**CMSC203 Lab 2**

**Java Fundamentals**

**Lab Objectives**

Use command-line and IDE compilation and running

Write arithmetic expressions to accomplish a task

Use casting to convert between primitive types

Use a value-returning library method and a library constant

Use string methods to manipulate string data

Communicate with the user by using the Scanner class and dialog boxes

Create a program from scratch by translating a pseudocode algorithm

Be able to document a program

GitHub

**Introduction**

This lab is designed to give you practice with some of the basics in Java. We will correct syntax, run-time, and logic errors while looking at mathematical formulas in Java. We will compile and run a program from the command line, and from an IDE.

There are three types of error:

1. Syntax Errors—errors in the “grammar” of the programming language. These are caught by the compiler and listed out with line number and error found. You will learn how to understand what they tell you with experience. All syntax errors must be corrected before the program will run. If the program runs, this does not mean that it is correct, only that there are no syntax errors. Examples of syntax errors are spelling mistakes in variable names, missing semicolons, unpaired curly braces, etc.
2. Run time errors—errors that do not occur until the program is run, and then may only occur with some data. These errors emphasize the need for completely testing your program.
3. Logic Errors—errors in the logic of the algorithm. These errors emphasize the need for a correct algorithm. If the statements are out of order, if there are errors in a formula, or if there are missing steps, the program can still run and give you output, but it may be the wrong output. Since logic errors are not reported by Java, you may not realize you have errors unless you check your output. It is very important to know what output you expect. You should test your programs with different inputs, and know what output to expect in each case. You should usually test your programs with several “normal” cases, and also with “boundary conditions” and known erroneous inputs. Calculate each case by hand before running your program so that you know what to expect. You may get a correct answer for one case, but not for another case. This will help you figure out where your logic errors are.

We will explore the difference between integer division and division on your calculator as well as reviewing the order of operations. We will also learn how to use mathematical formulas that are preprogrammed in Java. This lab also introduces communicating with the user. We will need to learn how to program user input, using the Scanner class. We will also learn the method call needed for output.

We will bring everything we have learned together by creating a program from an algorithm. In calculus, we often need to compute values of functions, for example, when computing limits, so we will learn how to compute a function that returns a value. On your calculator there are buttons to be able to do certain operations, such as raise a number to a power or use the number pi. Similarly, in Java, we will have programs that are available for our use that will also do these operations. Mathematical operations that can be performed with the touch of a button on a calculator are also available in the Math class. We will learn how to use a Math class method to cube the radius in the formula for finding the volume of a sphere.

Finally, you will document the program by adding comments. Comments are not read by the computer; they are for use by the programmer. They are to help a programmer document what the program does and how it accomplishes it. This is very important when a programmer needs to modify code that is written by another person.

**Task #1 Correcting Syntax and Run-time Errors in Formulas**

In this task you will use an existing Java program and correct several errors.

1. Download the file NumericTypes.java from Blackboard.

2. Take screen shots of each of the following tasks using the command line, and using Eclipse. Compile the source file in the command line\* and in Eclipse (or other IDE). Note that there is a syntax (compile-time) error. Correct the error, recompile, and run the program.

3. Notice that there is a run-time error. Fix that error, recompile, and re-run the program.

\*See the file *CommandLine Java.pptx* to understand how to use the command prompt tool in Windows for compiling and running Java programs.

**Task #2 Correcting Logic Errors in Formulas**

1. In this task you will use your partially-corrected NumericTypes.java program and now correct logic errors.

2. Run the program and observe the output. Some of the output is ***incorrect***. You need to **correct logic errors** in the *average* formula and the temperature conversion formula. The logic errors could be due to conversion between data types, order of operations, integer division, or formula problems.

