

PROGRAMMING ASSIGNMENT 1:

READ: Savitch: **Ch 4 & 5** – Defining Classes and Methods and Constructors.
Plus information given in class.

PROGRAMMING: You will create a directory called **PA1** in your cs8w home directory.
All of your PA1 work will be done in this **PA1** directory.
You will write **ComplexP1.java**
You will be given the driver program PA1.java

DUE: **Wednesday, April 17, 2002 – 11:00pm**

Turnin: **cd**
turninPA1 -- all one word

This will make sure you are in your home directory (cd) and then run the turninPA1 script which will look in your PA1 directory to search for your ComplexP1.java file that is to be turned in.

You will write a class named **ComplexP1** which defines operations for creating Complex objects (constructors), performing simple arithmetic (addition, subtraction, multiplication, and division) with Complex numbers, producing a String representation of a Complex object, accessor and mutator methods (get() and set() methods) for the real and imaginary parts of a Complex object, determining if two Complex objects are equal or not (equals() method), and determining the hash code of a Complex object (hashCode()).

You are provided the test driver program named **PA1**. **PA1.java** is located in the public/Assignments directory. You can copy it into your PA1.

Your **class ComplexP1** will define

- two **private** instance variables **real** and **imaginary** to represent the real part and the imaginary part of a complex number – both are type **double**.
- four constructors

public ComplexP1()

- no-arg constructor initializing both real and imaginary parts to 0.0.
- Recommended to use same-class constructor this(...).

public ComplexP1(double real)

- one-arg constructor with the arg representing the real part.
- imaginary part initialized to 0.0.
- Recommended to use same-class constructor this(...).

public ComplexP1(double real, double imaginary)

- two-arg constructor initializing the real part and the imaginary part with the arguments.

public ComplexP1(ComplexP1 c)

- copy constructor initializing the real and imaginary parts with the real and imaginary parts of ComplexP1 object argument.

- the toString method

public String toString()

- returns a String representation of a ComplexP1 object in the form

[real +/- imaginary"i"] – 2 places of precision for real and imaginary

for example: **[-1.23 + 17.35i]**
[43.92 - 22.17i]

- methods to perform addition and subtraction of ComplexP1 numbers

```

public ComplexP1 add( ComplexP1 c )
    - add this.real with c.real and this.imaginary with c.imaginary
    - return a new ComplexP1 object representing the result of this
      addition

public ComplexP1 sub( ComplexP1 c )
    - subtract c.real from this.real and c.imaginary from this.imaginary
    - return a new ComplexP1 object representing the result of this
      subtraction

public ComplexP1 mul( ComplexP1 c )
    - multiply this.real with c.real and this.imaginary with c.imaginary
    - return a new ComplexP1 object representing the result of this
      multiplication

public ComplexP1 div( ComplexP1 c )
    - divide this.real by c.real and this.imaginary by c.imaginary
    - return a new ComplexP1 object representing the result of this
      division

```

- accessor and mutator methods for the real and imaginary parts

```

public double getReal()          - return the real part of this object
public double getImaginary()    - return the imaginary part of this object

public ComplexP1 setReal( double r )          - return this to enable
public ComplexP1 setImaginary( double imaginary ) chaining for this PA.

```

- equals method to test equality of two ComplexP1 objects

```

public boolean equals( Object o )
    - if the argument o is null, return false.
    - if this.getClass() is not equal to o.getClass(), return false.
    - if we made it this far, then we know the object reference by o is
      a ComplexP1 object (from the getClass() check above). So if the real
      and imaginary parts of this object are equal to the real and
      imaginary parts of the object referenced by o cast to (ComplexP1),
      return true. Otherwise return false.

```

- hashCode method to generate a hash code value of this ComplexP1 object

```

public int hashCode()
    - we can use String's hashCode() as long as we have a good toString()

    this.toString().hashCode()

```

If your program does not compile with the provided PA1.java, you will receive a ZERO for the entire Programming Assignment. Your program will be tested with several different valid input values. Your program will be graded for proper comments and style as discussed in class.

A sample program is available for you to try in the public/Assignments directory.

```

cd ../../public/Assignments
java PA1

```

Turn in your assignment with **turninPA1**

You can verify your turning with **verify PA1**

Example:

java PA1

Input dialog box titled "Input" with a question mark icon. The text "Real value of Complex c1:" is displayed above a text field containing "1.23". Below the text field are "OK" and "Cancel" buttons.

Input dialog box titled "Input" with a question mark icon. The text "Imaginary value of Complex c1:" is displayed above a text field containing "3.45". Below the text field are "OK" and "Cancel" buttons.

Input dialog box titled "Input" with a question mark icon. The text "Real value of Complex c2:" is displayed above a text field containing "5.67". Below the text field are "OK" and "Cancel" buttons.

Input dialog box titled "Input" with a question mark icon. The text "Imaginary value of Complex c2:" is displayed above a text field containing "7.89". Below the text field are "OK" and "Cancel" buttons.

Output:

```
c1 = [1.23 + 3.45i]
c2 = [5.67 + 7.89i]
```

```
c3 = c1 + c2 = [6.90 + 11.34i]
c3 = c1 - c2 = [-4.44 - 4.44i]
c3 = c1 * c2 = [6.97 + 27.22i]
c3 = c1 / c2 = [0.22 + 0.44i]
```

Constructors:

```
c4 = new ComplexP1() = [0.00 + 0.00i]
c4 = new ComplexP1( 1.23 ) = [1.23 + 0.00i]
c4 = new ComplexP1( c1 ) = [1.23 + 3.45i]
```

getReal()/setReal():

```
c1.setReal( c3.getReal() )
c1 = [0.22 + 3.45i]
```

getImaginary()/setImaginary():

```
c1.setImaginary( c3.getImaginary() )
c1 = [0.22 + 0.44i]
```

Chaining test:

```
c2.setReal( c3.getReal() ).setImaginary( c3.getImaginary() )
c2 = [0.22 + 0.44i]
```

equals():

```
c1.equals( c2 ) = true
c3.equals( c4 ) = false
```

hashCode():

```
c1.hashCode() = 470268110
c2.hashCode() = 470268110
c3.hashCode() = 470268110
c4.hashCode() = 66240012
```

```
c1 = [0.22 + 0.44i]
c2 = [0.22 + 0.44i]
c3 = [0.22 + 0.44i]
c4 = [1.23 + 3.45i]
```