package Codes;

import java.io.BufferedInputStream;

import java.io.FileInputStream;

import java.io.FilterInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.util.Objects;

public class Complex

{

public final double re; // the real part

public final double im; // the imaginary part

// create a new object with the given real and imaginary parts

public Complex(double real, double imag)

{

re = real;

im = imag;

}

// return a string representation of the invoking Complex object

public String toString()

{

if (im == 0) return re + "";

if (re == 0) return im + "i";

if (im < 0) return re + " - " + (-im) + "i";

return re + " + " + im + "i";

}

// return abs/modulus/magnitude

public double abs()

{

return Math.hypot(re, im);

}

// return angle/phase/argument, normalized to be between -pi and pi

public double phase()

{

return Math.atan2(im, re);

}

// return a new Complex object whose value is (this + b)

public Complex plus(Complex b)

{

Complex a = this; // invoking object

double real = a.re + b.re;

double imag = a.im + b.im;

return new Complex(real, imag);

}

// return a new Complex object whose value is (this - b)

public Complex minus(Complex b)

{

Complex a = this;

double real = a.re - b.re;

double imag = a.im - b.im;

return new Complex(real, imag);

}

// return a new Complex object whose value is (this \* b)

public Complex times(Complex b)

{

Complex a = this;

double real = a.re \* b.re - a.im \* b.im;

double imag = a.re \* b.im + a.im \* b.re;

return new Complex(real, imag);

}

// return a new object whose value is (this \* alpha)

public Complex scale(double alpha)

{

return new Complex(alpha \* re, alpha \* im);

}

// return a new Complex object whose value is the conjugate of this

public Complex conjugate()

{

return new Complex(re, -im);

}

// return a new Complex object whose value is the reciprocal of this

public Complex reciprocal()

{

double scale = re\*re + im\*im;

return new Complex(re / scale, -im / scale);

}

// return the real or imaginary part

public double re() { return re; }

public double im() { return im; }

// return a / b

public Complex divides(Complex b)

{

Complex a = this;

return a.times(b.reciprocal());

}

// return a new Complex object whose value is the complex exponential of this

public Complex exp()

{

return new Complex(Math.exp(re) \* Math.cos(im), Math.exp(re) \* Math.sin(im));

}

// return a new Complex object whose value is the complex sine of this

public Complex sin()

{

return new Complex(Math.sin(re) \* Math.cosh(im), Math.cos(re) \* Math.sinh(im));

}

// return a new Complex object whose value is the complex cosine of this

public Complex cos()

{

return new Complex(Math.cos(re) \* Math.cosh(im), -Math.sin(re) \* Math.sinh(im));

}

// return a new Complex object whose value is the complex tangent of this

public Complex tan()

{

return sin().divides(cos());

}

// a static version of plus

public static Complex plus(Complex a, Complex b)

{

double real = a.re + b.re;

double imag = a.im + b.im;

Complex sum = new Complex(real, imag);

return sum;

}

// See Section 3.3.

public boolean equals(Object x)

{

if (x == null) return false;

if (this.getClass() != x.getClass()) return false;

Complex that = (Complex) x;

return (this.re == that.re) && (this.im == that.im);

}

public int hashCode()

{

return Objects.hash(re, im);

}

public static void main(String[] args) throws Exception

{

//call Function

}

}