**Project Documentation**

**Name: Dan Beck**

**Assignment: Project 3**

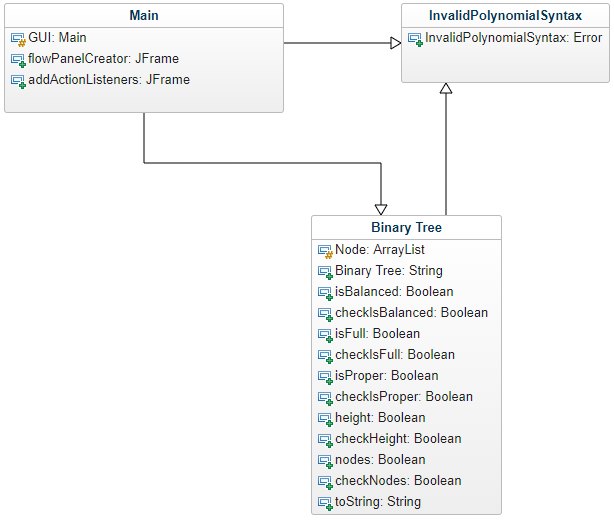
**Date: September 29th, 2020**

**Problem Statement**: A program that allows the user to enter a binary tree in parenthesized prefix format and then allows for checking if it Is Balanced, Is Full, Is Proper, Height, Nodes, and In Order.

**Analysis:** Tree used:

(A(G(j)(1))(z(5)))

(A(G(j)(1)) (z())) – for error

**Design (for project assignments only):** 

**Code:**

package BeckProject3;

import java.awt.event.ActionListener;

import javax.swing.JButton;

import javax.swing.JComponent;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JTextField;

import java.awt.Component;

import java.awt.FlowLayout;

import java.awt.GridLayout;

/\* File: Project 3 - GUI

\* Author: Dan Beck

\* Date: September 29, 2020

\* Purpose: Class that generates the GUI and passes parameters to

\* other classes.

\*/

public class GUI extends JFrame

{

private static final long serialVersionUID = 1L;

private JTextField input = new JTextField(40);

private JTextField output = new JTextField(40);

private static BinaryTree categories;

public static void main(String[] args)

{

// Executes the program

GUI createFrame = new GUI();

createFrame.setVisible(true);

}//End Main

public GUI()

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: Constructor that generates the frame

\* 1. Default settings

\* Layers:

\* 2. Input Field

\* 3. Buttons

\* 4. Output Field

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: 1. Default Settings

\* A. Title

\* B. Size

\* C. Layout

\* D. Default Settings

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//A. Title

super("Binary Tree Categorizer");

//B. Size

setSize(800, 200);

setLocationRelativeTo(null);

//C. Layout

//For this GUI, three levels 3x1 (input text, buttons and output text)

setLayout(new GridLayout(3, 1));

//D. Default settings

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setResizable(false);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: 2. Input Field

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

JComponent[] inputText =

{

new JLabel("Enter Tree: "), input

};

flowPanelCreator(inputText);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: 3. Buttons

\* Creates the buttons laterally in order from left to right

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

JButton[] buttons =

{

new JButton("Make Tree"),

new JButton("Is Balanced?"),

new JButton("Is Full?"),

new JButton("Is Proper?"),

new JButton("Height"),

new JButton("Nodes"),

new JButton("In Order")

};

flowPanelCreator(buttons);

addActionListeners(buttons);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: 4. Output Field

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

JComponent[] outputText =

{

new JLabel("Output: "), output

};

flowPanelCreator(outputText);

output.setEditable(false);

}//end public GUI()

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: Creates a flow panel from array of panelObjects

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

private void flowPanelCreator(JComponent[] panelObjects)

{

JPanel jp = new JPanel(new FlowLayout());

for (Component panelObject: panelObjects)

{

jp.add(panelObject);

}//end for (Component panelObject: panelObjects)

add(jp);

}//end private void flowPanelCreator(JComponent[] panelObjects)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: Method that adds ActionListener to panel's buttons

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

private void addActionListeners (JButton[] buttons)

{

for (JButton button: buttons)

{

button.addActionListener(panelListener);

}//end for (JButton button: buttons)

}//end private void addActionListeners (JButton[] buttons)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: What the ActionListener performs for each button

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

private final ActionListener panelListener = event ->

{

try

{

switch ((event.getActionCommand()))