The necessary formulas are:

𝑎𝑣𝑒𝑟𝑎𝑔𝑒=(𝑠𝑐𝑜𝑟𝑒1+𝑠𝑐𝑜𝑟𝑒2)/𝑛𝑢𝑚𝑏𝑒𝑟𝑂𝑓𝑆𝑐𝑜𝑟𝑒𝑠

𝐶=(5/9)\*(𝐹−32)

3. Each time you make changes to the program code, you must compile again for the changes to take effect before running the program again.

4. When you re-run the program, make sure that the output makes sense. The average of 95 and 100 should be 97.5 and the temperature that water boils is 100 degrees Celsius, or 212 degrees Fahrenheit.

**Task #3 Using the Scanner Class for User Input**

In this task you will read in the user’s name, and let the user enter values to replace the “hard-coded” values in Task #2.

1. Add an import statement above the class declaration to make the Scanner class available to your program.

2. In the main method, create a Scanner object and connect it to the System.in object. Reuse this Scanner object to read in each of the following entries.

3. Prompt the user to enter the first score.

4. Read the first score from the keyboard using the nextInt() method, and store it into the variable called score1 (comment out the line that originally defined score1).

5. Prompt the user to enter the second score.

6. Read the second score from the keyboard using the nextInt() method, and store it into the variable called score2 (comment out the line that originally defined score2).

7. Prompt the user to enter another temperature in Fahrenheit after the original temperature conversion is printed out.

8. Read the temperature and print out the result in Celsius.

9. Compile, debug, and run, using other score values as test data.

10. **Submit** your completed NumericTypes.java **(NOT the class file)** and screen shots to Blackboard**.**

**Task #4 – Create a program from scratch using predefined Math Functions**

In this task you will create a new program that calculates the result of a mathematical formula. You will use string expressions, assignment statements, input and output statements to communicate with the user.

1. Create a new file in your IDE or text editor. Create the shell for your first program by entering:

public class SphereVolume

{

public static void main(String[] args)

{

// add your declaration and code here

}

}

2. Save the file as SphereVolume.java. In Java, each public class is written in a file with the same name as the class name.

3. Translate the algorithm below into Java, using the hints in 4 through 7 below. Don’t forget to declare variables before they are used. Each variable must be one word only (no spaces). Java naming convention is that variables start with a lower-case letter (your compiler will not object, though, if you do not follow this rule).

Pseudo-code for SphereVolume.java:

*Print the purpose of the program*

*Print a prompt asking for the diameter of a sphere*

*Read the diameter*

*Calculate the radius as diameter divided by 2*

*Calculate volume of the sphere using the formula (see Task #3b)*

*Print the volume*

4. Using the Scanner class as in Task #2 above, print a prompt to the user asking for the diameter of a sphere.

5. Read in diameter which you will store in a variable called *diam.* You must declare *diam* before you use it. Declare it as a double. There are two ways to read in the value.

a. First option is to use in.next(), which returns the user input is returned as a String. Then you will need to use the method Double.parseDouble(*<user input>*) to change the String user input into a double.

b. Second option is to use in.nextDouble(), which returns a double directly. Either choice will cause an exception if the user enters something that does not parse into a double, but do not do any error-checking for now.

6. The diameter is twice as long as the radius, so calculate and store the radius in an appropriately named variable.

7. Calculate volume of the sphere using the formula:

Use the built-in Java math methods Math.PI for π and Math.pow to cube the radius.

8. Print the volume with appropriate labels.

9. Compile the program and debug, repeating until it compiles successfully.

10. Run the program and test it using the following sets of data and record the results:

|  |  |  |
| --- | --- | --- |
| Diameter | Volume (hand calculated) | Volume (resulting output) |
| 2 |  |  |
| 25.4 |  |  |
| -5 |  |  |

11. **Submit** your completed SphereVolume.java **(NOT the class file)** and test table to Blackboard.

**Task #4 – Upload Java Files to GitHub**

Upload your completed NumericTypes.java and SphereVolume.java in a directory named CMSC203\_Lab2 to your repo in GitHub (see Lab 1).