Lab9: Improving Structure with Inheritance

Answer each question in a different directory, named Lab9-[number]. Make sure that you include all the .java files we need in order to run your code. Unless otherwise specified the class that contains the main method should be named Lab9[number]Test We should be able to compile and run your code on LCPU (e.g. by typing `javac *.java` and `java Lab9[number]Test`).

All exercise directories should have ¡username¿-Lab9 as their parent directory. This folder should be compressed into a single file prior to submission. File submissions to moodle must be a zip or tar archive, the file name should be of this form:

`<username>- lab<number>.tar.gz` **or** `.zip`

For example: `ddk20-Lab9.zip` **or** `ddk20-Lab9.tar.gz`

There are often various ways of solving an exercise and some parts are left intentially vague. In those cases, you can decide how you code the exercise.

If you have questions about this lab sheet or are stuck with one of the questions talk to the tutors in the lab or post your question on Moodle. If your question needs to include code, please email it to the mailing list rather than posting it to Moodle.

This lab is part of your lab coursework. **The deadline for submission is 10am Tuesday 11 December 2012**

The code associated with the this lab sheet can be downloaded from Moodle.

Exercises 1. Design and implement an inheritance hierarchy for the classes Shape, Triangle, Square and Rectangle. The classes should (at least) contain a method to calculate the surface area of the shape and a toString() method. The driver program (which includes the main function) should create one of each of the shapes and store them in a collection. A loop should then be used to print out the details of the shape (dimensions and surface area) making use of polymorphism. Demonstrate in your test class that your code is working properly.

20 marks

Exercises 2. Continuation of Shapes hierarchy.

1. Add a 'number of sides' field to your Triangle and Rectangle classes, and include the necessary accessor methods. Explain in the documentation of your main class, why only these two classes have been chosen to contain this additional coding?

2. Update your toString methods so they also print the number of sides of each shape created.

3. Add a print statement to the constructor of your shape class that prints out the details of the shape objects as they are created. What do you notice at runtime? Why does this occur?

Demonstrate your main class that your code works properly.

20 marks

Exercises 3. Complete the simulation as explained on the last three slides of the lecture handout (available on Moodle) Demonstrate that your code works correctly in the main method by creating a Workstation "Bill Gates", PrintServer "Apple", Nodes "A","B", "C" and a Fileserver "silicon". Connect them together in a LAN. Put the following Packets on the LAN:

- Destination: "Silicon" Content "windows"
- Destination: "Apple" Content "Mac"
- Destination: "A" Content "B"
- Destination: "D" Content "wrong"

20 marks

Exercises 4. Extend your LAN simulation with two subclasses of Packet: ASCIIPacket and PostscriptPacket. When creating an ASCIIPacket the contents will be altered by adding the string length and "ascii:" to the front of the string (e.g. "contents" becomes ascci:contents). For PostscriptPackets we alter the content by adding '@@' after each character. Demonstrate that your code is working correctly by extending the example LAN created earlier.

20 marks

Exercises 5. Expand your LAN simulation further with an ASCIIPrintserver and a PostscriptPrintserver. An ASCIIPrintserver should only be capable of handling ASCIIPackets. The PostscriptPrintserver can handle both. Make sure that both printers display the original contents of your package. As before demonstrate that your code is working correctly by extending the example LAN and demonstrating all the features of your implementation.

20 marks

Note: you might find the following code useful

```
import java.io.*; //input and output classes

public class Writing
{
```

```
    public static void main(String[] args){

        /**
        * Do not worry about understanding try or catch.
          These will be covered in a later lecture..
        */
        try {

            FileWriter writer = new FileWriter("m.txt",true);
            writer.write("Marina\n");
            writer.close();
            BufferedReader reader =
                new BufferedReader(new FileReader("c://m.txt"));
            String line = reader.readLine();
            System.out.println(line);
            reader.close();
        }
        catch(IOException e) {

            System.out.println("nn");
        }
    }
}
```