EECS2030F: LAB 1 Due: Sept. 23, 2025 - 11:59 pm

The purpose of this lab is to ensure that you practice

A) designing recursive algorithms.

B) testing your code thoroughly using JUnit test cases.

C) Writing your own test cases.

D) generating the existing javaDoc into an html format.

1. Lab Policies

a) **Academic Integrity**: Submit your own work. Do not copy code from classmates or online sources. All violations will be reported as academic misconduct.

b) **Submission Format**: Submit only the file: **Lab1.java**. Do not upload ZIP files or project folders.

c) **Work Environment**: Complete the methods and test your submission by using tester file**.** Your code should not contain a main method and Scanner Class objects and its methods. Ensure your code compiles without errors.

d) **Deadline:** Submit your **Lab1.java** file to the eClass course page by **Tuesday, September 23 (11:59pm).** No late submissions are accepted. Email submissions are not accepted.

e) Your lab assignment is not graded during the weekly lab sessions scheduled. The lab sessions are meant to get your questions answered from TAs.

2. Downloading and importing the Starter Project

You have already done this step in Lab0. Follow the below listed steps to set up your working environment by:

a) Download the Eclipse Java project archive file from eClass: EECS2030_Lab1.zip

b) Launch Eclipse and browse to EECS2030-workspace (for instance or your own created workspace).

c) In Eclipse:

- Choose File->Import
- Under General, choose Existing Projects into workspace
- Choose Select archive file. Browse your compressed zip folder and attach it.
- Make sure that the EECS2030_Lab1 box is checked under Projects and you don't have the same project already in the workspace. Then Finish.

- You should see two files, one is called Lab1.java and one Test_Lab1.java.

3. Important Notes:

You are not allowed to use any regular expressions, loops, or any methods such as "contains," "replace," and so on to solve the problem. In other words, the problem should purely be solved by recursion, and therefore, any method that allows you to deviate from designing a recursive algorithm should be avoided. Also, the use of any kind of instance variables is not allowed. This is to ensure that the problem is solved solely by the recursive method.

To practice testing, we have only provided a set of incomplete test cases. Please make sure to add enough test cases to the tester that thoroughly tests your code. Look at the tester code, Test_Lab1.java, and add more test cases to ensure comprehensive testing of your code. To do this, you can copy one of the methods, change the name of the method to avoid having a duplicate method name, and modify the body of the method to test your code with your selected input.

4. Javadoc generation

The Javadoc has been written for you. All you need to do is generate it as an HTML file to make navigation easier. To do this, click on the lab1 package, select Project -> Generate Javadoc. It will ask you for the location where you want to store the documentation. You can choose the default location or select your own folder. Enter the path, and then click on Finish. If you look at the location where you stored the documentation, you'll see a file called index.html. Clicking on this file will display the project documentation in your browser.

If you have any doubts on the Javadoc generation, please review lecture slides of week2.

5. Programming Tasks for this Lab

❖ Programming Task 1

For this task, we are asking you to implement a recursive method that finds the sum of n consecutive integer numbers, starting from the integer start and ending with the integer end. For example, if start is 0 and end is 5, we expect the method to return 15 (i.e., 0 + 1 + 2 + 3 + 4 + 5). The name of the method is **sum**, and the header of the method has been written for you in Lab1.java.

❖ Programming Task 2

For this task, you are expected to implement a recursive function that creates and returns a string of length *n* using the given character. For example, if the given character is '*', and *n* is 5, the output should be '*****'. Please note that a string with a length of zero is also possible. To concatenate to a string, use the simple '+' operator. The name of the method is **makeString**, and the header of the method has been written for you in Lab1.java.

❖ Programming Task 3

For this task, you should implement a recursive function that takes three parameters and returns a string made of the first two input parameters repeatedly. This means the output contains the first input followed by the second input, followed by the first input again, and so on. The number of repetitions of the input strings is specified by the third argument of the method.

Here are some examples:

interlace("Hello ", "World ", 0) returns ""

interlace("Hello ", "World ", 1) returns "Hello "

interlace("Hello ", "World ", 2) returns "Hello World "

interlace("Hello ", "World ", 3) returns "Hello World Hello"

The name of the method is **interlace**, and the header of the method has been written for you in Lab1.java.

❖ Programming Task 4

In this task, you are required to write a recursive function that takes a string and two characters as parameters and returns the substring that is enclosed between the two given characters. You can assume that the given string includes only one instance of each enclosing character.

For example, if the input string is '*This is [quite} an example!*' and the first enclosing character is '[' and the second is '}', the function should return 'quite.'

The name of the method is **getSubstring**, and the header of the method has been written for you in Lab1.java.

❖ Programming Task 5

For this task, you need to write a recursive function that converts a positive integer, including zero, to its binary equivalent. This method takes one integer input.

The binary form of an integer is calculated by repeatedly dividing the integer number (and later, its quotient) by 2. You continue this division process until the quotient becomes zero. The remainders of the divisions, from the last division to the first, form the binary representation of the integer number. Let's consider an example in which 23 is converted to a binary number. 'R' stands for remainder.

23/2=11 R=1

11/2=5 R=1

5/2=2 R=1

2/2=1 R=0

1/2=0 R=1

Therefore, the binary representation of 23 is '10111.'

The name of the method is **decimalToBinary**, and the header of the method has been written for you in Lab1.java.

6. Testing your code

A test case file has been provided to evaluate your methods. Your implementation should pass all the test cases if it correctly handles both normal and edge cases. **Do not modify or update the provided test cases in any way. You should create your own test cases too to thoroughly test your code.**

7. Submit

Submit only one file named "Lab1.java" via eClass by clicking on the lab link.

8. Marking Schema

You are graded based on the correctness of your code. For each method, there will be a few test cases to examine the correctness of the code. All test cases carry the same weight.

Please note that in all the labs, even if the test cases are provided, we will test your code with a different set of test cases to ensure that you have tested your code completely.

Note:

- The program that does not compile will get zero marks.
- Don’t change the method headers. The test cases will fail if you change the method headers.

- Your submission should strictly have the name as **Lab1.java**, otherwise it can not be used to test your assignment. It will have zero grades.
- If any method has iterative solution, the method would be graded as zero and its corresponding test cases will be considered as failed which will further reduce the grades for test cases.
- No resubmissions are allowed. Therefore, please make sure you submit the correct file within due date.

****