**KV4001 Week 3 Inheritance Lab**

This lab is to allow you to explore the implementation of inheritance in Java. It will cover inheritance basics, access issues, constructors, creating multilevel inheritance hierarchies. It is heavily based on the material covered in the first part of the lecture.

The exercises in this lab involve answering some questions and writing some code. The questions require only very brief answers, so there's no need to create additional files. Just write the answers as comments in the code.

It is assumed that you are using BlueJ as you IDE. If you are not using BlueJ then make appropriate adjustments to the tasks. This will most probably mean you will need to write driver classes.

**Before the Lab**

You should read through the lecture notes and also chapter 8 the set text. In addition, you should read the material covering inheritance in the Java Tutorial at:

<https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html>

**Basic Inheritance Exercises**

Java supports inheritance by allowing one class to include another class in its declaration. This is done using the keyword ***extends***. The use of the word extends provides a useful clue to what is happening: the subclass is extending the superclass is some manner.

Our starting point will be to consider a simple example based on geometric shapes. Create a new project called Inheritance1. Place a copy of the class ***TwoDShapeForLab.java*** from Blackboard in the project. Make sure you understand the code by checking it out in BlueJ. A copy of the class is shown in figure 2.

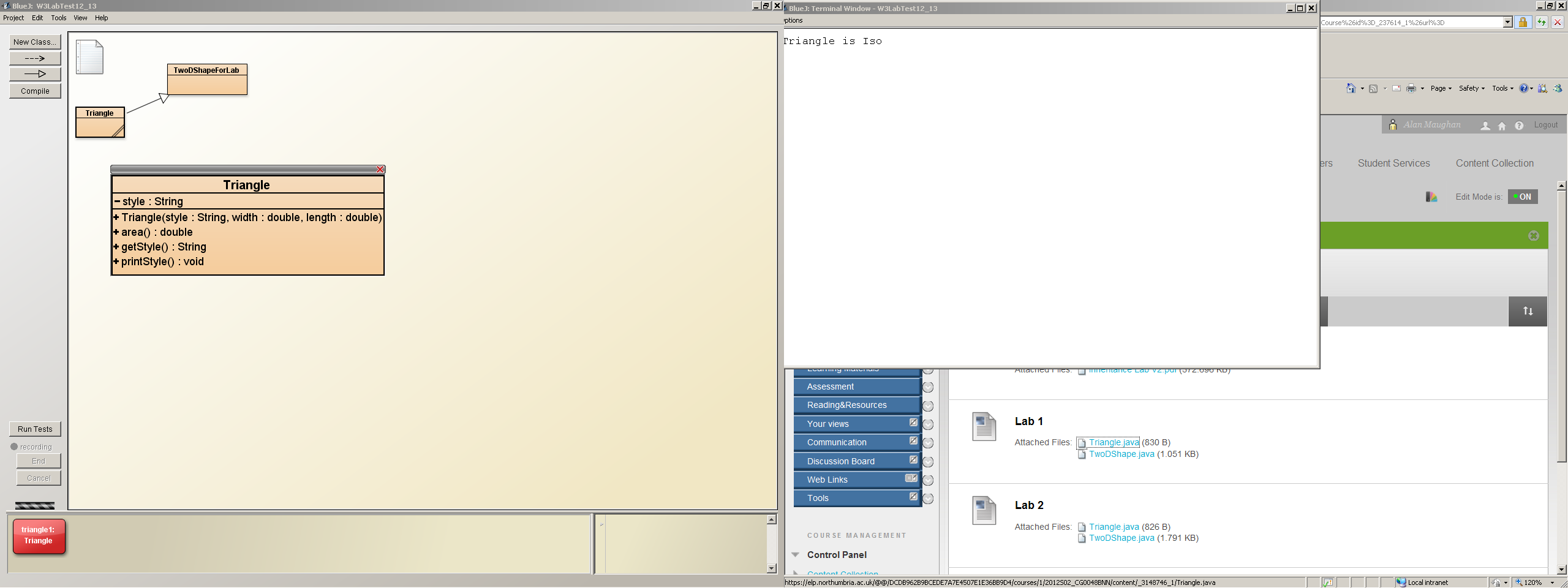
**Task 1**

Write a small class Triangle.java to extend TwoDShapeForLab. This class should have a new attribute that holds a String defining the type of triangle, i.e. Scalene, Isosceles, or Equilateral. You do not need to provide any error checking for this attribute. The class will have three methods:

