coe318 Lab 2
ComplexNumber objects

Objectives

- Implement a `ComplexNumber` class.
- Learn how immutable objects work.
- Create a project with more than one class.
- Duration: one week.

Overview

In mathematics, complex numbers combine two real numbers and can be thought of as specifying a point on a plane. The most common ways to express the two components of the complex number are:

- *rectangular*: where the two numbers represent the x- and y-components of the point;
- *polar*: where one number represents the distance between the origin and the point and the other represents the angle between the x-axis and the line connecting the origin and the point.

In this lab, we only consider the rectangular version.

The Design of ComplexNumber

The design of a class means specifying all of its public members. Usually, only some methods and constructors are public (and hence designed); it is highly unusual for instance variables to be public.

In Java, a design is expressed in javadocs: specially formatted comments in the source code that describe the API for the class.

You are given the design for the `ComplexNumber` class below. The design specifies methods such as `getX()`, `add(ComplexNumber z)`, etc. The source code can be more conveniently accessed here

The source code provided consists mainly of *method stubs*: methods that compile but produce dummy results. A notable exception is the method `toString()`; this method, which gives a String representation of a complex number does work and should not be modified. (This method is implicitly invoked in the testing class that produces output.)

The main objective of the lab requires that you fix the method stubs so that the program works.

ComplexNumber.java source code

```
package coe318.lab2;
/**
 * ComplexNumber models a complex number expressed
 * in rectangular form (real and imaginary parts).
```

```
 * It is an <em>immutable</em> object.
 *
 * @author Your Name
 */
public class ComplexNumber {
  //Instance variable declarations

  /**
   * Construct a ComplexNumber given its
   * real and imaginary parts.
   * @param re The real component
   * @param im The imaginary component
   */
  public ComplexNumber(double re, double im) {
      //Initialize the instance variables
  }

  /**
   * Returns the real component.
   * @return the real
   */
  public double getReal() {
      return 0.0;  //A stub: to be fixed
  }

  /**
   * Returns the imaginary component.
   * @return the imaginary
   */
  public double getImaginary() {
      return 0.0;  //A stub: to be fixed
  }

  /**
   * Returns a new ComplexNumber number that is
   * the negative of <em>this</em>.  Note: the
   * original ComplexNumber is <b>NOT</b>
   * modified.
   * @return -this
   */
  public ComplexNumber negate() {
      return null; //A stub: to be fixed
  }

  /**
   * Returns a new ComplexNumber that is the
   * sum of <em>this</em>  and <em>z</em>.
   * Note: the  original ComplexNumber is
   * <b>NOT</b> modified.
   * @param z
   * @return this + z
   */
  public ComplexNumber add(ComplexNumber z) {
      return null;  //A stub: to be fixed
  }

  /**
   * Returns a new ComplexNumber that is the
   * difference of <em>this</em>  and <em>z</em>.
```

```
     * Note: the  original ComplexNumber is
     * <b>NOT</b> modified.
     * @param z
     * @return this + z
     */
    public ComplexNumber subtract(ComplexNumber z) {
        return null;  //A stub: to be fixed
    }

    /**
     * Returns a new ComplexNumber that is the
     * product of <em>this</em> and <em>z</em>.
     * Note: the original ComplexNumber is
     * <b>NOT</b> modified.
     * @param z
     * @return this * z
     */
    public ComplexNumber multiply(ComplexNumber z) {
        return null;  //A stub: to be fixed
    }

    /**
     * Returns a new ComplexNumber that is
     * the reciprocal of <em>this</em>.
     * Note: the original ComplexNumber is
     * <b>NOT</b> modified.
     * @return 1.0 / this
     */
    public ComplexNumber reciprocal() {
        return null;  //A stub: to be fixed
    }

    /**
     * Returns a new ComplexNumber that is
     * <em>this</em> divided by <em>z</em>.
     * Note: the original ComplexNumber is
     * <b>NOT</b> modified.
     * @param z
     * @return this / z
     */
    public ComplexNumber divide(ComplexNumber z) {
        return null;  //A stub: to be fixed
    }

    /**
     * Returns a String representation of
     * <em>this</em> in the format:
     * <pre>
     * real +-(optional) i imaginary
     * </pre>
     * If the imaginary part is zero, only the
     * real part is converted to a String.
     * A "+" or "-" is placed between the real
     * and imaginary parts depending on the
     * the sign of the imagrinary part.
     * <p>
     * Examples:
     * <pre>
     *  ..println(new ComplexNumber(0,0); -> "0.0"
```

```
     *  ..println(new ComplexNumber(1,1); -> "1.0 + i1.0"
     *  ..println(new ComplexNumber(1,-1); -> "1.0 - i1.0"
     * </pre>
     * @return the String representation.
     */
    @Override
    public String toString() {
        //DO NOT MODIFY THIS CODE!
        String str = "" + this.getReal();
        if (this.getImaginary() == 0.0) {
            return str;
        }
        if (this.getImaginary() > 0) {
            return str + " + i" + this.getImaginary();
        } else {
            return str + " - i" + -this.getImaginary();
        }
    }
  }
```

