Introduction to Computer Programming

Using the Java Programming Language

4. Looping

Computers are very good at executing your operations efficiently. They will follow your instructions more precisely than most humans. What they even do better than following instructions is that they will follow the same instructions over and over again and never take a shortcut. We use computers to do processes over and over all the time. While writing most programs, we have a need to tell the computer to perform a sequence of events more than once. We utilize loops in our code to achieve this. Java provides three looping structures for our convenience: For loop, While loop, and Do-While loop. Each have their own benefits.

Increment & Decrement Operators

One thing that we do in many loops is to increment or decrement a variable each time we go through the loop. We may want to count how many times we are performing the action, or we may want to continue a process until something has run out (decremented to zero).

Increment (++) and decrement (—) operators in Java programming let you easily add 1 to, or subtract 1 from, a variable. For example, using increment operators, you can add 1 to a variable named `a` like this:

```
a++;
```

An expression that uses an increment or decrement operator is a statement itself. That's because the increment or decrement operator is also a type of assignment operator because it changes the value of the variable it applies to.

You can also use an increment or decrement operator in an assignment statement:

```
int num1 = 5;
int num2 = num1--;
```

In this case, both `num1` and `num2` are set to 4. In the first line num1 is initially set to 5. In the second line, first `num1` is decremented to 4, and then the value of 4 is assigned to `num2`.

Increment and decrement operators can be placed before (prefix) or after (postfix) the variable they apply to. If you place an increment or decrement operator before its variable, the operator is applied before the rest of the expression is evaluated. If you place the operator after the variable, the operator is applied after the expression is evaluated.

Be careful with increment and decrement operators and understanding whether you should use a prefix or postfix option.

Examples of prefix and postfix operators

```
int a = 5;
int b = 3;
int c = a * b++;
int d = a * ++b;
```

After execution of these lines of code, `c` is set to 15 and `d` is set to 20.

Examples of using increment operator in a `Println`

```
num=5;
System.out.println("The number is: " + num++);
```

Would display the following to the console:

```
The number is: 5
```

However, look at the difference when we use a prefix operator.

```
num=5;
System.out.println("The number is: " + ++num);
```

Would display the following to the console:

```
The number is: 6
```

for Loop

A `for` statement in Java creates loops in which a counter variable is automatically maintained. The `for` statement lets you set an initial value for the counter variable, the amount to be added to the counter variable on each execution of the loop, and the condition that's evaluated to determine when the loop should end.

A `for` statement follows this basic format:

```
for (initialization-expression; test-expression; count-expression)
    statement;
```

The three expressions in the parentheses following the keyword `for` control how the `for` loop works:

- The `initialization-expression` is executed before the loop begins. This expression initializes the counter variable. If you haven't declared the counter variable before the `for` statement, you can declare it here.

- The `test-expression` is evaluated each time the loop is executed to determine whether the loop should keep looping. This expression tests the counter variable to make sure that it's still less than or equal to the value you want to count to. The loop keeps executing as long as this expression evaluates to true. When the `test-expression` evaluates to false, the loop ends.

- The `count-expression` is evaluated each time the loop executes. Its job is to increment the counter variable.

The statement part of the for loop can either be a single statement or a block of statements contained in a pair of braces.

Here's a simple `for` loop that displays the numbers 1 to 10 on the console:

```
for (int i = 1; i <= 10; i++)
    System.out.print(i + " ");
```

After running the code above, the following will be displayed to the console:

```
1 2 3 4 5 6 7 8 9 10
```

If you declare the counter variable in the initialization statement (as in the previous example), the scope of the counter variable is limited to the `for` statement itself. Thus, you can use the variable in the other expressions that appear within the parentheses and in the body of the loop, but you can't use it outside the loop.

