Lab Assignment 2
Binary Trees

Due Dates	
Wednesday Labs	**Wednesday 10/02/19 @ 11:59pm**

Objectives

- Review Recursion
- Review Interfaces
- Review Binary Trees (a data structure in Computer Science)

Problem Specification

Develop a Java application to implement a binary tree data structure. A **tree** data structure starts from the top node, called the **root**. Each node in the tree has a set of children, which are also nodes of the tree and can have children of their own, and so on. This keeps on going till we get to the bottom of the tree, to the "**leaf**" nodes. Each node in the tree, except for the root, has a parent. A **binary** tree is a specialized form of a tree data structure in which each node in the tree has **at most two children**. Shown below is a labelled diagram of a binary tree.

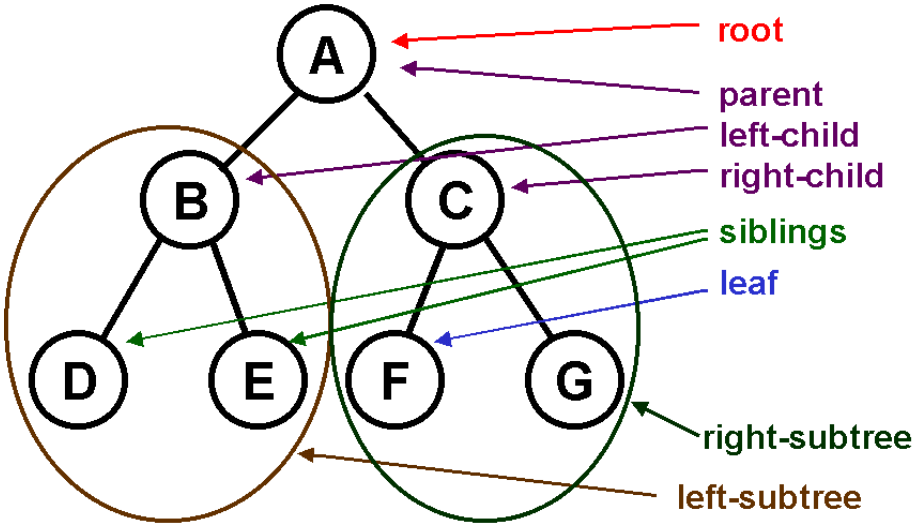

Figure 1: Image taken from www.mamtatutorial.com

A binary tree can be represented using the format in the example below. Each line contains at most three characters. The first character is the ID of the node itself. The next two characters are the child nodes. If a line has only 2 characters, then that node has only one child. If a line has one character only, then it is a leaf node (i.e., it has no children).

Example:
A B C
B D E
C F G
D H I
E J K
F L
G
H
I
J
K
L

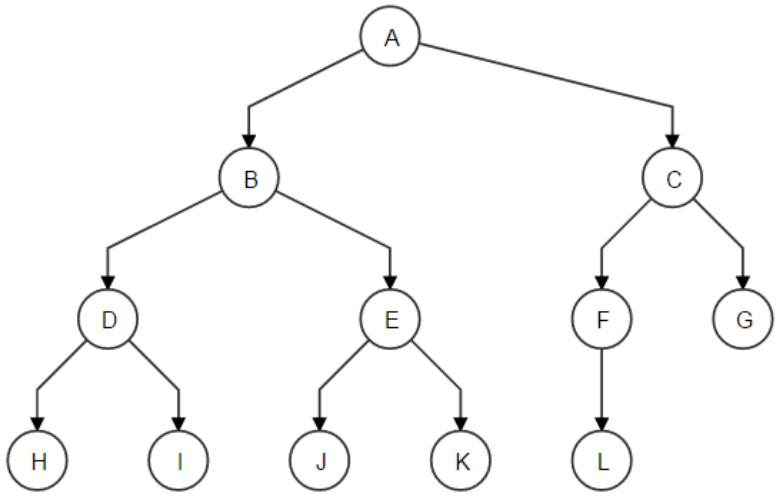

Figure 2: A diagrammatic representation of the binary tree in our example ('A' is the root node.).

