Advanced Programming 2021 – Year 2
Labwork 3: (5% - or 50 points out of 500 points for labwork this semester)

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 – Try-with-resources (5 points)

Create an Eclipse Project called **Lab3Part1**. Add a text file to the project directory called *myFavTeam.txt* which contains only one line with the name of your favorite team (pick Manchester United or something if you have no favorite team). Create a File variable called **teamFile** that references the *myFavTeam.txt* file (simply pass the name of the file to the File constructor). Write a program to read your team from the **teamFile** using the **BufferedReader** class and corresponding **readLine()** method. **Use the *try-with-resources* approach** to read the file and output the contents of the file to the default output device (System.out). [NOTE: You must use try-with-resources approach to receive marks for this exercise – see the lecture notes!!]

- Create file (2 points)
- Use try-with-resources to read your name from file (3 points)

Part 2 – Retry an action in catch block (15 points)

Create an Eclipse Project called **Lab3Part2**. Create a simple JFrame GUI with a JButton called **inputButton.** Create an array in your program called **stringArray** with the contents defined as {"A","B","C"}. Create and add a **JLabel** with the String "Please enter the index of the array to output: ". Use a **JTextfield** called **inputField** to input the index number of the array requested by the user. Create and add a **JLabel** called **outputLabel** to output the contents of the array at the index requested by the user, e.g., if the user enters '0' output "A" to the output label, e.g., outputLabel.setText(array[0]). Add the listeners necessary to get the index and display the contents of the array and display in the output field. Place a **try..catch** block around the code to output the contents of the array: catch an **ArrayIndexOutOfBoundsException**. If this exception is caught (requesting beyond the array size!!) use an option pane **input dialog** to give the user a second chance to try and output within the array bounds and show the output in the **outputLabel**, e.g., message via the input dialog "You attempted to access beyond the limits of the array, please try again". If the user fails to keep within limits again you do not have to deal with that! Jar the project and test the running of the project from the jarfile. Javadoc the project.

Required activities and marking guideline:

- GUI created (3 points)
- Listeners working (3 points)
- actionPerformed with try and catch (4 points)
- Retry works with input dialog (3 points)
- Javadoc (1 point)
- Jar and run the GUI from the Jar (1 point)

Part 3 – Write a custom exception, throw it and handle it (15 points)

Create an Eclipse Project called **Lab3Part3**. Write a class called **MyMobileNetworkChecker** that will attempt to verify the network of your mobile phone and throw an exception using exception handling if the make entered does not match. Create a custom exception class called **NotMyNetworkException**.
Write a static method within the **MyMobileNetworkChecker** class called **checkMyMobileNetwork(String inputNetwork)** that declares **(throws)** the custom **NotMyNetworkException** exception. Implement the checkMyMobileNetwork method so that it matches the string entered with your network name (e.g. Vodafone, Three etc.), if it succeeds output a message to verify the network passed is correct, if the string passed is checked and it is NOT the same as your make of phone throw a NotMyNetworkException using the keyword **throw**. Test the calling of the checkMyMobileNetwork method in the main (test the passing of the incorrect phone make to the method so that the exception is thrown). Fully Javadoc the project including the method. Jar the project and test the running of the project from the jarfile.

Required activities and marking guideline:

• Implement the custom exception (extends Exception)	(3 points)
• Implement the check make method with throws keyword	(4 points)
• Handle the custom exception in the main	(3 points)
• Test the custom exception is thrown\caught	(3 points)
• Javadoc the class (especially the method)	(1 point)
• Jar the project and run the Jarfile	(1 point)

Part 4 Add a new exception to an existing system (15 points)

Create an Eclipse Project called **Lab3Part4**. Import the com.raeside.family and com.raeside.exceptions packages into this project (available on Moodle). As explained in the lecture this system is meant to ensure that all members added to a family have the same surname and that no two people in the same family can share a first name; this should be accomplished using exception handling. Currently the system works for the first name exception but not for the surname exception (your job is to fix this so that both exceptions are handled). Run and test this program (run the **MakeFamilyRobinson** class). Add a new exception class to the existing system called **SurnameMismatchException**. Add some meaningful message to the exception e.g. "You must have surname Robinson to join this family". Declare the **addFamilyMember()** method to throw SurnameMismatchException as well as the **FirstNameExistsException**. You will need to use the **correctFamilyName()** method to check that the Person being added to the Family has the correct surname, throw the new SurnameMismatchException from the **addFamilyMember** method (the addFamilyMember method already throws an exception but it is possible to throw more than one exception from the same method using comma to separate each exception thrown). Finally HANDLE the **SurnameMismatchException** exception in the **MakeFamilyRobinson** class in addition to handling the **FirstNameExistsException** when attempting to add a family new member using the **MakeFamilyRobinson** class. Test the new exception class is working by attempting to add a Person called "Jessy James" to the Robinson family (i.e. the correct **SurnameMismatchException** should be thrown, note: the **FirstNameExistsException** should still work if the first name of the Person being added is already in use in the family, but only one of the exceptions will be thrown at any one time). Test the **FirstNameExistsException** still works by attempting to add another new Person to the family whose first name would be the same as an existing family member but that has the correct surname, e.g., "Paul Robinson" (where there is a Paul already in the family).

Required activities and marking guideline:

- Create new custom exception SurnameMismatchException (3 points)
- Add new throw to addFamilyMember method (surname) (3 points)
- Add logic to throw the SurnameMismatchException exception (3 points)
- Create a new Person object and add to the family to test the addition of a member without correct family name (3 points)
- Create a new Person object and add Test the attempted addition of a family member with a first name that is already in use (3 points)