IMPORTANT

If you want, you can declare the counter variable outside the `for` loop. Then, you can use the counter variable after the loop finishes. For example:

```
int i;

for (i = 1; i <=10; i++)
   System.out.print(i + " ");

System.out.println("\nThe final value is " + i);
```

The results of the above code would produce the following displayed to the console:

```
1 2 3 4 5 6 7 8 9 10
The final value is 11
```

while Loop

A while statement in Java programming creates a loop that executes continuously as long as some conditional expression evaluates to true. The basic syntax is this:

```
while (expression)
    statement
```

The while statement begins by evaluating the expression. If the expression is true, statement is executed. Then the expression is evaluated again, and the whole process repeats. If the expression is false, statement is not executed, and the while loop ends.

The statement part of the while loop can either be a single statement or a block of statements contained in a pair of braces.

Here's a snippet of code that uses a while loop to print the even numbers from 2 through 20 on the console:

```
int number = 2;
while (number <= 20) {
System.out.print(number + " ");
    number += 2;
}
```

If you run this code, the following output is displayed in the console window:

```
2 4 6 8 10 12 14 16 18 20
```

The conditional expression in this program's while statement is `number <= 20`. That means the loop repeats as long as the value of number is less than or equal to 20. The body of the loop consists of two statements. The first prints the value of number, followed by a space to separate this number from the next one. Then the second statement adds 2 to number.

do...while Loop

A do…while statement in Java programming is similar to a while statement but with a critical difference: In a do…while statement, the condition that stops the loop isn't tested until after the statements in the loop have executed. The basic form of a do…while statement is this:

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

The `while` keyword and the expression aren't coded until after the body of the loop. As with a while statement, the body for a do…while statement can be a single statement or a block of statements enclosed in braces.

Also, notice that a semicolon follows the expression. do…while is the only looping statement that ends with a semicolon.

An example of a loop that counts from 1 to 20 can be seen in this code:

```
int number = 1;
do {
    System.out.print(number + " ");
    number ++;
} while (number <= 20);
```

The output of this code would result in the following displayed to the console:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

The statement or statements in the body of a do…while statement always execute at least one time. By contrast, the statement or statements in the body of a while statement aren't executed at all if the while expression is false the first time it's evaluated.

IMPORTANT

Early Termination of a Loop

Often times it is convenient to terminate a loop immediately at some point. The `break` statement does just that and will place the execution of the program at the line following the end of the loop, essentially "kicking you out of the loop." The `break` statement in Java programming language has the following two usages:

- When the `break` statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

- It can be used to terminate a case in the switch statement

Let's see how we may use it. Keep in mind that we can restructure our logic in many cases to not need the `break` statement.

Example

```
for (int i = 1; i <= 10; ++i) {
     if (i == 5) {
          break;
     }

     System.out.println(i);
}
```

This code segment would produce the following display to the console:

```
1
2
3
4
```

Nested Loops

Sometimes you need to process something using multiple loops. For example, when working with tabular data, you might use one loop to process the rows and another loop to process the columns. There are multiple columns for each row, so the Columns loop appears within the Rows loop. Placing one repeating loop within another is called nesting the loops.

Each iteration of the main loop executes the entire subordinate loop. So, when you start processing the first row, it executes all the column tasks for that row before moving to the next row.

Nesting is the process of enclosing one structure within another of the same type. Java uses nesting in a number of ways, so you'll see this term used quite often. When working with structures, one structure acts as a container to hold the other structure.

The container structure is called the main, or parent, structure. The structure within the main structure is called the subordinate, or child, structure.

The multiplication tables are one of the better ways to demonstrate nesting because you need to create a loop for rows and another for columns. In addition, you need to create the headings that show the numbers being multiplied, which means using an additional loop.

Code for a multiplication table generator can be seen here.

```
/*
 * Programmer: Mr. Kulla
 * Date: January 1, 2020
 * Purpose: Generates a Multiplication Table.
 */

public class MultiplicationTableGenerator {
      public static void main(String[] args) {
           final int ROWMAX=10;
           final int COLMAX=10;

           for (int i=1; i<=ROWMAX; i++) {
                System.out.printf("%4d",i);
                for (int j=1; j<=COLMAX; j++)
                     System.out.printf("%4d", (i*j));

                System.out.println();
            }
      }
}
```