The application should exhibit the following functionality (**see the sample output provided**):

- Display a menu to the user and request for the desired option.
- Based on the user's input, request for additional information as follows:
    - If the user wants to add a node, request for the name / ID of the new node to be added as well as the name of the desired parent of that node.
        - If the parent node already has two children, the new node cannot be added since this is a binary tree (and each node can only have at most two children).
            - Display an appropriate message to the user.
            - Display the menu again for the user to make another choice.
        - If the parent node already has one child, use **recursion** to add the new node as the second child (right child) of the parent node.
            - Display the new tree with the new node added. This should be done using a **recursive** method in your project (**printTree**()).
            - Display the menu options.
        - If the parent node has no children, use **recursion** to add the new node as the left child of the parent node.
            - Display the new tree with the new node added.
            - Display the menu options.
    - If the user wants to know the size of the tree (i.e., the number of nodes in the tree or sub-tree), request for the name / ID of the node that is the **root** of the desired tree / sub-tree. The size of a tree or sub-tree is the number of its children and other 'descendants' (including any leaf nodes) plus one – the root node itself. For example, the size of the sub-tree with root 'B' in Figure 2 above is 7.
        - **Recursively** count the number of nodes in the tree / sub-tree and display this with an appropriate message.
            - Display the newly updated tree.
            - Display the menu options.

- If the user wants to find a node, request for the name / ID of the node to search for in the tree.
  - Search **recursively** for the node in the tree; if the node is found, display an appropriate message, otherwise, indicate that the node does not exist in the tree.
    - Display the menu options.
- If the user wants to exit, terminate the program.

A sample output (to be followed exactly) is included below.

Example Output

```
A B C
B D E
D H I
H
I
E J K
J
K
C F G
F L
L
G
There are 12 nodes in this tree.

Please select from one of the following options:
1. Add Node
2. Tree Size
3. Find Node
0. Exit
->1
Please input the node you want to add->
P
Please input the parent node of P->
A
Parent already has 2 children.
Please select from one of the following options:
1. Add Node
2. Tree Size
3. Find Node
0. Exit
->1
Please input the node you want to add->
P
Please input the parent node of P->
F
Node successfully added!
A B C
B D E
D H I
H
I
E J K
J
K
C F G
F L P
```

```
L
P
G
Please select from one of the following options:
1. Add Node
2. Tree Size
3. Find Node
0. Exit
->2
Please input the root node->
F
There are 3 nodes in that tree
F L P
Please select from one of the following options:
1. Add Node
2. Tree Size
3. Find Node
0. Exit
->3
Please input the node you want to look for->
P
Node P found!
P
Please select from one of the following options:
1. Add Node
2. Tree Size
3. Find Node
0. Exit
->3
Please input the node you want to look for->
N
Node N does not exist.
Please select from one of the following options:
1. Add Node
2. Tree Size
3. Find Node
0. Exit
->0
```

Design Requirements

The main class in your application should be named **BinaryTree**. This class should instantiate an instance of the second class – to be named **TreeDataStructure** – and that instance should be used to call the appropriate methods to generate the initial tree, display the menu to the user and exhibit the functionality described above.

Apart from the main method, the BinaryTree class should also have a method to print the menu options (**void printMenu**()). You can copy and paste the code below into your main class to generate the initial tree.

```
public static void main(String[] args) {
	TreeDataStructure root = new TreeDataStructure("A");
	root.addChild("B", "A");
	root.addChild("C", "A");
	root.addChild("D", "B");
	root.addChild("E", "B");
```

```
        root.addChild("F", "C");
        root.addChild("G", "C");
        root.addChild("H", "D");
        root.addChild("I", "D");
        root.addChild("J", "E");
        root.addChild("K", "E");
        root.addChild("L", "F");


        /*
         *
         * Include code here to implement required functionality.
         */

    }
```

The **TreeDataStructure** class should implement the **INode** interface (which is provided below). The methods **addChild**(), **find**(), **printTree**(), and **size**() in the TreeDataStructure class must be implemented **recursively**.

