Lab 2 – Java Syntax and Methods

Goal:
The goal of this lab is to learn the basic syntax of Java and to learn how to write and use methods.

Part 1 – The basic “total finder” program
To understand the basic syntax of Java you’re going to be writing a basic total finding program. The program will find the total of a series of integers and output the total to the screen. Start off by creating a new project in Eclipse. You can use the same workspace you used in the first lab. Once you’ve created your project create a package under the source folder called "totalFinderPackage". In that package create a class called "TotalFinder" with the main method inside. Keep in mind that you should be commenting your code as you write it. If a variable name isn’t self-explanatory, you should provide a comment saying what it is for. In addition, any non-trivial code should have a comment explaining what it does, and every method should be documented with its purpose, what its parameters are, and what it returns.

To start off you’re just going to total the first 10 integers starting from 1. First, you’ll need a variable to keep track of the total. So create a variable called "total" and set its value to 0. Next you’ll need to add the first 10 integers to that total. When you need to repeat an action a set number of times you should use a for loop. The syntax of a for loop in Java, modified for use here, is as follows:

```
for (int i = 1; i <= 10; i++) {
    total = total + i;
}
```

This will repeat whatever code is within the braces while the variable i is less than or equal to 10. Since i starts at 1 and its value is incremented after each loop iteration, the loop will run 10 times. Inside the braces add the value of i to the total by writing the following:

```
total = total + i;
```
or equivalently

```
total += i;
```

The last thing to do is to output the total to the screen. But just outputting the total isn’t very informative. If you just saw "55" printed on the screen you wouldn’t know what it meant. Your program should include what "55" means. You should create a string to output along with the total. Add the following line after the for loop:

```
String output = "The total of the integers from 1 to 10 is " + Integer.toString(total);
```
The value that this string takes is the first part, "The total of the first 10 integers is ", concatenated with the string conversion of the total. It is syntactically correct to leave out the "Integer.toString" part and just add the total, but you should always be explicit when you convert one data type to another (called casting) for both clarity and error avoidance. The compiler might not always use the data type you think it will. Casting will be discussed in more detail below.
Finally, write one more line to output the string. Compile and run your program. You should see "The total of the integers between 1 and 10 is 55" written to the console.

Part 2 – A different series of numbers

What if you wanted to change the numbers totaled from the first 10 to something else? You would have to go back and change your code in a few different spots; specifically, you would need to change the for loop initialization, the for loop condition, and the output string. This is why hard-coding values is bad. If you want to make a change later you have to go back and change every line where that variable is used. And what if you used the value 10 elsewhere for a different purpose? It’s much harder to be sure that you’re changing the correct number, making it easy to inject errors into your code.

Instead, you should use constants to define these numbers. A constant is just what it sounds like – a variable whose value is constant. You can declare a variable as constant by beginning the declaration with the "final" keyword. For example, "int x = 0;" would become "final int X = 0". It’s important to note that constant variable must have their values assigned when they’re declared since the variable can’t be assigned to elsewhere. Change your program so that it knows where to start and end the series by reading from two constant variables called "STARTING_VALUE" and "ENDING_VALUE". Remember to make changes in any place you used 1 or 10 previously and to convert the integers to a string when you make the output. Now you can change just the values that you initialize STARTING_VALUE and ENDING_VALUE to instead of going back through your code when you want to use a different series of numbers.

Part 3 – Total finder as a method

Finding the total of a series of numbers is something you might want to do multiple times in a program, so it’s good to make it into a method. Creating a method to find a series of numbers would allow you to call the method instead of rewriting the code as well as make the code more readable. The method should have a name that lets you know what the method does, so a good name would be "findTotalOfSeries". The method needs to know where the series starts and ends, so the starting and ending values (which are integers) should the method parameters. Finally, the method returns the total, which is an integer. Your method header should look like this:

    public static int findTotalOfSeries(final int startingValue, final int endingValue)

It’s okay if you don’t know what the keywords public and static mean. You’ll learn about them next week. The keyword final in this context just means that the method can guarantee that the argument won’t be changed in the method.

Don’t forget to document your method. Quality documentation should tell you what the method does, what the parameters are, and what the return value is. Here’s some documentation that would be useful here:
/**
 * Finds the total of the numbers in a series.
 *
 * @param startingValue The value at which the series starts.
 * @param endingValue The value at which the series ends.
 * @return The total of the numbers in the series.
 */

As you can see this tells you everything you need to know about a method.