{

case "Make Tree":

categories = new BinaryTree(input.getText());

output.setText(categories.toString());

break;

case "Is Balanced?":

output.setText(String.valueOf(categories.isBalanced()));

break;

case "Is Full?":

output.setText(String.valueOf(categories.isFull()));

break;

case "Is Proper?":

output.setText(String.valueOf(categories.isProper()));

break;

case "Height":

output.setText(String.valueOf(categories.height()));

break;

case "Nodes":

output.setText(String.valueOf(categories.nodes()));

break;

case "Inorder":

output.setText(categories.inOrder());

break;

}//end switch ((event.getActionCommand()))

}//end try

catch (InvalidTreeSyntax its)

{

JOptionPane.showMessageDialog(getParent(),its.getMessage());

}//end catch (InvalidTreeSyntax its)

catch (IndexOutOfBoundsException e)

{

JOptionPane.showMessageDialog(getParent(),"No input given!");

}//end catch (IndexOutOfBoundsException e)

};

}//End GUI class

package BeckProject3;

import java.util.EmptyStackException;

import java.util.Stack;

/\* File: Project 3 - BinaryTree Class

\* Author: Dan Beck

\* Date: September 29, 2020

\* Purpose: Class that accepts the tree and check is the tree

\* Is Balanced, Is Full, Is Proper, Height, Nodes, and In Order

\*/

public class BinaryTree

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: Creates the node

\* Creates nodes to be used in entered tree

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public static class Node

{

private String info;

private Node left;

private Node right;

public Node(String info)

{

this.info = info;

}//end public Node(String info)

private void addChild(Node child) throws InvalidTreeSyntax

{

//simple conditions for nodes, can have at most 2 children

if (this.left == null)

{

this.setLeft(child);

}//end if (this.left == null)

else if (this.right == null)

{

this.setRight(child);

}//end else if (this.right == null

else

{

throw new InvalidTreeSyntax("Nodes can only have 2 children!");

}//end else

}//end private void addChild(Node child) throws InvalidTreeSyntax

//Setters for the left and right nodes

private void setLeft(Node newLeft)

{

left = newLeft;

}//end private void setLeft(Node newLeft)

private void setRight(Node newRight)

{

right = newRight;

}//end private void setRight(Node newRight)

@Override

public String toString()

{

return toString(this);

}//end public String toString()

// recursively printing out the nodes

private static String toString(Node root)

{

return (root == null) ? "" : "(" + root.info + toString(root.left) + toString(root.right) + ")";

}//end private static String toString(Node root)

}//end public static class Node

//After the constructor, parent and child nodes created

Node parent, child;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: Buttons

\* 1. Make Tree

\* 2. Is Balanced?

\* 3. Is Full?

\* 4. Is Proper?

\* 5. Height

\* 6. Nodes

\* 7. In Order

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: 1. Make Tree

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//1. Make Tree

public BinaryTree(String input) throws InvalidTreeSyntax

{

Stack<Node> nodeStack = new Stack<>();

// remove first & last parenthesis

String[] inputArray = input.substring(1, input.length()-1)

//and split the String into an arr of strings, retain the parenthesis

.split("(?<=\\()|(?=\\()|(?<=\\))|(?=\\))");

//setting the new first character to the root

parent = new Node(inputArray[0]);

//loop starting on the third character of the string

for (int i = 1; i < inputArray.length - 1; i++)

{

//if there is another child. Child becomes parent if one exists

if (inputArray[i].equals("("))

{

nodeStack.push(parent);

if (child != null)

{

parent = child;

}//end if (child != null)

}//end if (inputArray[i].equals("("))

else if(inputArray[i].equals(")"))

{

try

{

child = parent; parent = nodeStack.pop();

}//end try

catch (EmptyStackException emptyStack)

{

throw new InvalidTreeSyntax("Incorrect parenthesis!");

}//end catch (EmptyStackException emptyStack)

//if it gets here, it is a value to be assigned as child to parent.

}//end else if(inputArray[i].equals(")"))

else

{

child = new Node(inputArray[i]);

if (parent != null)

{

parent.addChild(child);

}//end if (parent != null)

}//end else

}//end for (int i = 1; i < inputArray.length - 1; i++)

//for every node, will have 2 parenthesis

if (this.checkNodes(parent) \* 3 != input.length()) throw new InvalidTreeSyntax("Incorrect Syntax");

}//end public BinaryTree(String input) throws InvalidTreeSyntax

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: 2. Is Balanced

\* determine if the absolute difference between branches is at most 1.

