**Advanced Programming 2017 – Year 2**

**Labwork 9: (6% - or 60 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 – Using package keyword and Javadoc (10 points)**

Create a project called **Lab9Part1**. Create a class called **StringUtility**. Place the **StringUtility** class in a package called **com.yourSurname.util** (where “yourSurname” is your actual surname, e.g., “raeside”). Add **two static methods** to the utility class called **getSumOfAcsiiValues(String s)** and **getProductOfAsciiValues(String s)**. The getSumOfAcsiiValues(String s) method will take each ascii value of a string passed and compute the sum of the ascii values (get each ascii value and multiply each together to return one integer, e.g., Luke = 76 + 117 + 107 + 101 = 401). The getProductOfAsciiValues(String s) method will take each ascii value of a string passed and compute the product of the ascii values (get each ascii value and multiply each together to return one integer, e.g., Luke = 76 \* 117 \* 107 \* 101 = 96095844). Finally create a test class called **TestStringUtility**. Place this class in a package called **com.yourSurname.testUtil** and import and call the two utility methods of StringUtility using a sample string of your choice.

Required activities and marking guideline:

* Create utility class with two static methods in package (6 points)
* Complete Javadoc for class and methods (3 points)
* Test the utility method in a class in different package (1 point)

**Part 2 – Resource bundles and internationalization (15 points)**

Create an Eclipse Project called **Lab9Part2**. Create a simple JFrame class with one JLabel showing a single word, e.g., “Hello”. Add three buttons to the JFrame which displays the name of three languages of your choice. Change the label to translate the word shown in the label to each of the languages shown using either **ListResouceBundle** or **PropertyResourceBundle** approach (the choice is yours!). All buttons and the label should change to show the current locale.

Required activities and marking guideline:

* Three translations classed for buttons and label (2 points each) (6 points)
* GUI constructed with listeners (3 points)
* Locales created and language switching active (4 points)
* Full translations working on buttons and label for 3 languages (2 points)

**Part 3 – Threads and objects combined (20 points)**

Create an Eclipse Project called **Lab8Part2**. Create a mock banking system which stores Bank accounts in a class called **BankAccount**. The BankAccount class should have one attribute called balance (of type int), a constructor with a parameter of type int for starting balance, and two methods to **getBalance()** and **makeWithdrawl(int)**. The makeWithdrawl(int) method will subtract the amount passed from the **balance** attribute. Create a second class called **BankTransaction** that will be threaded (extends Thread). The **BankTransaction** class will have one static reference to one **BankAccount** object (set the opening balance at 10,000). The **run()** method of the **BankTransaction** class will withdraw 10 euros from the static **BankAccount** object and print the name of the thread that called the withdraw and the print the current balance. Create a third class called **RunBank** which create 1000 **BankTransaction** objects and call start() on the 1000 threads (there should be no money left at the end of the thread calls 1000 x 10 withdrawls). Examine the output from the several test runs you perform [sometimes, depending on the operating system it is possible that the last reported balance is incorrect when not using **synchronized**, e.g., too much money or too little money remains]. Finally make the **getBalance()** and the **makeWithdrawl(int)** methods of the **BankAccount** class **synchronized** and run and observe several tests runs again [using synchronized there should never be inconsistency in the balance reported].

Required activities and marking guideline:

* BankAccount class defined (constructor, attribute, methods (x2) (7 points)
* BankTransaction class defined (threaded) (5 points)
* RunBank class creates and runs 100 transactions (5 points)
* Synchronize methods getBalance() and makeWithdrawl(int) (3 points)

**Part 4 – Anonymous objects and anonymous inner class (15 points)**

Create an Eclipse Project called **Lab9Part4**. Create a JFrame called **FormFrame** with two inner classes called **RegisterPanel** and **SubmitResponder**. The **RegisterPanel** must extend JPanel and build a JPanel with a very simple input form (e.g. you can have one input field and\or label) and one button with the text set to “Submit”. The second inner class called **SubmitResponder** must implement ActionListener and will be the responder to the submit button defined in the **RegisterPanel**. The **FormFrame** constructor (the outer class) will create and add a non-anonymous instance of the **RegisterPanel** (as a class field) and the **RegisterPanel** will add an anonymous instance of the **SubmitResponder** to the submit button. When the submit button is pressed in the **RegisterPanel** the input field will print the message “Submit completed” and the input field will be set as uneditable; also once the submit button is pushed the title of the outer **FormFrame** to “Submit Completed!!!” using setTitle.

[N.B.: Marks will not be awarded if the **RegisterPanel** and **SubmitResponder** are not **inner classes** of the **FormFrame**.]

Required activities and marking guideline:

* Outer Frame with non-anonymous instance of panel added (3 points)
* Inner RegisterPanel created to create simple panel (3 points)
* Inner RegisterPanel class adds anonymous SubResponder (3 points)
* SubmitResponder changes the outer frame title and panel field (6 points)