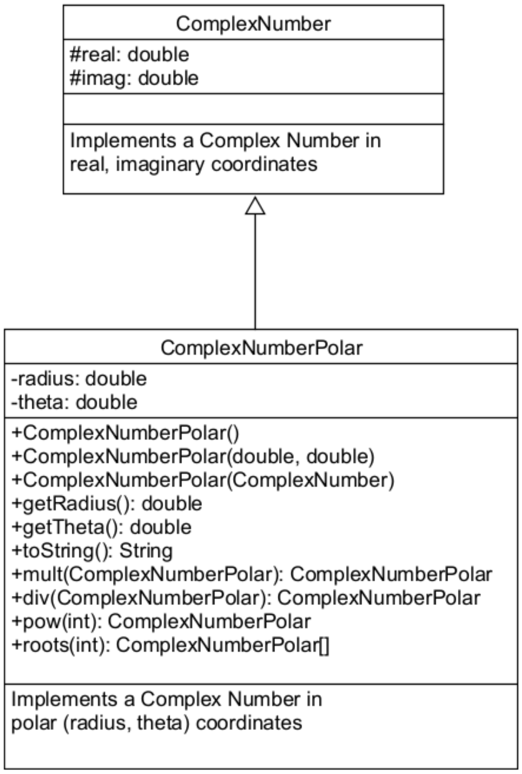
Complex Numbers – Polar Coordinates

Background

In a previous assignment you implemented a class that stores a complex number in *cartesian* coordinates using two floating point values, *real* and *imaginary*. A complex number can also be represented in *polar* form using two floating point values, *radius* and *angle*. The polar coordinate form makes multiplication and division significantly faster.

Assignment

Create a class called **`ComplexNumberPolar`** that inherits **`ComplexNumber`**. The UML Class Diagram looks as follows:


ComplexNumber
#real: double
#imag: double
Implements a Complex Number in real, imaginary coordinates
ComplexNumberPolar
-radius: double
-theta: double
+ComplexNumberPolar()
+ComplexNumberPolar(double, double)
+ComplexNumberPolar(ComplexNumber)
+getRadius(): double
+getTheta(): double
+toString(): String
+mult(ComplexNumberPolar): ComplexNumberPolar
+div(ComplexNumberPolar): ComplexNumberPolar
+pow(int): ComplexNumberPolar
+roots(int): ComplexNumberPolar[]
Implements a Complex Number in polar (radius, theta) coordinates

Function **`mult(ComplexNumberPolar rhs)`** multiplies two complex numbers in polar form, $this * rhs$

Function **`div(ComplexNumberPolar rhs)`** divides two complex numbers in polar form, $this/rhs$.

Function **`pow(int n)`** raises a complex number in polar form to the given power, $this^n$

Function **roots(int n)** computes the n^{th} complex roots (in polar form) of the complex number (in polar form), $roots(this)$

Function **toString()** returns a String of the form:

$$real + imag\ i : radius(\cos(theta) + i\sin(theta))$$

Where *real*, *imag*, *radius*, and *theta* are the values of the complex number in cartesian and polar coordinates.

e.g. **1.0 + 1.0i : 1.4142135623730951(cos(0.78539) + i sin(0.78539))**

Use the following main function to demonstrate your program

```
public static void main(String[] args) {
  ComplexNumberPolar cnp = new ComplexNumberPolar(Math.sqrt(2), Math.PI / 4.0);
  System.out.println(cnp);

  ComplexNumber cn = new ComplexNumber(1, 1);
  cnp = new ComplexNumberPolar(cn);
  System.out.println(cnp);

  ComplexNumberPolar cnp1 = new ComplexNumberPolar(cnp);
  System.out.println(cnp1);

  ComplexNumberPolar p0 = new ComplexNumberPolar(new ComplexNumber(1, 1));
  ComplexNumberPolar p1 =
                    new ComplexNumberPolar(new ComplexNumber(Math.sqrt(3), -1));

  System.out.println(p0.mult(p1));
  System.out.println(p0.div(p1));

  p0 = new ComplexNumberPolar(new ComplexNumber(0.5, 0.5));
  System.out.println(p0.pow(10));

  System.out.println("==========================");
  p0 = new ComplexNumberPolar(new ComplexNumber(-8, 0));
  ComplexNumberPolar[] roots = p0.roots(6);
  for (int k = 0; k < roots.length; ++k) {
    System.out.println(roots[k]);
  }
}
```

Deliverables:

- All source code
- Essay including
  - Screen shot of your programming running the given main function

- Self reflection describing
  - Degree of success achieved
  - Difficulties encountered
  - How you tested your code to verify correct operation

Notes:

- This is a Java only assignment
- Make member variables of your ComplexNumber class protected (not private)
- When you set radius and theta values (in the derived class), make sure you set real and imaginary values (in the base class)