Introduction to Java

CS9053

Section I

Thursday 6 PM – 8:30 PM

Prof. Dean Christakos

Assignment 6

March 18, 2021

Due: March 24, 2021

**Part I: Generics**

Create a class called **MaxFinder** with a Generic constraint into which you can add Numbers and has a method max() which will return the largest value within that collection. It should be able to take any kind of Numbers. It could take all Numbers, or it could take just Doubles, or it could take just Integers. How you implement it is up to you, but it should have methods:

add(T t) -> add an object of type T

add(ArrayList al) -> add an array list of objects. This needs to take an ArrayList of type T OR an ArrayList that’s a subclass of T. For example, if your MaxFinder is parameterized for Number, the add(ArrayList al) method needs to take ArrayList <Integer>, ArrayList <Double>, etc., as well as ArrayList <Number> (hint: use wildcards)

T max() -> return the maximum values object with return type T

To think about: if you have two numbers:

Number n1

Number n2

How do you compare n1 to n2? The Number class does not have a compareTo method, so you’ll have to think about this.

Have code that tests this out.

**Part II: Events**

The code in SimpleAddition.java will create a window with two text fields containing numbers and a JLabel that displays the sum. Any time either text field is changed *and then after you press return*, an event is generated. Create event listeners for each text field such that the value of the Sum data field is updated to have the latest sum of the two addends.

You do not have to worry about any of the code in setupComponentValues(), nor do you have to understand any GUI development.

You must add the correct arguments to addend1Field.addActionListener() and addend2Field.addActionListener(). Minimize redundancy of code.

Part **III Single Abstract Method Interfaces and Lambdas:**

In part III, there are two files. The first is called MathOperation.java, a Single Abstract Method Interface which has one method, “operation,” which returns a double and takes two doubles as parameters. The other is ResultPrinter.java. It prints out the result of MathOperation, which you can pass into the constructor, on two arguments, a and b. There are also two static methods: go(double a, double b, MathOperation op), which takes in two arguments and a MathOperation and go(Collection<Pair<Double,Double>> c, MathOperation op), which takes in some kind of collection of pairs of doubles which will be applied to the MathOperation.

Implement the three versions of “go”.

Pass in the MathOperation parameter to the constructor, and to the static go methods as a Lambda.

Instantiate ResultPrinter with a simple math operation that does addition. Execute rp.go() and show that it gives the correct result.

Implement and execute the static method ResultPrinter.go with two arguments and a Lambda function that does multiplication.

Implement and execute the static method FunctionResultPrinter.go with the array list of doubles and a lambda function that does division and have the method loop through the array list of pairs and execute that operation on the two members of the pair as arguments.

Show what the output printed out is.

Note: A “Collection” is a superclass of ArrayList. This shouldn’t matter for your code, but using a Collection makes the code more general. We will get to the different Collection classes later in the semester.