ComplexTry.java source code

You are also provided with another class, `ComplexTry`, which includes a main method that you can use to test your implementation of `ComplexNumber.java`.

The source code can be accessed more conveniently here

```
package coe318.lab2;
/**
 * Do <b>NOT</b> modify this class.
 * Its output will be used as part
 * of the marking procedure for this lab.
 * @author kclowes
 */
public class ComplexTry {
    /**
     * The output should be:
     * <pre>
     * a = 1.0 + i2.0
     * b (-a) = -1.0 - i2.0
     * c = 2.0 + i3.0
     * d (a + c) = 3.0 + i5.0
     * e (a - c) = -1.0 - i1.0
     * f (1 / a) = 0.2 - i0.4
     * g (a / c) = 0.6153846153846154 + i0.07692307692307693
     * h (a * c) = -4.0 + i7.0
     * </pre>
     * @param args Command line args not used.
     */
    public static void main(String[] args) {
        ComplexNumber a, b, c, d, e, f, g, h;
        a = new ComplexNumber(1, 2);
        System.out.println("a = " + a);
        b = a.negate();
        System.out.println("b (-a) = " + b);
        c = new ComplexNumber(2, 3);
        System.out.println("c = " + c);
```

```
        d = a.add(c);
        System.out.println("d (a + c) = " + d);
        e = a.subtract(c);
        System.out.println("e (a - c) = " + e);
        f = a.reciprocal();
        System.out.println("f (1 / a) = " + f);
        g = a.divide(c);
        System.out.println("g (a / c) = " + g);
        h = a.multiply(c);
        System.out.println("h (a * c) = " + h);
    }
}
```

Step 1: Create a Netbeans project

1. Create a Netbeans project called **`Lab2ComplexNumber`**.
2. Create a Java file (class library type) called `ComplexNumber` specifying the package as `coe318.lab2` and copy and paste the provided source code.
3. Similarly, create the Java file `ComplexTry`. Ensure that Netbeans sets the package to `coe318.lab2`.
4. Generate the javadocs and compile and run the project.
5. It should compile correctly and produce output. Unfortunately, the output is incorrect and you have to fix it.

Step 2: Add instance variables and fix constructor and getters

1. Add instance variables for the two components of a complex number.
2. Modify the constructor so that they are properly initialized.
3. Fix the `getReal()` and `getImaginary()` methods so that they return the appropriate component.
4. Compile and run your project. The statement
   `System.out.println("a = " + a)`
   should now produce the correct output.

Step 3: Fix remaining methods

1. Fix the remaining methods.
2. Suggestion: fix them in the order they are used in `ComplexTry`.
3. Hint: `subtract()` and `divide()` are easier to write by using previously fixed methods.

Step 4: Submit your lab

1. Submit your lab by zipping it to a file called `lab2.zip`
2. Then use the command `submit coe318 lab2 lab2.zip` to complete the submission.