3 Program Flow & Public Classes

In this short lesson we discuss two topics that we have seen earlier: Control of Program Flow and Using Public Classes.

3.1 Controlling Program Flow

In the previous lessons we have observed several ways that a Java programmer can control the flow of a classes methods. The simplest of programs consists of a list of statements that are executed in order, one at a time. Unfortunately, listing every step of a program one-by-one is at best inefficient, and often insufficient method of programming. A program that prints the first 25 integers to the screen, for example, could be written as 25 separate `println` statements — or one simple loop. On the other hand, a program that prints the first $n$ integers (with $n$ obtained as program input) cannot be written without a loop! (Afterall, how many `println` statements would you require?)

The Java statements that control program flow fall into three categories: loop statements, conditional statements, and transfer statements.

3.1.1 Loops

The most important method of controlling flow in is the `for` loop, with which we have now had considerable experience. The general form of a `for` statement is:

```java
for( Initialization ; Test ; Increment ) { . . . }
```

The portion of the `for` statement appearing within the `braces` is the list of statements that is repeated; the portion in the `parentheses` describes how the loop is to be performed. The `Initialization` is a statement (or series of statements separated by commas) that is run once at the beginning of the first iteration of the loop. The initialization step is usually used to declare and initialize a `counter`. The `Test` is a boolean (True/False) condition that is checked at the beginning of the loop. If the boolean condition is `true` then the statements between the braces are run; if it is `false`, the `for` statement ends. Finally the `Increment` is a statement (or series of statements separated by commas) that is run at the end of every loop to prepare for the next iteration.

Java has two other kinds of loops that appear less frequently. The `while` loop has the form:
while (Test) {
    . . .
}

As with the for loop, the Test is a boolean expression. The statements between the braces are repeatedly executed until the boolean Test becomes false.

The do-while loop has the form:

do
    . . . Statements . . .
while (Test);

In this case, again, the statements between the do and while are repeatedly executed until the boolean Test becomes false.

The difference between the while and the do-while loops is that the boolean condition in the while loop is tested before the statements are executed (so, it is possible for the statements to never be executed), while the boolean condition in the do-while loop is tested after the statements are run, so the statements must be run at least once.

3.1.2 Conditionals

Java provides one conditional statement: the if-else statement. In its simplest form, the if statement has the form:

    if (Test) {
        . . .
    }

The boolean condition Test is checked and the statements between the braces are run if the Test evaluates to true. If the Test is false, the statements are not executed.

A more elaborate form of the if statement permits inclusion of statements to be run when the condition is false. The general form of such a conditional statement is:

    if (Test)
        {
            . . .
        }
    else
        {
            . . .
        }

The condition Test is checked. If the result is true, the first group of statements is executed; if the result is false, the second group of statements is executed.
3.1.3 Transfer

We have encountered two of the three common methods of transferring control from one place to another in a Java method: the return statement and the break statement.

The return signals the immediate end of a Java method and provides a value to be given back to the caller of the method. (Each of our example programs that contain two methods include a main method that calls a secondary method that has a return statement.)

The break statement signals the immediate end of a loop. If the break statement is encountered within a for, while, or do-while loop, the loop immediately ends, and the method continues with the next statement following the loop.

We mention one additional transfer statement: the continue statement. Like the break statement, the continue statement can appear within any of the loop statements. The continue statement causes the current iteration of the loop to end and the next next iteration to begin immediately.

3.2 Using Public Classes

One of the most important features of Java is its rich collection of classes (the core libraries) that have been already written and are available for use. These handle numerous standard tasks including networking, building and controlling windows and menus, and sophisticated mathematics operations. The process of using a library function is straight forward. First, the library must be declared with an import statement. The import statements are typically collected before the beginning of a class definition and identifies the classes that are to be used. Typical import statements look like this:

```java
import java.math.BigInteger;  /* Import the BigInter Class */
import java.math.*;          /* Import *all* Math Classes */
import java.util.*;          /* Import *all* Utility Classes */
```

Once a class has been imported to your Java source, you are free to invoke its methods within your own methods. In the following example, we make use of the class BigInteger, which provides utilities for computing with integers of arbitrary size. (It does this by manipulating strings of digits, much
as you would if you computed by hand.) The following programs computes powers, \( m^n \), of integers. First a version using \texttt{int}:

```java
/*
   June 9, 2020 Walter Carlip Compute powers.
*/
public class Powers {
    public static int power(int m, int n){
        int answer = 1;
        for(int i = 2; i <= n ; i++){
            answer = answer * m;
        }
        return(answer);
    }
    public static void main(String[] args) {
        System.out.println(power(Integer.parseInt(args[0]),Integer.parseInt(args[1])));
    }
}
```

As some of you have already noticed, this program rapidly overwhelms the \texttt{int} data type! (Try computing, for example, \( 2^{50} \).) Even changing every \texttt{int} to \texttt{long} only succeeds in delaying the onset of overflow.

Fortunately, Java provides the math library, with its \texttt{BigInteger} class, which provides utilities for computing with arbitrarily large integers. Here’s a revision of the program above that can handle arbitrarily large powers:
import java.math.BigInteger;
public class BigPowers {
    public static BigInteger power(String m, String n){
        BigInteger bigbase = new BigInteger(m);
        BigInteger bigpower = new BigInteger(n);
        BigInteger biganswer = new BigInteger(m);
        long stop = Long.parseLong(n);
        for(long i = 2; i <= stop ; i++){
            biganswer = biganswer.multiply(bigbase);
        }
        return(biganswer); 
    }
    public static void main(String[] args) {
        System.out.println(power(args[0],args[1]));
    }
}

In order to use a Java library, you need to know what libraries are available and what methods they contain. One way to obtain this information is to purchase a manual that describes the library functions. (A good place to start might be with David Flanagan’s book Java in a Nutshell: A Desktop Quick Reference, Second Edition, published by O’Reilly in 1997. Alternately, a huge amount of information is available on the Internet. An extensive listing of Java library calls can be found at the following web site:

    http://docs.oracle.com/javase/6/docs/api/
Summary

- **Java** programs alter the flow of statements within methods with **Loops**, **Conditionals**, and **Transfer Statements**.

- There are three kinds of loops, **for**, **while**, and **do-while**. (In reality, one rarely needs more than the **for** statement.)

- The basic conditional statement is the **if-else** statement.

- There are three ways to transfer program control from one part of a program to another, **return**, **break**, and **continue**.

- **Java** has lots of built-in libraries that perform a myriad of functions.