Advanced Programming 2025 – Year 2
Labwork 1: (5% - or 50 points out of 300 points for labwork this semester)

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

TOPICS INCLUDE: IDE's, Jarfile, Javadoc, Packages, Modules, Recursion

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. A COMPLETED SELF-ASSESSMENT SHEET WILL BE USED TO GUIDE THE ASSESSMENT. 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 (SUBMISSION DEADLINES WILL BE POSTED ON MOODLE).**

- **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.**

- **YOU MUST USE AN IDE TO COMPLETE TASKS (e.g., Eclipse or IntelliJ or NetBeans or similar). Support and help provided for Eclipse primarily.**

- **CHATGPT and other similar AI tools that can code simple solutions are NOT PERMITTED. THEY DO NOT TEACH YOU HOW TO BECOME A GOOD PROGRAMMER EITHER!**

Part 1 – Creating, running and jar'ing your first IDE project (10 points)

Using an IDE create a class called **Lab1Part1.java** (this should be done within a project for Eclipse, e.g., a project called **Lab1Part1** - or equivalent for your chosen IDE). Add this class to a **package** called *firstPackage*. In the class output your name and student number to the default output device using System.out.print. Add a static method called **getMyNameFromMyInitial** to the class that takes an initial (e.g., LR for Luke Raeside) of your name as input and **returns** your full name as a String **ONLY IF** you pass the correct initial. If the incorrect initial is passed return the String "INPUT INITIAL NOT CORRECT". Test the method using the main and outputting the result of the method call. Create a **Jarfile** for this project called **Lab1Part1.jar** and run the project from the jarfile. **Javadoc** the static method with appropriate Javadoc tags and generate the Javadoc (@return and @param).

Required activities and marking guideline:

- Create the project and class and package (1 point)
- Put the output statement in the main method (1 point)
- Write and test the method (must have a return statement) (2 points)
- Add Javadoc to method with appropriate tags (@param etc) (2 points)
- Generate the Javadoc pages (and review) (1 point)
- Test and run the project\class, including calling the method (1 point)
- Create the jarfile for this project (1 point)
- Run the project from the jarfile (1 point)

Part 2 – Multiple classes within THE SAME project (10 points)

Create a second class within the **same project** from part 1 above. Name this class **Lab1Part2.java** and put this class in a package called *secondPackage*. Make sure **Lab1Part1.java** is in this project in it's package *firstPackage*. Add another method to the **Lab1Part1** class called **getMyInitialFromMyName** that does exactly the opposite as the previous method (pass your full name and return the initial, e.g., pass John Doe and return JD). Call the **getMyNameFromMyInitial** and the **getMyInitialFromMyName** method found in the **Lab1Part1.java** class from the new class called Lab1Part2 (you can call the methods from within the main method of Lab1Part2). Test the calling of the methods from this new class by outputting the results of the method call using System.out.print. Javadoc all of the classes and methods in the project and generate the Javadoc. Create a **jarfile** called **Lab1Part2.jar** of all of the classes together and run it.

- Create the new class within the same project with a main (2 points)
- Call the method found in the other class from this new class (1 point)
- Create the new method to return initials from the name passed (2 points)
- Output the result of calling the method call using System.out (1 point)
- Write and generate the Javadoc (classes and methods) (2 points)
- Test and run this new class within this same project (1 point)
- Create the jarfile and run the project from the jar (1 point)

Note: These first two parts will bring up a few issues\learning points. Firstly, how to call a **static method from a main method (two aspects: from the same class and from another class: for the latter precede the method name with the class name). If the methods are not static you won't be able to call them from main unless you create an object. Also, if the **access modifier** is not correctly set you will not be able to call the methods in the first part from the second part (i.e., not private). Also, you may have to configure the main class within the project to run the correct class.*

Part 3 Putting it all together (Jars, Packages, Javadoc IDE) (15 points)

Create a new Eclipse Project called **Lab1Part3**. Put in a package called *thirdPackage*. Create a simple JFrame GUI called Lab1Part3 with ONE JTextField in the center and ONE JButton at the bottom. Use a JLabel beside the field to invite the user to enter an initial into the textfield, e.g., "Input initial here: ".

Add a second label to the center of the frame with the text "Result of method call will appear here". Set the text of the button to "Click here to retrieve student name from initials".

Import the jarfile from part 1 above (Lab1Part1.jar). Now import this jarfile into this new project (In Eclipse: Project → Build Path → Configure Build Path. Under Libraries tab, click Classpath folder and click Add Jars or Add External JARs and point to the jarfiles to import the jarfile called Lab1Part1.jar you have created using the code in part 1 above: this link that follows provides full instructions for importing jars https://stackoverflow.com/questions/3280353/how-to-import-a-jar-in-eclipse).

Once the jar is imported into this project you will be able to re-use the method you wrote in the previous part of this labwork in part1 above, i.e., **getMyNameFromMyInitial** method**. DO NOT COPY LAB1PART1 IN TO THE PROJECT, IMPORT IT!!!.** You will also be able to see the Javadoc from Lab1Part1.

Use event handling to capture the initial entered into the field within the GUI when the button is pushed and call the **getMyNameFromMyInitial** method from the event handler to check if the student initial matches. Output the String result of the method call to the GUI label created above (the full name or the error message).

In each case output the result using the output label in the frame. Jar and Javadoc this project (including the imported classes) and run from the jarfile.

Required activities and marking guideline:

- Create GUI the labels, fields and buttons (within frame, look neat!) (3 points)
- Create and add class to package (using keyword package) (2 points)
- Create and Import the jarfile to the new project (Lab1Part1.jar) (2 points)
- Add listeners and handlers (and code the responses, re-use part1) (2 points)
- Javadoc all of this new project and generate the Javadoc (2 points)
- Create and run the jarfile from this new project (2 points)
- Test the GUI (you should see the method call result in the Label) (2 points)

Part 4 – Recursion and Eclipse projects (15 points)

Create an Eclipse Project that contains a class called **Lab1Part4** within that project. Put this class in a **package** called *recursive*. Create a method inside the class called **recursiveSum(int)** that will find the total (add together) all the numbers from the number passed as parameter down to 1, e.g., if you pass the value 10 the method MUST use recursion (NO RECURSION = NO MARKS) to calculate 10+9+8+7+6+5+4+3+2+1 and return the result (Note: this is a non-void method that returns a result).

Create a second recursive method within this class called **buildAlphabetString(char)** to return an alphabet string from a char you pass to the method up to the small char 'z', e.g., you call the method, for example, using **buildAlphabetString('a')** and a String is returned with the entire alphabet up to and including z. This **recursive** method MUST BE DONE RECUSIVELY (NO RECURSION = NO MARKS) AND YOU MUST EXPLAIN HOW IT WORKS TOO!!!

Add **javadoc** comments to the above methods to describe the function of the methods and method parameters (@param and @return tags MUST be used at a minimum). Generate the **javadoc** documents for the project. Finally, Jar the project in a **jarfile** called **Lab1Part4.jar** and test the execution of the **jarfile**.

Required activities and marking guideline:

- Create the Eclipse project and class (1 point)
- Create and add class package (1 point)
- Write recursiveSum method (4 points)
- Write buildAlphabetString method (with parameter) (5 points)
- Add and generate FULL set of Javadoc with tags (2 points)
- Jar and execute the program from the jar (2 points)