Introduction to Java

CS9053

Thursday 6 PM – 8:30 PM

Prof. Dean Christakos

June 30th, 2023

Due: July 7th, 2023

Part I: Generics

1. Problem Description:

A complex number is a number of the form , where a and b are real numbers and i is . The numbers a and b are known as the real part and imaginary part of the complex number, respectively. You can perform addition, subtraction, multiplication, and division for complex numbers using the following formula:









You can also obtain the absolute value for a complex number using the following formula:



(A complex number can be interpreted as a point on a plane by identifying the  values as the coordinates of the point. The absolute value of the complex number corresponds to the distance of the point to the origin)

**Design a class named ComplexNumber for representing complex numbers and the methods add, subtract, multiply, divide, abs for performing complex-number operations, and override toString method for returning a string representation for a complex number. The toString method returns a + bi as a string. If b is 0, it simply returns a**.

**Provide three constructors Complex(a, b), Complex(a), and Complex().** Complex() creates a Complex object for number 0 and Complex(a) creates a Complex object with 0 for b. Also provide the getReal () and getImaginary () methods for returning the real and imaginary part of the complex number, respectively.

Your Complex class should also implement the Cloneable interface.

Write a test program that prompts the user to enter two complex numbers and display the result of their addition, subtraction, multiplication, and division. Here is a sample run:

<Output>

Enter the first complex number: 3.5 5.5

Enter the second complex number: -3.5 1

(3.5 + 5.5i) + (-3.5 + 1.0i) = 0.0 + 6.5i

(3.5 + 5.5i) - (-3.5 + 1.0i) = 7.0 + 4.5i

(3.5 + 5.5i) \* (-3.5 + 1.0i) = -17.75 + -15.75i

(3.5 + 5.5i) / (-3.5 + 1.0i) = -0.5094 + -1.7i

|3.5 + 5.5i| = 6.519202405202649

<End Output>

The template for the code is:

**import** java.util.Scanner;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Scanner input = **new** Scanner(System.*in*);

System.*out*.print("Enter the first complex number: ");

**double** a = input.nextDouble();

**double** b = input.nextDouble();

Complex c1 = **new** Complex(a, b);

System.*out*.print("Enter the second complex number: ");

**double** c = input.nextDouble();

**double** d = input.nextDouble();

Complex c2 = **new** Complex(c, d);

System.*out*.println("(" + c1 + ")" + " + " + "(" + c2 + ")" + " = " + c1.add(c2));

System.*out*.println("(" + c1 + ")" + " - " + "(" + c2 + ")" + " = " + c1.subtract(c2));

System.*out*.println("(" + c1 + ")" + " \* " + "(" + c2 + ")" + " = " + c1.multiply(c2));

System.*out*.println("(" + c1 + ")" + " / " + "(" + c2 + ")" + " = " + c1.divide(c2));

System.*out*.println("|" + c1 + "| = " + c1.abs());

Complex c3 = (Complex)c1.clone();

System.out.println(c1 == c3);

System.out.println(c3.getReal ());

System.out.println(c3.getImaginary ());

}

}

**class** ComplexNumber {

// Write your code

}

1. This is a followup to Question 1

A complex number is a type of Number. You should have figured that out, so I don’t feel like I am giving anything away. Implement ComplexNumber as a subclass of Number. For simplicity’s sake, getInt(), getDouble(), etc. should all return the real value of the complex number, and that will be what you will use for comparison when implementing the class **maxFinder.**

Create a class called **maxFinder** with a Generic constraint into which you can add a collection of Numbers and has a method max() which will return the largest value within that collection. How you implement it is up to you, but it should have methods:

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

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

**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, 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 three files. The first is Pair.java. You should implement a Pair class that is parametized with two types, and where each pair can be retrieved with getA() and getB().

For example, you would declare:

Pair<String, Integer> p = new Pair<String, Integer>();

Or

Pair<String, Integer> p = new Pair<String, Integer>(“Hello”, 77);

Alternately, to set the first part of the pair, A, you would call p.setA(“Hello”). To set the second part of the pair, B, you would call p.setB(77). The result of p.getA() would return a String (and only a String), and p.getB() would return an Integer (and only an Integer). Obviously, this should work for any two classes, not just Strings and Integers. You will use that code for the next part:

The second 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 last 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.