Now just copy and paste your code into your method and add a line to return the total. The method code should look like this:

```java
/**
 * Finds the total of the numbers in a series.
 *
 * @param startingValue The value at which the series starts.
 * @param endingValue The value at which the series ends.
 * @return The total of the numbers in the series.
 */

public static int findTotalOfSeries(final int startingValue,
                                    final int endingValue) {
    int total = 0;
    for (int i = startingValue; i <= endingValue; i++){
        total = total + i;
    }
    return total;
}
```

This method is almost perfect, but it’s missing a crucial detail; what happens if the given ending value is smaller than the starting value? Your method should handle every possible case and your documentation should be explicit about unintended use. In this case you just need to test for invalid arguments and do something about it. Normally you would do something called throwing an exception, but you won’t learn about exceptions until much later. Suffice it to say it’s basically the program telling itself that something went wrong. Instead of throwing an exception, you can simply return a default value (0 is a good value) and make sure it’s documented. Just add an if statement whose body returns 0, and add a line in your documentation saying that the method returns 0 if the given ending value is less than the given starting value.

Once you’re finished with your method you can just replace the code in main with a call to your method. You’ve reduced this:

```java
int total = 0;   // The total of the sequence of numbers.
for (int i = startingValue; i <= endingValue; i++){
    total = total + i;
}
```
As you can see, the new code has far more obvious meaning and it’s reusable.

**Part 4 – Boolean values, floats, and control structures**

The last thing that you’re going to do is to create two more simple methods that demonstrate boolean values, floats, and control structures. Keeping with the theme of finding totals, the two methods will 1) determine if the total of a series is even and 2) calculate the average value of a series.

The first method, which will be the one to determine evenness, should have the return type boolean. It will also need to know the starting and ending values of the series so that it can calculate the total. Thus the method header will be the following:

```java
public static boolean isSeriesTotalEven(final int startingValue, final int endingValue)
```

Again, document what the method does, its parameters, and its returns value (true if the total is even, false otherwise).

The method itself is very simple – barely long enough to warrant having a method. First, it should calculate the total of the series (which you now have a method for) and then return true if the total is even. If it’s not even then it returns false. Go ahead and try writing the method. You can determine if a number is even by using the modulus operator % . If x % 2 = 0 then x is even because there’s no remainder. In fact, the Java expression x % 2 == 0 resolves to a boolean value.

Once you’ve finished writing the method add some if-else statements in the main method that outputs different string depending on whether the output is odd or even (a simple "It’s even" or "It’s odd" will do).

The second method, which calculates an average, is a little more complicated. The method needs to return a value that includes a decimal (float), but the numbers used in the calculation are integers. The header is simple:

```java
public static float findSeriesTotalAverage(final int startingValue, final int endingValue)
```

The calculation you need to do is just to divide the total of the series by the size of the series. When you actually perform the calculations you’ll need to cast the variables from integers to floats. Casting a variable to a different type tells the compiler to treat it like that type, so in this case the integer will be treated as a float (thus giving you decimal precision). To explicitly cast a variable from one type to another just add the target type in parenthesis before the variable. For example, the following code casts the return value of FindTotalOfSeries(), which is an integer, to a float and stores it in a variable of type float.

```java
float total = (float)findTotalOfSeries(startingValue, endingValue);
```

Likewise, you’ll need to treat the size of the series as a float:
float size = (float)(endingValue - startingValue + 1);

Now when you perform a mathematical operation the answer will have decimal precision. Go ahead and divide the total by the size and return that value, the type of which is float. If you had not cast the integers to floats then the answer, which would be an integer type, would have its decimal portion truncated. There are a few different ways to cast one type to another. The one used here is called explicit cast.

Don’t forget to check each method’s input to see if the given values are valid. Choose a default value to return (false or 0 would make the most sense in the absence of exceptions), and again make sure to document what it returns under those conditions.

Submission
When you’ve finished the lab you can export the project to a zip file and submit the zip file via web-based turnin. To export a project, go to File | Export... and choose Archive File from the General folder. Check off the entire project in the top left window and make sure the format is .zip and not .tar. Finally, give the archive file a name and click finish.

Acknowledgement
Thank you to Tom Whirtner, who designed this lab for CS 220X.