COEN 160 OO Analysis, Design and Programming Winter 2017

TA: Michael Schimpf Email: mschimpf@scu.edu

TA Office Hrs: Tuesday 1:15 pm – 2:15 pm EC605, Thursday 5:00 pm – 6:00 pm EC 618

Total Points (75)

Objectives:

- To get started with Eclipse
- To have a light introduction into Object-Oriented Programming using Java

Description: In this lab, you will start up Eclipse, create a project with packages, create classes, and fix errors. We will go through all of this together, except for fixing the errors.

Exercise 1 (30 pts)

1. Start up Eclipse JAVA.

2. Eclipse will ask you to specify a workspace. A workspace is an actual physical directory on your machine. When a workspace is selected, Eclipse will load in any projects within that directory. You may also add and edit projects inside this directory as well. For now, simply just specify a directory (**coen160labs**, for example) where you intend to keep your project files.

3. If you see the welcome screen, simply click "workbench" to continue.

4. Go to **File->New->Java Project**. Fill a name for the project (**lab1**, for example) and click "Finish". Your project should appear in the Package Explorer. If you cannot click "Finish", then click "Next" twice.

5. Within your project in the Package Explorer, you should see a "src" folder. Right click the source folder, and go to **New->Package**. A window should appear. Give the package a name **(lab1a**). Click "Finish".

6. Right click the package you just created within the Package Explorer. Go to **New->Class**. A window should appear. Give the class the name "Puppy" and make sure the Modifier is set to public. Click "Finish". A java file should appear under the package within the Package Explorer.

7. If the java file isn't open, double click the java file in the Package Explorer. Type the following code into the java file.

```
package lab1a;

public class Puppy {
	private String name;
	private int age;

	public Puppy(){
		name = "Name not given yet";
		age = 0;
	}

	public Puppy(String name, int age){
		this.name = name;
		this.age = age;
	}

	public static void main(String[] args) {
		Puppy myPuppy = new Puppy("Fido",2);
		System.out.println("Puppy name: "+myPuppy.name);
	}
}
```

a) Compile and run the class Puppy. Do you see the name? Change the puppy name from "Fido" to a name of your choice. Compile and run it.

b) Now, do step 6 from above to create a class called Tester. **Move** *(after moving, class Puppy should have no main())* the main() from class Puppy into class **Tester**.

c) Try to compile the class Tester. You will see compiler errors. The reason is you are trying to access a private data member (name) of class Puppy from another class (Tester). Private members of a class are accessible only to the other members of the same class. So we need a public accessor method, **getName()** in class Puppy, which returns the name and change it also in the main function.

Add the following method to class Puppy.

```
public String getName() {
		return name;
}
```

d) Compile and run class Tester. Did you see the correct output?

e) Now we want to display the age of the puppy as well. Add the necessary method to class Puppy. Call the method on the instance of Puppy you have created in main(). Did it work?

f) From main(), create another instance of Puppy.

```
Puppy pup2 = new Puppy();
System.out.println("Puppy name: "+pup2.getName());
```

What is the output? Name not given yet

We want to give a name to pup2. Why don't you choose a name?

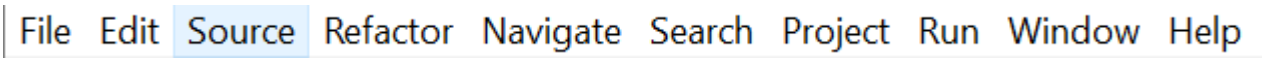
But how do we give the name, after we constructed the object with a "no-args" constructor? Similar to a getter() method we have written earlier, we need to write a setter method. Add the following method to class Puppy.

```
public void setName(String name){
	this.name = name;
}
```

Now from main(), give a name to pup2, using the setter. Make sure it is working correctly.

Since, writing setters and getters is a common task that we do when we define classes, you can let Eclipse generate them for you.

Go to **Source** and choose the option, Generate Getters and Setters.



Choose the data members for which you need the getters and setters. You are not required to provide public setters for every data member.

Exercise 2 (10 pts)

Create a new class called Cube and type the following code into it. Identify the errors, fix and compile it.

```
Public class Cube
{
   public static void main()
   {
       double height = 3.0; \ inches
       double cube-volume = height * height * height
       System.out.println("Volume = " cube_volume);
    }
```

Exercise 3 (30 + 5 pts)

In this exercise, you will learn to use **static** methods and get practice on using loops.

Part 1

Complete the class, **PatternMaker** (given below) with two static methods, **drawOneLine()** and **drawPattern().** You are given the partial class; use the comments and complete the methods.

Note: In Eclipse, create a new package ***lab1b***. Under this package, create a class called **PatternMaker** and copy the code given below. Complete the class, compile and run it.

You will test each method after defining it, before you proceed to write the next method.

Step 1: Write the class **Patternmaker** and complete the method, **drawOneLine()** as described in the comments.

Step 2: Test the method as described in **Part2 a**.

Step 3: Complete the method, **drawPattern()** as described in the comments.

Step 4: Test the method as described in **Part2 b**.

```
package lab1b;

class PatternMaker
{
     public static void drawOneLine(char symbol, int noOfTimes, int
offset ){

     // Write Java code to draw the symbol for the noOfTimes
     // after drawing a number of blankspaces (offset) .
     // For example, if the symbol is "*", noOfTimes is 10
     // and offset is 3,it should draw 3 blank spaces and
     // then draw 10 stars.

     System.out.print();   // This will print on the same line
     System.out.println(); // This will start the drawing on a new
line.

     // Write a for - loop to draw offset number of blank spaces.
     // for example, if offset is 3, it should draw 3 blank spaces.
     // Use System.out.print(" ") to print a single blank space.
     // Write a for - loop to draw the symbol, the noOfTimes.
}

public static void drawPattern(){
     /* This method should draw the following pattern, where the first
     line, should have 5 blank spaces followed by 5 stars, second line
     will have 10 blank spaces followed by 10 stars and the last line
     will have 15 blank spaces followed by 15 stars.

  *****
    *********
      *************

     The method should call the method, drawOneLine() with the correct
     number of values for parameters.
     For example, the first call to drawOneLine() is  drawOneLine
     ('*', 5, 5); */
}

public static void main (String [] args){

}
}
```

Part 2 – Testing

a. Go to the main() in PatternMaker class, and add the following statement:

```
PatternMaker.drawOneLine('+',12,6);
```

Compile the class PatternMaker and run it.

b. Go to the main() in PatternMaker class, and add the following statement:

```
PatternMaker.drawPattern();
```

Compile the class PatternMaker and run it.

Question (5 pts)

When you tested your program in Part2, you did not create any instances (objects) of **PatternMaker** using *new* (similar to how you created instances in class Puppy), to call the methods on.

Why did they work without instances?

We didn't need an instance of the class PatternMaker because we were writing a function to be called within the class