Advanced Programming 2015 – 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

NOTE: YOU MUST 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)

Part 1 – Nested try-catch statement (5 points)

Create an Eclipse Project called **Lab3Part1**. Demonstrate using a Java class how you can nest a try-catch statement inside another try-catch statement (place a try-catch inside the catch of another outer try-catch. Use any exception classes you wish in this example\demonstration.

- Outer try catch statement in a class (2 points)
- Inner try catch nested statement in the outer catch above (3 points)

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

Create an Eclipse Project called **Lab3Part2**. Create a simple JFrame GUI with only one JButton called **stringLengthButton** and one JTextField called **inputField** and one JLabel called **outputLabel**. Set the JButton text to "Get String Length". Set the JLabel text to "String length = ". Implement the actionPerformed method for the **stringLengthButton** so that when the button is pushed the length of a string entered into the **inputField** (JTextField) is displayed, e.g., if "Book" is entered into the **inputField**, the **outputLabel** displays "String length = 4". However, this program **MUST** deal with the entry of no String (or **null** string) into the **inputField** by adding a try catch block to the actionPerformed method. The try will attempt to get the length of the String entered and output the length to **outputLabel**. Implement a **NullPointerException** catch statement which will provide the user with an input dialog (use JOptionPane) to reattempt the entry of the string and set the **outputLabel**. Test the GUI by entering no string into the **inputField** and test the retry option of the catch statement using the input dialog. Fully Javadoc the project (including the method(s)). Jar the project and test the running of the project from the jarfile.

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 **MyMobilePhoneMakeChecker** that will attempt to verify the make of your mobile phone and throw an exception using exception handling if the make entered does not match. Create a custom exception class called **NotMyPhoneMakeException**. Write a static method within the class called **checkMyMobilePhoneMake(String inputPhoneMake)** that declares **throws** the custom **NotMyPhoneMakeException** exception. Implement the checkMyMobilePhoneMake method so that it matches the string entered with your make of phone, if it succeeds output a message to verify the phone make passed is correct, if the string passed is checked and it is NOT the same as your make of phone throw a NotMyPhoneMakeException using the keyword **throw**. Test the calling of the checkMyMobilePhoneMake 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)