Aashray Shrestha

October 2, 2019

CS 1120

Assignment 2 : Binary Tree

The following pseudocode was used to develop a Java application that creates a Binary tree which can be modified according to the user input. There is also a find node method that finds the node the user want to check in the tree. The find node ,add node, and size method use recursion.

Pseudocode

MainMethod : BinaryTree

Class : TreeDataStructure implements the INode class

Interface : INode

The following pseudocode was used to develop the program :

Binary Tree :

- Creating a scanner to be used throughout the program to get user inputs
- Instantiating the TreeDataStructure object defined by **root** and passing “A” as an argument to denote the root of the tree
- The addChild method from the TreeDataStructure is used to create the Binary tree. The arguments passed are the id of the child, and the parentId.
- Printing the tree after the initial completion of the addition of the given code which ends after adding “L” as a child to the “F” node
- Using the size method from the TreeDataStructure to calculate the number of nodes. Initially, it just prints out the number of nodes in the given tree before any user input.
- The first input is stored in the variable **user** that has called the method **getInput.** The getInput method takes the Scanner bucky as an argument and returns an integer that will run

the program accordingly after input Validation on the user entered number as the only values that are accepted are 1,2,3 and 0.

- After the integer is returned, the program runs according to the value returned :
    - Input is 0 : Exit System
    - Input is 1 : Add Node
        - Asks the user the name of the node to add
        - Asks the user what parent node it is supposed to be added to
        - The output is according to the return value of the addChildmethod.
            - true : Print Successful Message
            - false : displays unsuccessful message (from the addChild method itself.)
            - True = Print Successful Method
            - False = Print Unsuccessful Method (used as else in the addChild method itself
    - Input is 2 : Tree Size
        - Asks the user the name of the node to start the count for the tree size
        - The use of **INode find** and the **root** is used to get the **INode size** which can
    - Input is 3 : Find Node
        - Asks the user the parent Node
        - The use of **INode find** and the **root** is used to get the **INode value** which can either be a null value returned ,which displays node does not exist message ,or a value which is that the node is found.
    - Input is 0 : System exit
- getInput (Scanner input) : This method checks to see if the user input value is 1, 2, 3, or 0 and does not work until the flag is false in the while loop. When the flag is false, the while loop

stops and returns the number entered by the user. This method uses the userValidation method.

- userValidation (String user, String validationType) : the method returns a boolean
    - This uses the try catch exception handling to check if the number entered by the user is valid. It returns a boolean value for the getInput method to use.
- printMenu () : void method to print the menu

```
System.out.println(" 1. Add Node\n 2. Tree Size\n 3. Find Node\n 0. Exit");
```

TreeDataStructure Class :

The TreeDataStructure class has a private String id, with a getter method, and private objects of TreeDataStructure leftNode and rightNode.

- Constructors :

1. TreeDataStructure(String root) : one parameter that is the id for the root only
2. TreeDataStructure(String ID, String parentID) : two parameters of child ID and parentID

addChild method (String ID, string parentID) :

- returns a boolean value true or false
- Initial value of both boolean : one for checking left side and one for checking right side of the tree is false
- The main idea is to go on checking each of the nodes to find the node(parentId) where the user wants the child to be added. This has been done using many if statements that go from the root to the left and right child of the root, followed by the left and right child of each of the first left and child.
- In the if command itself, creation of a new TreeDataStructure objects help change the node value and have the program recursively until it reaches a conclusion (with the use of the INode find method)which can be as:

  1. Did not find parent node : return false
  2. Found parent node : has no child : add to left
  3. Found parent node : has one child : add to right
  4. Found parent node : has two children : can not add message

INode find(String value) :

- The method uses the getter method for the getId and checks if the value is the first value and returns accordingly.
- Furthermore, the if statement is used to see if the value exists in the tree and returns the node according to the if-else command and statements.
    - It checks through each of the nodes recursively.

INode getParent() :

- Gets the parent node passed in the constructor according to constructor used

String getId() :

- A getter method to get the ID for the private field id

int size() :

- Checking the size of the tree recursively by using the left and right node recursive calls
- Returns a value which is the count of the Node

toString() method :

- The toString method runs through the program to create the binary tree and returns the final String which contains the binary tree format.
- The binary tree format is String representation if the Node ID

printTree() : Print out the Binary tree using the toString method and recursion until all the value is printed out.

Public Interface Uses the Following Code :

```java

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 nodes 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 parrentID);
    /**
     * 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

```