*area* which returns the area of the triangle using the formula w \* l /2.0

*printStyle* prints the style string on the console window

*getStyle* which will return the style String.



*Figure 1*

Once you have written the code and successfully compiled it instantiate a Triangle object in the BlueJ

object bench. Click on the object and you will see the methods you have defined in Triangle listed. In addition, there should be an entry *Inherited from TwoDShapeForLab*. If you follow that link you should see the list of public methods available in TwoDShapeForLab.

Experiment by invoking the methods in Triangle and those inherited from **TwoDShapeForLab**.

public class **TwoDShapeForLab**

{

private double width;

private double length;

public **TwoDShapeForLab**(double width, double length)

{

this.width = width;

this.length = length;

}

public double **getWidth**()

{

return width;

}

public double **getLength()**

{

return length;

}

public String **toString()**

{

return Width is " + width + " Length is " + length;

}

}

*Figure 2*

**Task 2**

In both TwoDShapeForLab and Triangle we have provided constructors with parameters. In the Triangle constructor we should also have made use of ‘super’ to call the constructor of TwoDShapeForLab. If not, do so now.

In the lecture we noted that when using inheritance there are various rules about the provision of constructors, in particular the default constructor. If you provide a constructor with parameters the Java Virtual Machine (JVM) will not provide you with a default constructor (a basic no-arg constructor).

To see what happens, comment out the call super(width, length) from your Triangle constructor and compile.

Now modify TwoDShapeForLab by adding the following constructor:

public TwoDShapeForLab()

{

this.width = 0.0;

this.length = 0.0;

}

Recompile the project. Because there is now a no-arg constructor the previous syntax error should have vanished.

Move the call to super in Triangle from the first line of the constructor to the last. Re-compile. What happens? Return it to the first line of the constructor.

Modify TwoDShapeForLab by adding setter methods for width and length.

**Task 3**

Strictly this is nothing to do with inheritance but it does demonstrate the use of the ***static*** qualifier. In TwoDShapeForLab add the following lines of code:

private static int shapeCount ; // with the attribute declarations shapeCount++; // in each constructor

And add the following method:

public void printShapeCount()

{

System.out.println("Number of shapes created is " + shapeCount);

}

Instantiate several Triangle objects and examine the value of shapeCount. You should also inspect the class TwoDShapeForLab where you should see the value of shapeCount. Remove the objects. What happens to the value of shapeCount?

**Some Questions:**

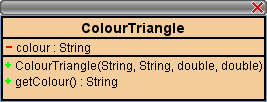
1. How does a subclass call its superclass constructor?

2. Can a parameter be passed via super()?

3. Can super () go anywhere in the subclass constructor?

**Task 4**

In this exercise you are going to build a multilevel inheritance hierarchy. We already have two parts of the hierarchy – TwoDShapeForLab and Triangle. Your task now is to extend Triangle and produce a class ColourTriangle with a new attribute called colour.



*Figure 3*

Create an instance of a ColourTriangle and experiment call in the inherited methods to confirm they are still behaving correctly.

**Question**

What order are the constructors called? Confirm your answer by adding System.out.println() calls to each constructor in your hierarchy.

**Task 5**

We have a toString method in TwoDShapeForLab. This returns a string of the form:

"Width is " + width + " length is " + length

Add a toString method to Triangle that will return a string of the form:

"Width is " + width + " length is " + length

“Style is “ + style

Similarly for ColourTriangle add a toString method to return a string of the form

"Width is " + width + " length is " + length

“Style is “ + style

“Colour is “ + colour