CS 340 Fall 2022
Programming Project 3

Due 11:59 PM Monday October 10
This project must be submitted on time.
Project 3

- Implement the class shown in the following slides. The class implements a binary tree. You must write your own test driver. Your class implementation must not depend on the test driver. I will use my own test driver to test your class.
public class BinaryTree {
    //Implements a Binary Tree of Strings
    private class Node {
        private Node left;
        private String data;
        private Node right;
        private Node parent; //reference to the parent node
            // the parent is null for the root node

        private Node(Node L, String d, Node r, Node p) {
            left = L;
            data = d;
            right = r;
            parent = p;
        }
    }
    private Node root;
}
public BinaryTree() {
    //create an empty tree
}

public BinaryTree(String d) {
    //create a tree with a single node
}

public BinaryTree(BinaryTree b1, String d, BinaryTree b2) {
    //merge the trees b1 AND b2 with a common root with data d
    //this constructor must make a copy of the contents of b1 and b2
}
public BinaryTree(String t, String open, String close, String empty) {

/*create a binary tree from the post order format discussed in class. Assume t is a syntactically correct string representation of the tree. Open and close are the strings which represent the beginning and end markers of a tree. Empty represents an empty tree. The example in class used ( ) and ! for open, close and empty respectively. The data in the tree will not include strings matching open, close or empty. All tokens (data, open, close and empty) will be separated by white space. Most of the work should be done in a private recursive method */
}

Binary Tree String Representation
Postorder Format

• Empty Tree: !

• Non-Empty Tree: ( left right data )

• Examples
  
  • !
  
  • ( ! ! A )
  
  • ( ( ( ! ! C ) ! B ) ! A )
  
  • ( ! ( ! ( ! ! C ) B ) A )
  
  • ( ( ! ! C ) ( ! ! B ) A )
  
  • ( ( ( ! ! D ) ( ( ! ! F ) ( ! ! G ) E ) B ) ( ! ( ( ! ! I ) ! H ) C ) X )
  
public class PostorderIterator implements Iterator<String> {
    //An iterator that returns data in the tree in a post order pattern
    //the implementation must use the parent pointer and must not use an
    //additional data structure

    public PostorderIterator() {
    }

    public boolean hasNext() {
    }

    public String next() {
    }

    public void remove() {
        //optional method not implemented
    }
}
public class InorderIterator implements Iterator<String> {
    //An iterator that returns data in the tree in an in order pattern
    //This implementation must use a stack and must not use the parent pointer
    //You must use Java’s stack class

    public InorderIterator() {
    }

    public boolean hasNext() {
    }

    public String next() {
    }

    public void remove() {
        //optional method not implemented
    }
}
public class PreorderIterator implements Iterator<String> {
    //An iterator that returns data in the tree in a pre order pattern
    //This iterator will recursively do the complete iteration saving the
    //values in the order they are visited in a list (you can use ArrayList or
    //LinkedList

    public PreorderIterator() {
    }

    public boolean hasNext() {
    }

    public String next() {
    }

    public void remove() {
        //optional method not implemented
    }
}
Project 3

```java
public class LevelorderIterator implements Iterator<String> {
    //An iterator that returns data in the tree in a level order pattern
    //This implementation uses a FIFOQueue

    public LevelorderIterator() {
    }

    public boolean hasNext() {
    }

    public String next() {
    }

    public void remove() {
        //optional method not implemented
    }
}
```
public Iterator<String> inorder() {
    //return a new in order iterator object
}

public Iterator<String> postorder() {
    //return a new post order iterator object
}

public Iterator<String> preorder() {
    //return a new pre order iterator object
}

public Iterator<String> levelorder() {
    //return a new level order iterator object
}

public String toString() {
    //returns the string representation of the tree using the post order format discussed in class. If the tree was created from a string, use the //the values of open, close and empty given to the constructor otherwise //use (, ) and ! for open, close and empty respectively //most of the work should be done in a recursive private method.
}

Project 3 Submission

• Upload one zip file to Canvas. The zip file must contain only one file called BinaryTree.java. Do not upload your whole Eclipse project!

• Include a comment with your name at the top of the file and add comments for any private instance variables or methods that you add.