

---

# Control Statements

In this laboratory 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
5. Learn the rudiments of the `for` statement
6. Learn the rudiments of the `switch` statement

## The while and do-while Statements

Recall that the while statement, introduced in Session 2, 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 5.1

**Step 1.** Predict the output of this program.

```
class J05E01
{
    public static void main(String[] args)
    {
        int numChips = 1;
        System.out.println("Open the bag and ...");
        do
        {
            System.out.println("  can't stop ...");
            numChips = numChips + 1;
        } while(numChips < 6);
        System.out.println("  CAN'T STOP!");
    }
}
```

---



---



---



---



---



---

**Step 2.** Compile and execute J05E01.java to confirm your answer. If your prediction was incorrect, explain the discrepancy.

---

---

---

---

---

---

---

### Experiment 5.2

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

```
class J05E02A
{
    public static void main(String[] args)
    {
        int x = 5;
        while(x < 5)
        {
            System.out.println(x);
            x++;
        }
        System.out.println("That's all folks!");
    }
}
```

---

---

---

---

---

---

---

**Step 2.** Compile and execute program J05E02A.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 J05E02B
{
    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 J05E02B.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.

In Session 2 we 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>&gt;</code>	is greater than	<code>y &gt; 6</code>	<code>true</code>
<code>&lt;</code>	is less than	<code>x + 3 &lt; y</code>	<code>false</code>
<code>&gt;=</code>	is greater than or equal to	<code>y - x &gt;= 0</code>	<code>true</code>
<code>&lt;=</code>	is less than or equal to	<code>x + 10 &lt;= 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>

In Experiment 3.7 we saw 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 5.4

**Step 1.** Compile and execute J05E04.java. Test the program by responding with 6. Record the results.

```
import java.util.Scanner;
class J05E04
{
    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.

---



---



---



---



---



---



## 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 5.5

**Step 1.** Fill in the table below with the results of executing the program J05E04.java multiple times.

```
import java.util.Scanner;
class J05E05
{
    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);
    }
}
```



## Experiment 5.6

**Step 1.** Compile and execute the program J05E06.java multiple times, each time modifying the response as indicated in the table below. Record the results.

```
import java.util.Scanner;
class J05E06
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);

        int x, y, z = 1;
        System.out.print("Enter an integer between 0 and 10: ");
        x = scan.nextInt();
        if(x >= 0 && x <= 10)
        {
            y = 1;
            while(y <= x)
            {
                z = z * y;
                y++;
            }
            System.out.println("Calculated value is " + z);
        }
    }
}
```

x value	Calculated result
-5	
0	
3	
5	
10	
12	

**Step 2.** Explain in your own words, what this program does.

---



---



---



---



---



---



---

**Step 3.** Modify the program in Step 1. Change `(x >= 0 && x <= 10)` to `(x >= 0 || x <= 10)`. Try the following values and record the result. Comment below.

x value	Calculated result
-5	
5	
12	
20	

---



---



---



---



---

## The for Statement

The `for` statement, like that of the `while` statement, accomplishes the task of repeating a block of statements as long as a certain condition remains true. It has the form

```
for (initialization; condition; update)
{
    body
}
```

and is semantically equivalent to the `while` statement

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

*initialization* is an assignment statement, or possibly a declaration-assignment statement, that initializes the variable that controls the loop. The *condition* is the test condition that determines if the body of the loop is executed. And, *update* is a statement that changes the value of the controlling variable. For example, the `for` loop below is controlled by the variable `j`. The first line of the `for` loop tells us that initially `j` is assigned a value of 2; the loop is executed as long as `j` is less than 4; and that each time through the loop, `j` is incremented by 1. This `for` loop

```
for (j = 2; j < 4; j++)
{
    System.out.println("Every day is");
    System.out.println("Earth Day!\n");
}
```

is equivalent to

```

j = 2;
while (j < 4)
{
    System.out.println("Every day is");
    System.out.println("Earth Day!\n");
    j++;
}

```

Both loops cause the message

```

Every day is
Earth Day!

```

```

Every day is
Earth Day!

```

to be displayed. More precisely, the variable `j` is first initialized to 2, the condition `j < 4` is tested and found to be true, and the body of the loop is executed. Then, `j` is incremented to 3 and the condition `j < 4` is tested again. Since the condition is still true, the body is executed a second time, and once more the value of `j` is incremented, this time to 4. Now when the condition `j < 4` is tested, the result is false, which terminates the loop.

### Experiment 5.7

**Step 1.** Predict the results of executing the program `J05E07.java`.

```

class J05E07
{
    public static void main(String[] args)
    {
        int j;
        System.out.println("Open the fridge ...");
        for(j = 0; j < 3; j++)
        {
            System.out.println("Chomp!");
        }
        System.out.println("... close the fridge");
        System.out.println("When done the loop counter is " + j);
    }
}

```

---



---



---



---



---



---



---



---



---



---

**Step 2.** Execute the program and compare your prediction with the actual results.

---

---

---

---

---

---

---

---

---

---

**Step 3.** A common programming error is to place a semicolon ( ; ) at the end of the line containing the opening statement of the `for` loop. Modify the program from Step 1 to include this error.

