Advanced Programming 2015 – Year 2
Labwork 6: (6% - or <u>60 points out of 500 points</u> for labwork this semester)

NOTE: ALL LABS TO BE COMPLETED IN PROJECTS USING ECLIPSE

NOTE: YOU <u>MUST</u> USE YOUR OWN EXAMPLES FOR THESE EXERCISES, I.E., YOU CANNOT RE-USE LECTURE EXAMPLE(S) AS SUBMISSIONS (THESE WILL BE GIVEN ZERO MARKS\POINTS)

<u>REVISION EXERCISE 2</u> – (10 points) – THERE WILL BE 5 x 10 MARKS GOING TOWARD REVISION ACTIVITIES IN THE NEXT 5 LABS

Create a project called **Revision2**. Create a very simple **JFrame** class called **DisplayImageFromJar** that will create a JFrame with only a JLabel contained inside it. The JFrame using the JLabel must load and display an image of your choice by calling a method called **displayImage(String imageName)**. Fully Javadoc the class. Fully Jar the class. Test the image display GUI using the Jar file.

Required activities and marking guideline:

• Create JFrame with JLabel	(2 points)
• Write displayImage method to set the image to display	(3 points)
• Fully Javadoc the class and method (use tags)	(2 points)
• Jar and execute the jar to display image (include Javadoc in Jar)	(3 points)

Part 1 – Utility Class (10 points)

Create an Eclipse Project called **Lab6Part1**. Create a utility class called **ShapeCalculations** with three **static** methods called **double getVolumeOfCylinder(),double getVolumeOfCube(), double getAreaOfSphere().** Include a **final static** variable called PI that holds a permanent value for pi, e.g., 3.14159265. Write a second program that tests all of the static methods available in the ShapeCalculations utility class and prints out the value of PI to the default output device.

Required activities and marking guideline:

• Final static variable PI declared and initialized	(2 points)
• Implement three calculation methods (use PI, 2 marks each)	(6 points)
• Test methods using sample data	(1 point)
• Output PI to default output using ShapeCalculations class	(1 point)

Part 2 – Object Orientated Programming (10 points)

Create an Eclipse Project called **Lab6Part2**. Create an **abstraction** of a **Computer** using a Java class with attributes for the computer speed (in GHz), memory (in GB) and screen size (in inches). Add an appropriate constructor for the Computer objects. Create a second class called **ComputerTest** that creates at least three Computer objects and prints each objects' details to the default output device (System.out).

Required activities and marking guideline:

- Computer class with attributes (4 points)
- Computer constructor (with attributes passed) (3 points)
- Computer test class with at least three objects of Computer (3 points)

Part 3 – Inheritance in object oriented programming (10 points)

Create an Eclipse Project called **Lab6Part3**. Create a subclass of the Computer class in **Part2** above called **LaptopComputer** that inherits directly from the Computer class but has an extra specialized attribute for **battery life**. Add a constructor to the **LaptopComputer** that will re-use the constructor from the superclass (using keyword **super**) and that will also set the new specialized attribute battery life. Create a second class called **ComputerInheritanceTest** that create at least two regular Computer objects and two specialized LaptopComputer objects (reference all objects using the superclass reference type Computer, e.g., Computer comp1, comp2 etc.).

Required activities and marking guideline:

- Implement inherited subclass LaptopComputer with new attribute (4 points)
- Write LaptopComputer constructor re-using **super** constructor (3 points)
- Test class for objects using only **Computer** reference types (3 points)

Part 4 – Vector of custom objects (complete hierarchy) (20 points)

Create an Eclipse Project called **Lab6Part4**. Create a complete hierarchy of electronic devices from a generic class **ElectronicDevice** (the most generic) down to specific devices, e.g., **WalkieTalkie**. Make the classes that are not specific enough to instantiate **abstract** classes. Create a test class **TestFullHierarchy** that creates a Vector of ElectronicDevice types and adds at least one object of <u>each</u> non-abstract type to the Vector. Print the enter Vector list to the default output device (System.out, Note: overriding toString() and recalling super.toString() within the sub-class toString() method can make output easier). Each non-abstract (concrete) sub-class must provide a specialized constructor that re-uses the super class constructor. Create the following inter-related classes:

- An abstract ElectronicDevice class with attribute 'manufacturer'
- An abstract HandHeldDevice sub-class of ElectronicDevice with specialized attribute 'weight'
- A Computer <u>sub-class of ElectronicDevice</u> class with attribute as per Part2 above
- A LaptopComputer sub-class of Computer as per Part3 above
- A WalkieTalkie sub-class of HandHeldDevice with the specific attribute 'rangeInKm'
- A MobilePhone sub-class of HandHeldDevice with the specific attribute 'networkName'
- A **TestFullHierarchy** that create objects of type reference ElectronicDevice (at least one per sub-class type) and stores all objects in a Vector

Required activities and marking guideline:

- ElectronicDevice abstract class plus constructor (3 points)
- HandHeldDevice abstract class plus constructor (3 points)
- Computer and Laptop classes (re-used\modify for new hierarchy) (2 points)
- WalkieTalkie subclass with constructor (re-use super) (3 points)
- MobilePhone subclass with constructor (re-use super) (3 points)
- TestFullHierarchy create at least one of each sub-type (2 points)
- TestFullHierarchy Vector stores all objects (2 points)
- Print all details of all objects in Vector (2 points)