### COMP 1020 Lab 6

#### MATERIAL COVERED

Class Hierarchies

#### Notes:

- The three exercises are cumulative each builds on the previous one.
- Only one of the three exercises is required.
- Most students should be able to complete the Bronze and Silver exercises. The Gold exercise does not require too much additional code, but it's trickier, as usual.



### Creating a subclass

- 1. Create a **Real** class in a file **Real.java** which will represent a single real number. (This is essentially what the **Double** class already does, but we will need our own version in this lab.) Give this class:
  - a. A *private* instance variable containing the value of the number (a **double**).
  - b. A constructor which will set this value.
  - c. A **String toString()** method which will convert the value to a **String** which shows exactly 2 digits after the decimal point. [The built-in method **String.format("%4.2f",x)** will do this for any number x.]
  - d. A **double magnitude()** method which will return the magnitude (i.e. absolute value) of the number.
- 2. Create a **Complex** class in a file **Complex.java** which will be a *subclass* of the **Real** class. This will represent a complex number. [A complex number is made up of two real numbers the "real part" r and the "imaginary part" c. That's all you need to know about them here.] Give this class:
  - a. An extra private instance variable for the imaginary part of the number (a **double**)
  - b. A constructor **Complex(double r, double c)** which creates a complex number with a real part **r** and an imaginary part **c**. You will need to use the superclass constructor.
  - c. A **String toString()** method which will use the superclass's **toString()** method for the real part, then add "**±ci**" to it, where **c** is the imaginary part. The imaginary part should have exactly two digits after the decimal point, the same as the real part. [Be careful of the sign some extra code will be needed.] For example,

- d. A double **magnitude()** method, which will return the magnitude of the number. For a complex number r + ci this is defined as  $\sqrt{r^2 + c^2}$ . (To get the real part, you can use the superclass's **magnitude** method.)
- 3. Test your program using the supplied file **TestLab6Bronze.java** . The correct output is:

```
A Real number r (should print "123.46"): 123.46

A Complex number c1 (should print "3.20+6.70i"): 3.20+6.70i

A Complex number c2 (should print "3.20-6.70i"): 3.20-6.70i

Magnitude of r (should be 123.45670 ): 123.45670

Magnitude of c1 (should be 7.42496 ): 7.42496

Magnitude of c2 (should be 7.42496 ): 7.42496
```



# Creating a bigger class hierarchy

- 1. Create an *abstract* class **Number** in a file named **Number.java**. It will have no instance variables and no constructor. It should define only the two methods **String toString()** and **double magnitude()** that all subclasses of **Number** will implement, in order to allow polymorphism to be used. These are just "dummy" methods in the **Number** class, which should simply return "" or **0.0**.
- 2. Make the **Real** class a subclass of the new **Number** class. [This will make **Complex** a subsubclass of **Number**.]
- 3. Create a **Whole** class in the file **Whole.java** which will implement integers (whole numbers). [The name "Integer" is already used by the Java language, so a different name is required.] Make this class a subclass of the **Number** class, too. Like the **Real** class, this class should have a single private instance variable which holds the value of the number (an **int**), a constructor to set its value, and an implementation of the **String toString()** method and the **double magnitude()** method. (Note that the **magnitude** method returns a **double**, in order to match every other type of **Number**.)
- 4. Test your program using the supplied file **TestLab6Silver.java**. This test program will now store everything in **Number** variables, and expect polymorphism to work properly. The correct output is:

```
Number n1 is Real (should print "123.46"): 123.46
Number n2 is Complex (should print "3.20+6.70i"): 3.20+6.70i
Number n3 is Complex (should print "3.20-6.70i"): 3.20-6.70i
Number n4 is Whole (should print "13579"): 13579
Magnitude of n1 (should be 123.45670 ): 123.45670
Magnitude of n2 (should be 7.42496 ): 7.42496
Magnitude of n3 (should be 7.42496 ): 7.42496
Magnitude of n4 (should be 13579.0 ): 13579.0
```



## Casting and instanceof

1. Add one more method to the **Number** class and all of its subclasses (**Real**, **Complex**, and **Whole**) which will correctly add two numbers. Add the **Number** add(**Number** x) method to the **Number** class, and provide suitable implementations in all of the subclasses. This method should work when applied to any subclass of **Number**, and with any subclass of **Number** passed as the parameter. It should return a newly-created object representing the answer. It should use Java-style type rules to determine the type of the result: the result should always be the "bigger" type, where Complex > Real > Whole. For example, if you have

```
Number c = new Complex(3.5,4.6);
Number r = new Real(-2.4);
Number w = new Whole(8);
then
    c.add(r) should create a Complex
    r.add(c) should create a Complex
    r.add(r) should create a Real
    w.add(r) should create a Real
    w.add(w) should create a Whole
```

See the sample output below for additional examples.

[To add two complex numbers, just add the two real parts together, and add the two imaginary parts together. To add a real or integer number to a complex number, just add it to the real part and leave the imaginary part alone.]

2. Test your program using the supplied file **TestLab6Gold.java** . The correct output is:

```
Number n1 is Real (should print "-1.23"): -1.23
n1 is class Real
Number n2 is Complex (should print "3.20+6.70i"): 3.20+6.70i
n2 is class Complex
Number n3 is Complex (should print "3.20-6.70i"): 3.20-6.70i
n3 is class Complex
Number n4 is Whole (should print "-35"): -35
n4 is class Whole
Testing addition:
n1.add(n1) should be -2.46 (Real): -2.46 class Real
n1.add(n2) should be 1.97+6.70i (Complex): 1.97+6.70i class Complex
n1.add(n4) should be -36.23 (Real): -36.23 class Real
n2.add(n1) should be 1.97+6.70i (Complex): 1.97+6.70i class Complex
n2.add(n2) should be 6.40+13.40i (Complex): 6.40+13.40i class Complex
n2.add(n4) should be -31.80+6.70i (Complex): -31.80+6.70i class Complex
n4.add(n1) should be -36.23 (Real): -36.23 class Real
n4.add(n2) should be -31.80+6.70i (Complex): -31.80+6.70i class Complex
n4.add(n4) should be -70 (Whole): -70 class Whole
```