CSCI 2110 Data Structures and Algorithms
Lab 8
Release Date: Nov 1st
Due: Sunday 8 November
23h55 (five minutes to midnight)

Binary Trees

The objective of this lab is to help you get familiar with binary trees. Download the example code/files provided along with this lab document. You will need the following files to complete your work:

BinaryTree.java (Generic BinaryTree Class)
BinaryTreeDemo.java (A class demonstrating the methods of the BinaryTree class)

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. Markers can test cases beyond those shown in sample inputs, and may test methods independently.

Your final score will be scaled down to a value out of 10. For example, if there are three exercises and you score 9/10, 10/10 and 8/10 on the three exercises, your total is 27/30 or 9/10.

Error checking: Unless otherwise specified, you may assume that a 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 to dal.ca/brightspace using your Dal NetId.

What to submit:
Submit one ZIP file containing all source code (files with .java suffixes) and text documents containing sample outputs. For each exercise you will minimally have to submit a class containing your completed methods, a demo class, and sample outputs.

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

Your final submission should include the following files: *BinaryTree.java, BinaryTreeDemo.java, Exercise1.java, Exercise1out.txt*

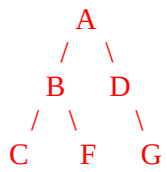
You MUST SUBMIT .java files that are readable by your TAs. If you submit files that are unreadable such as .class, you will lose points. Please additionally comment out package specifiers. Follow the naming guide per question for your java files and methods. Your code must compile. Failure to compile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

Exercise 0 (Binary Tree Methods) -No Marks

For this exercise, you will complete a number of methods in the BinaryTree.java file. It is recommended that you use pen and paper to first do a diagram of a binary tree and do a method trace by hand for each method below:

1. Complete the recursive method called nodes in the BinaryTree.java file. This method should return the number of nodes in a binary tree.
   To determine the number of nodes in a binary tree you can apply the following rules:
   - if the binary tree is empty (null), then the number of nodes is zero.
   - otherwise, the number of nodes is equal to one plus number of nodes in the left subtree plus number of nodes in the right subtree.
2. Complete the recursive method called height in the BinaryTree.java file. This method should return the height of a binary tree.
   To determine the height of a binary tree you can apply the following rules:
   - if the binary tree is empty (null), then the height of the binary tree is -1.
   - otherwise, the height is equal to 1 plus the height of either the left subtree or the right subtree, whichever is greater.
3. Complete the recursive isBalanced method in the BinaryTree.java file. This method should return a Boolean reflecting whether or not a binary tree is height balanced.
   A binary tree is height balanced if, for every node in the tree, the height of its left subtree differs from the height of its right subtree by no more than one. In other words, either the left and the right subtrees are of the same height, or the left is one higher than the right, or the right is one higher than the left.
4. Complete the levelorder method in the BinaryTree.java file. This static, dynamic method should perform a level order traversal of a binary tree passed to it as an argument. To accomplish this, you will implement a simple breadth first search. You can use an ArrayList of type BinaryTree as your agenda, adding your root node before iterating. Traverse the tree by removing the first element from the agenda, printing, and adding the children of the node removed to the agenda. Repeat until your agenda is empty.

Note: A level order traversal goes level by level, from left to right, starting at the root.
The level order traversal of the tree:



Is A B D C F G

Test the methods you have written with at least three different sets of inputs by making appropriate changes to the BinaryTreeDemo.java file provided.

Exercise1

For this exercise you will develop a program that can build a binary tree from user input. Create an Exercise1 class (Exercise1.java). You may reuse and expand upon code provided in the BinaryTreeDemo.java file, but your program must **accept input from a user** (an arbitrary number of Strings) and build a binary tree with String data from that input.

The most straightforward approach is to create single node binary tree from each String captured, storing them in an ArrayList or Queue of type BinaryTree. You can then quickly build a larger binary tree by assigning the first tree in your ArrayList to be the root node and attaching subsequent elements to that root. You may follow the steps below. Do an example with pen and paper first to see how the tree is built:

1. Create a queue called *trees* which will store binary trees of type String.
2. Read in the user input, store it in a binary tree, and store the tree in the queue *trees*.
3. Create a queue called *agenda* to store binary trees of type String
4. Create a binary tree reference called *trueroot*.
    a. Remove the first element from *trees* and store it in *trueroot*
    b. Add *trueroot* to *agenda*
5. While *trees* is not empty
    a. Remove the element at the front of *agenda*
    b. Remove the element at the front of *trees*
    c. Attach the element from *trees* to the left of the element from *agenda*
    d. Add the same element from *trees* that you just attached to *agenda*
    e. If *trees* is still not empty
        i. Remove the first element from *trees*
        ii. Attach it to the right of the element from *Agenda*
        iii. Add the same element from *trees* that you just attached to *agenda*

After building a binary tree, your program should display the height of the tree, the number of nodes in the tree, the inorder, preorder, postorder and level order traversals. You may reuse the test code from the BinaryTreeDemo.java file to this end.

Note: You are building a complete binary tree. It will, by definition, be height balanced. If the Strings provided by the user are Raspberry, Chocolate, Vanilla, Mango, and Cinnamon, the resulting tree will look like this:

Input/output

User entered Strings via scanner.
See below for output details

Student Generated Test Cases

Test your program with at least three different sets of inputs. Save the results of your inputs in a file called *Exercise1out.txt*. **Use the input below as your first test case.**

Input:
Enter name or done: Raspberry
Enter name or done: Chocolate
Enter name or done: Vanilla
Enter name or done: Mango
Enter name or done: Cinnamon
Enter name or done: done

Output:
Height of the tree is: 2
Number of nodes in the tree is: 5

Inorder:	Mango	Chocolate	Cinnamon	Raspberry	Vanilla
Preorder:	Raspberry	Chocolate	Mango	Cinnamon	Vanilla
Postorder:	Mango	Cinnamon	Chocolate	Vanilla	Raspberry
Level order:	Raspberry	Chocolate	Vanilla	Mango	Cinnamon