CSCI 2110 Data Structures and Algorithms
Laboratory No. 6
Release Date: Oct 18th
Due Date: Oct 25th
23h55 (five minutes to midnight)

Recursion

Introduction
This lab is designed to help you get comfortable with recursion. In a recursive method, the method will call itself. Remember that you must first establish a *base case* and a *recursive case.* The base case is a trivial case which is easily solvable and where the recursive calls terminate. The recursive case involves the problem being broken down into smaller versions of itself.

Marking Scheme
Each exercise carries 10 points.
Working code, Outputs included, Efficient, Good basic comments included: 10/10
No comments or poor commenting: subtract one point
Unnecessarily inefficient: subtract one point
No outputs and/or not all test cases covered: subtract up to two points
Code not working: subtract up to six points depending upon how many methods are incorrect.

Your final score will be scaled down to a value out of 10.

Error checking: Unless otherwise specified, you may assume that the user enters the correct data types and the correct number of input entries, that is, you need not check for errors on input.

Submission: All submissions are through Brightspace. Log on dal.ca/brightspace using your Dal NetId.

What to submit:
Submit one ZIP file containing all source code (files with .java suffixes) and a text documents containing sample inputs and outputs. For each exercise you will minimally have to submit a demo Class with a main method to be tested and your test code, sample outputs.

Java Versions: Please ensure your code runs with Java 11.

Your final submission should include the following files: *Exercise1.java, Exercise2.java, Exercise3.java, Exercise1out.txt, Exercise2out.txt, Exercise3out.txt*

You MUST SUBMIT .java files that are readable by your TAs. If you submit files that are unreadable such as .class, you will lose points. Please additionally comment out package specifiers. Follow the naming guide per question for your java files and methods. Your code must compile. Failure to compile may result in a mark of 0.

Exercise1
Create a file with a main method called *Exercise1.java.* Implement the static methods below:

a) Factorial of an integer n:
   if n == 0, then factorial(n) = 1 //base case
   if n > 0, then factorial(n) = n * factorial(n-1)

Write a recursive static method that calculates the factorial of a non-negative integer *n*. Call this method in your main method, using a loop to print the factorials of the integers 1 through 10. Your method header should be: *public static int factorial(int n)*

b) Exponentiation (*x* to the power *n*):
if n == 0, then power(x,n) = 1 //base case
if n>0, then power(x,n) = power(x,n-1)*x

Write a recursive static method that calculates *x* to the power *n*, where *x* and *n* are both positive integers. Call this method in your main method, **prompting a user to enter *x* and *n***, and print *x* to the power of *n*. Your method header should be: *public static int power(int x, int n)*

Input/output
No input for problem a.
Problem b: your program, *Exercise1.java*, should prompt the user to enter x and n via the Scanner class. Print x to the power of n.

Student Generated Test Cases
Run your code for part a
Run your code for part b showing 3 examples of it functioning.
Save your inputs and outputs in a text file called *Exercise1out.txt*

Sample inputs and outputs

[no input for part a] Enter the base: *2* Enter the exponent: *3*	Factorials 1! =1 2! = 2 3! = 6 … 2 to the power of 3 is 8

Exercise 2
Create a file with a main method called *Exercise2.java.* Implement the static method below:
Write a recursive method called *countDown* that takes a single positive integer *n* as parameter and prints the numbers *n* through 1 followed by 'BlastOff!'. Your method header should be:
public static void countDown(int n)

countDown(10) would print:

```
10      9       8       7       6       5       4       3       2       1       BlastOff!
```

Your program should **accept input from a user** (specifying an integer *n*) and count down to 1 from *n* before printing 'BlastOff!'. Test your program with a variety of inputs to ensure its proper operation. Do not hard code inputs. Save the inputs and outputs from one test run in a text file called *Exercise2out.txt*

Input/output
A single integer entered by the user via the Scanner class.
Print your countdown from left to right.

Student Generated Test Cases

Run your code showing 2 examples of it functioning.

Save your inputs and outputs in a text file called *Exercise2out.txt*

Sample inputs and outputs

Enter an integer: *10*	10 9 8 7 6 5 4 3 2 1 BlastOff!

Exercise 3

Create a file with a main method called *Exercise3.java.* Implement the static method below:
Write a recursive method called *multiples* to print the first *m* multiples of a positive integer *n*. Your method header should be:
public static void multiples(int n, int m)

multiples(2, 5) would print:
2, 4, 6, 8, 10

multiples(3, 6) would print:
3, 6, 9, 12, 15, 18

Your program should **accept input from a user** (specifying integers *n* and *m*) and print multiples. Test your program with a variety of inputs to ensure its proper operation. Do not hard code inputs. Save the input and output data from one test run in a text file called *Exercise3out.txt*
Note: You may print multiples in either ascending or descending order.

Input/output
Your program should accept input in the following format:
User entered integers n and m. Use the Scanner class.

Your program should provide output in the following format:
See below.

Six Student Generated Test Cases
Run your code showing 2 examples of it functioning.
Save your inputs and outputs in a text file called *Exercise3out.txt*

Sample inputs and outputs:

Enter a value: 3 How many multiples do you want? 6	The first 6 multiples of 3 are: 3, 6, 9, 12, 15, 18