be executed instead.

If any of the statement blocks within an `if` statement or an `if-else` statement consist of only one statement, then the opening and closing braces that enclose the block may be omitted.

Experiment 3.8

Step 1. Fill in the table below with the results of executing the program J03E08.java multiple times.

```
import java.util.Scanner;
class J03E08
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);

        int numerator, denominator, quotient;

        System.out.print("Enter an integer: ");
        numerator = scan.nextInt();
        System.out.print("Enter another integer: ");
        denominator = scan.nextInt();

        quotient = numerator / denominator;
        System.out.println(
            numerator + " / " + denominator + " = " + quotient);
    }
}
```

