Advanced Programming 2021 – Year 2
Labwork 6: (This lab is worth 60 points out of 500 points)

NOTE: ALL LABS TO BE COMPLETED IN PROJECTS USING ECLIPSE OR EQUIVALENT IDE (NO MORE TEXTPAD, EVER EVER!!!)

IMPORTANT NOTES:

- **NO COPYING PERMITTED AND ZERO MARKS WILL APPLY TO COPIED WORK. FURTHER ACTION MAY BE TAKEN AGAINST STUDENTS THAT HAVE BEEN FOUND TO COPY WORK.**

- **ASSESSMENT WILL INVOLVE ONE-TO-ONE QUESTIONS ABOUT YOUR SUBMITTED WORK. USE COMMENTS IN YOUR CODE TO ENSURE YOU DON'T FORGET WHY YOU WROTE CODE YOU MAY LATER BE ASKED ABOUT.**

- **ALL WORK MUST BE SUBMITTED TO MOODLE BY DATES SPECIFIED (2 LABS SUBMISSIONS OF FIVE LABS THROUGHOUT THE SEMESTER).**

- **MANY OF THE TASKS ASSIGNED BELOW CAN BE COMPLEX AND\OR THE DESCRIPTIONS MAY REQUIRE FURTHER CLARIFICATIONS. PLEASE USE THE AVAILABLE LAB TIMES TO ASK FOR CLARIFICATIONS AND ADVICE\HINTS ON THE TASKS BELOW.**

Part 1 – Utility Class **(10 points)**

Create an Eclipse Project called **Lab6Part1**. Create a utility class called **VolumeCalculations** with three **static** methods to calculate the volume of THREE common shapes. One of the shape volume formulae you choose MUST use pi (π). Include a **final static** variable called **PI** that holds a permanent value for pi, e.g., 3.14159. Write a second program that tests all of the static methods available in the VolumeCalculations 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 **(15 points)**

Create an Eclipse Project called **Lab6Part2**. Create an **abstraction** of a **Computer** using a Java class. Include THREE attributes of computers in the class (you may choose the attributes), e.g., processor speed, memory and screen size. Add an appropriate constructor for the Computer objects. Include a blank constructor in the class so that you can create Computer objects without parameters. 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 (5 points)
- Blank computer constructor (3 points)
- Computer constructor (with attributes passed) (4 points)
- Computer test class with at least three objects of Computer (3 points)

Part 3 – Inheritance in object oriented programming (15 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. Include a blank constructor in the class so that you can create Laptop objects without parameters. Create a second class called **ComputerInheritanceTest** that create at least THREE regular Computer objects and THREE specialized LaptopComputer objects (reference all objects using ONLY the superclass reference type Computer, e.g., Computer comp1 = new LaptopComputer(3), Computer comp2=new Computer() etc.).

Required activities and marking guideline:

- Implement inherited subclass LaptopComputer with new attribute (5 points)
- Blank computer constructor (2 points)
- Write LaptopComputer constructor re-using **super** constructor (3 points)
- Test class for objects using only **Computer** reference types (5 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 each non-abstract type to the Vector. Print the enter Vector list to the default output device (System.out, Note: You can research how to override 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' (Note: Include a blank constructor to avoid later compile errors with Computer)
- An abstract HandHeldDevice sub-class of ElectronicDevice with specialized attribute 'weight'
- A Computer sub-class of ElectronicDevice 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 (and blank 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)