\* calls recursive method, which also calls recursive height method.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public boolean isBalanced()

{

return checkIsBalanced(this.parent);

}//end public boolean isBalanced()

private boolean checkIsBalanced(Node root)

{

//base case

if (root == null)

{

return true;

}//end if (root == null)

//return true if the absolute difference is at most 1

return (Math.abs(checkHeight(root.left) - checkHeight(root.right)) <= 1) &&

(checkIsBalanced(root.left) && checkIsBalanced(root.right));

}//end private boolean checkIsBalanced(Node root)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: 3. Is Full

\* determines if a tree has the maximum nodes for the height or not.

\* calls recursive method, which also calls recursive height method.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public boolean isFull()

{

return checkIsFull(this.parent, checkHeight(this.parent), 0);

}//end public boolean isFull()

//the index of of parent in this exercise is 0

private boolean checkIsFull(Node root, int height, int index)

{

//if it is empty, by BT logic: it is full

if (root == null)

{

return true;

}//end if (root == null)

//check to see if height is same among leaves

if (root.left == null && root.right == null)

{

return (height == index + 1);

}//end if (root.left == null && root.right == null)

//one child empty

if (root.left == null || root.right == null)

{

return false;

}//end if (root.left == null || root.right == null)

//recursive call to both children

return checkIsFull(root.left, height, index+1) &&

checkIsFull(root.right, height, index+1);

}//end private boolean checkIsFull(Node root, int height, int index)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: 4. Is Proper

\* determines if every node in a tree has either 2 or 0 children.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public boolean isProper()

{

return checkIsProper(this.parent);

}//end public boolean isProper()

private boolean checkIsProper(Node root)

{

//base case

if(root == null) {return true;}

//returns true or false based on two or zero children

return ((root.left != null || root.right == null) &&

(root.left == null || root.right != null))

&& (checkIsProper(root.left) && checkIsProper(root.right)); // and calling recursively

}//end private boolean checkIsProper(Node root)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: 5. Height

\* finds the height of the binary tree, where the root node is 0.

\* calls the recursive method, which adds one to the the larger of left or right

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public int height()

{

return checkHeight(this.parent)-1;

}//end public int height()

//subtracted one since in this exercise, root is 0

private int checkHeight(Node root)

{

//adds one to the greater of left and right, zero if null

return (root == null) ? 0 : 1 + Math.max(checkHeight(root.left),

checkHeight(root.right));

// found every chance to use the conditional operator in this (had a lot of single if/else's)

}//end private int checkHeight(Node root)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: 6. Nodes

\* finds the amount of nodes in a binary tree. Calls the recursive method,

\* which adds one for every node of left and right subtree, 0 if null.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public int nodes()

{

return checkNodes(this.parent);

}//end public int nodes()

private int checkNodes(Node root)

{

//adds 1 for both left and right. If null, zero

return (root == null) ? 0 : 1 + checkNodes(root.left) +

checkNodes(root.right);

}//end private int checkNodes(Node root)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: 7. In Order

\* prints the info of the nodes in the binary tree in order.

\* Calls the recursive method which uses the algorithm left -> node -> right

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public String inOrder()

{

return checkInOrder(this.parent);

}//end public String inOrder()

private String checkInOrder(Node root)

{

return (root == null) ? "" : "(" + checkInOrder(root.left) + root.info + checkInOrder(root.right) + ")";

}//end private String checkInOrder(Node root)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DESCRIPTION: toString

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

@Override

public String toString()

{

return parent.toString();

}//end public String toString()

}//end public class BinaryTree

package BeckProject3;

/\* File: Project 3 - InvalidTreeSyntax Class

\* Author: Dan Beck

\* Date: September 23, 2020

\* Purpose:Class that creates InvalidTreeSyntax error to be caught in program

\*/

public class InvalidTreeSyntax extends RuntimeException

{

private static final long serialVersionUID = 1L;

InvalidTreeSyntax(String msg)

{

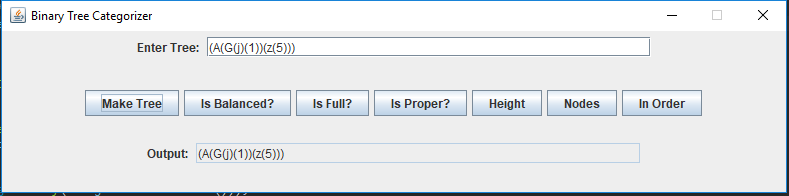
super(msg);

}//end InvalidPolynomialSyntax(String msg)