NOTE: *Using recursion is one of the main objectives of this assignment. A solution that does not use recursion to implement the required functionality will earn **zero points**.*

Create an interface named INode within your package (following steps similar to how you would create a class) and copy and paste the interface code below into it.

```
public interface INode {

    /**
     * This method checks to see if the specified parent node exists in the tree. If
     * not, it returns false after printing an appropriate message. If the node exists,
     * the method checks to see if it already has two children. If it does, the method
     * returns false. Otherwise, it either adds the new node as the parent node's left
     * child (if the parent has no children) or as the right child (if the parent
     * already has one child).
     *
     * @param ID new node to add
     * @param parentID parent node in Tree
     * @return true if successful, false otherwise
     */
    public boolean addChild(String ID, String parentID);

    /**
     * This method looks within the tree to find if "value" (the ID of the node to be
     * found) is contained in that subtree. The node used to call the find method acts
     * as the root of that tree / subtree.
     *
     * @param value a string (ID of a node) to be found in the tree
     *
     * @return the node if found.
     */
    public INode find(String value);

    /**
     * Gets the parent of this node.
     *
```

```
         * @return the parent node of the Node used to call this method.
         */
        public INode getParent();

        /**
         * Gets the size of the tree.
         *
         * @return the size of the tree starting from the node that is used to call this
         * method, all the way down to the leaf nodes.
         */
        public int size();
        /**
         * Method to get the ID of the node.
         *
         * @return String representation of the node ID
         */
        public String toString();
        /**
         * Method to get the ID of the node.
         * @return ID
         */
        public String getId();

        /**
         * Prints the node upon which this method is called as well as all the children
         * nodes to show the structure of the tree.
         * Uses the toString() format to print.
         */
        public void printTree();

}       // End of INode interface
```

Hints

1. It is important to store information about the parent of each node as part of that node's internal structure. In addition, each node should have information about its' children.
2. You may represent the children of a node possibly using an array of nodes.

Implementation Phase

Using the pseudocode developed, write the Java code for your assignment. This is a two-week assignment.

Testing Phase

- Your program should include code to validate user input and ensure that all inputs are of the required type (characters for IDs of nodes, integers from 0 to 3 for menu options). If input is not valid, your program should keep looping to force the user to provide valid input. Each time your program loops, it should provide information to the user to indicate what is considered valid input.

- Build your program incrementally, carefully testing each method as you go.

Additional Requirements

A proper *design* (with detailed pseudocode) and proper *testing* are essential.

*Note: **Correct pseudocode development** will be worth **40%** of the total LA grade.*

You will need to **generate Javadoc** for your project. Follow the steps shown in class (a copy of these steps can be found on the Content page in Elearning. If you follow the steps accurately, a "**doc**" folder (for Javadoc) should be created in your project.

Coding Standards

You must adhere to all conventions in the CS 1120 Java coding standards (available on Elearning for your Lab). This includes the use of white spaces and indentations for readability, and the use of comments to explain the meaning of various methods and attributes. Be sure to follow the conventions also for naming classes, variables, method parameters and methods.

Assignment Submission

- Generate a .zip file that contains all your files including:
    - Program files
    - Any input or output files
    - Detailed pseudocode for your program

  You can refer to the pdf on Elearning (" How to submit a programming assignment") provided for instructions on how to export your project to a .zip file.

- Submit the .zip file to the appropriate folder on Elearning.

NOTE: The eLearning folder for LA submission will remain open beyond the due date but will indicate how many days late an assignment was submitted where applicable. The dropbox will be inaccessible seven days after the due date by which time no more credit can be received for the assignment.

The penalty for late submissions as stated in the course syllabus will be applied in grading any assignment submitted late.