```
for(j = 0; j < 3; j++);
```

Compile and execute the program. Record the results.

---

---

---

---

---

---

---

**Step 4.** Explain the results from the program in Step 3.

---

---

---

---

---

---

---

---

---

---

**Step 5.** Java allows us to declare a variable any place within a method with the stipulation that a variable may not be used until it is declared. Once declared, a variable may only be used in the block of code in which it is declared. Modify the program by removing the statement

```
int j;
```

and instead, declare `j` inside the `for` statement as shown below.

```
for(int j = 0; j < 3; j++)
```

Compile the modified program and record what happens.

---

---

---

---

---

---

---

---

### Experiment 5.8

Rewrite the following program segment using a `for` loop rather than a `while` loop. Your segment should produce the same output as the original `while` loop. Convert your answer into a complete program, and test it to confirm that your version is correct.

```
glassPaper = 5;
while(glassPaper > 0)
{
    System.out.println("Reduce!\nReuse!\nRecycle!");
    glassPaper--;
}
```

---

---

---

---

---

---

---

---

---

---

---

---

## The switch Statement

The switch statement is an alternate selection statement that selects among many options. It has the form:

```
switch (variable)
{
    case option_1: statement_sequence_1; break;
    case option_2: statement_sequence_2; break;
    .
    .
    default: default_statement_sequence;
}
```

where *variable* is a variable identifier and *option\_1*, *option\_2*, etc., are possible values of that variable. The value of *variable* determines which of the statement sequences is executed. Each *statement\_sequence* can consist of zero or more statements. The `break` statement transfers control to the statement following the `switch` statement. If the value of *variable* does not appear as one of the listed options, then the default statement sequence is executed. (The terms `switch`, `case`, `default` and `break` are reserved words.)

As an example, if `sym` is defined to be of type `char`, then

```
switch (sym)
{
    case 'a': System.out.println("The symbol is an a."); break;
    case 'b': System.out.println("The symbol is a b."); break;
    case 'c': System.out.println("The symbol is a c."); break;
    default: System.out.println("The symbol is not a, b, or c");
}
```

reports whether the value of `sym` is `a`, `b`, `c`, or another symbol.

## Experiment 5.9

**Step 1.** Execute the following program `J05E09.java` and record the results.

```
class J05E09
{
    public static void main(String[] args)
    {
        char sym;
        sym = 'a';
        switch (sym)
        {
            case 'a': System.out.println("The symbol is an a.");
            case 'b': System.out.println("The symbol is a b.");
            case 'c': System.out.println("The symbol is a c.");
            default: System.out.println("The symbol is not a, b, or c");
        }
        System.out.println("Switch is completed");
    }
}
```

---

---

---

---

---

---

---

**Step 2.** What can you conclude about the absence of break statements in a switch statement? Correct the program by inserting the appropriate break statements, and confirm that the program performs correctly.

---

---

---

---

---

---

---

**Step 3.** Delete the `default` line from the program in Step 1, and change the statement

```
sym = 'a';
```

to the statement

```
sym = 'd';
```

Execute the modified program and record the results. What can you conclude about the absence of the `default` option?

---

---

---

---

---

---

---

---

---

---

## Optional Post-Laboratory Problems

- 5.1.** Write a program that asks the user for two integers and then responds by reporting whether the first value is greater than, less than, or equal to the second.
- 5.2.** Write a program that asks the user for two integers and then responds by printing the integers in numerical order.
- 5.3.** Write a program that asks the user for three integers and then responds by printing the integers in numerical order.
- 5.4.** Write a program that calculates a worker's weekly pay determined by the number of hours worked and the hourly rate. Any hours more than 40 are paid at 1.5 times the hourly rate. The user should enter the number of hours worked and the hourly rate.
- 5.5.** Write a program that determines whether a user-entered integer is or is not a prime number. Recall: A prime number is a positive integer with exactly two distinct factors, itself and 1. Hint:  $f$  is a factor of  $n$  if the remainder of  $n / f$  is 0. See Experiment 2.6.
- 5.6.** Write a program that uses a `switch` statement to print the word `one`, `two`, `three`, or `four` depending on whether the user enters the integer 1,2,3 or 4, respectively. Use the `default` option appropriately.
- 5.7.** Solve the preceding problem using `if-else` statements rather than a `switch` statement.
- 5.8.** Write a program that implements the game "Guess my number". After each guess respond with either "higher", "lower" or "You guessed it". To write this program, you will need to assign the target number a value. That is, the program will run with the same number each time. In the next session, we will see how to choose a random number for the target value.
- 5.9.** Follow these two steps to write a program that uses a nested loop, a loop inside of a loop, to print a 3 x 5 rectangle of asterisks (\*).

```
*****
*****
*****
```

- a. First, write a program that uses a `for` loop to print the string "\*\*\*\*\*" three times.
- b. Now, modify the code that you wrote in part a. Replace the statement that prints "\*\*\*\*\*" with a `for` loop that prints only one asterisk, five times. Hint: To print the "\*" use `print()` instead of `println()`.
- 5.10.** Modify the program that you wrote in problem 5.9, by allowing the user to enter both the number of rows and the number of columns, i.e. the number of asterisks in each row.
- 5.11.** Replace the nested `for` loops in problem 5.9 with `while` loops.
- 5.12** Write a program that prints a triangle with a user-entered number of rows. For example, if the user enters 4, the program should print one of the following triangles:

```
A.  *      B.   *      C.  ****      D.   ****      E.    *
   **      **      ***      ***      ***
  ***      ***      **       **       ****
 ****      ****     *        *        *****
```