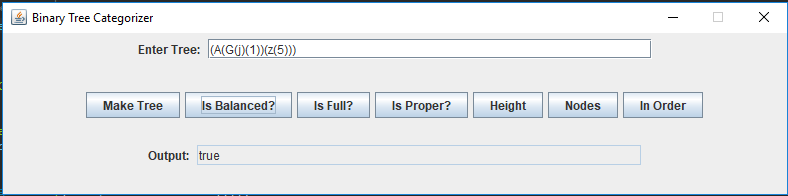
}//end class InvalidPolynomialSyntax

**Output:**

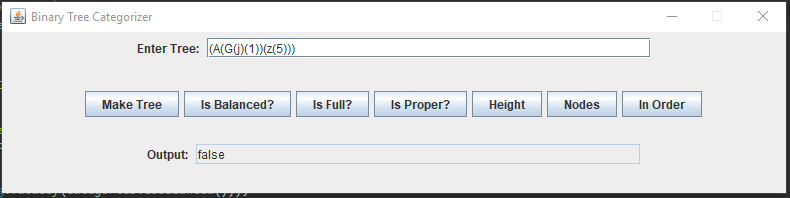
Make Tree



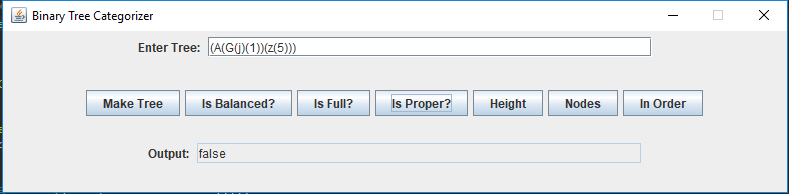
Is Balanced



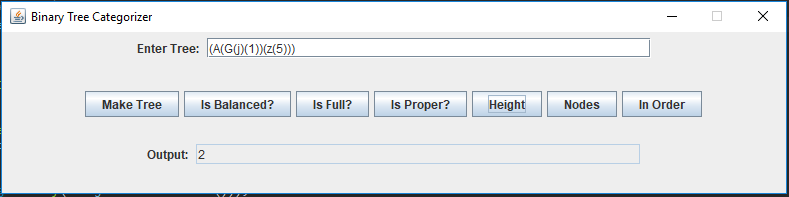
Is Full



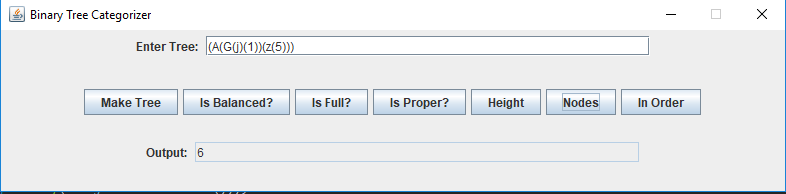
Is Proper



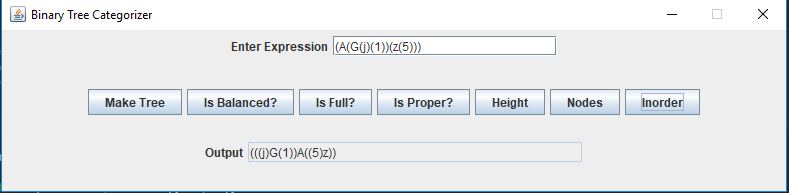
Height



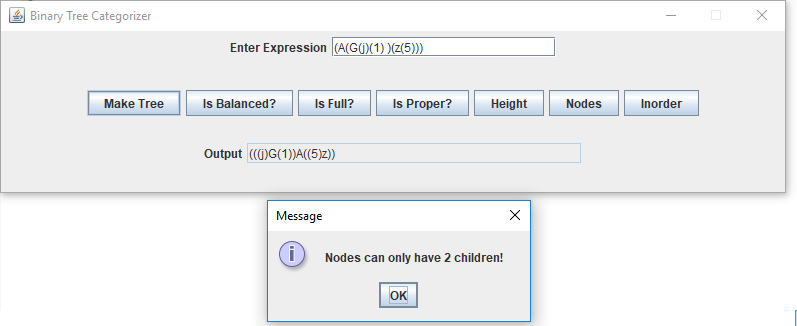
Nodes



In Order



Syntax Error



**Reflection:** I found this project to be much more difficult than the other projects so far as I had no familiarity with binary trees. This project gave me excellent exposure to the binary tree algorithm making me want to dive deeper into how they work.