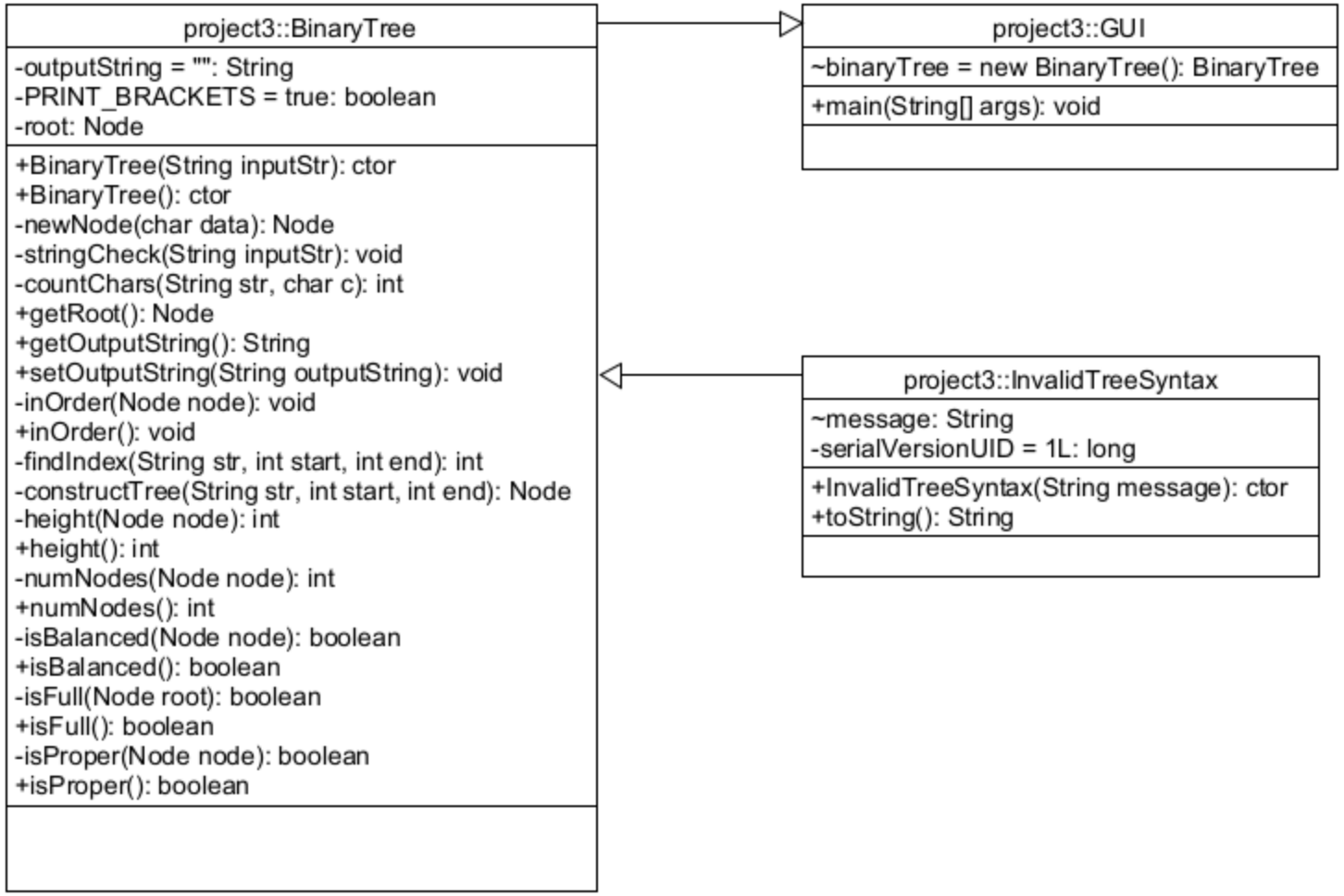
CMIS 350 6382

Project 3

Allen Taylor

2/12/2022

UML Diagram:


project3::BinaryTree
-outputString = "": String
-PRINT_BRACKETS = true: boolean
-root: Node
+BinaryTree(String inputStr): ctor
+BinaryTree(): ctor
-newNode(char data): Node
-stringCheck(String inputStr): void
-countChars(String str, char c): int
+getRoot(): Node
+getOutputString(): String
+setOutputString(String outputString): void
-inOrder(Node node): void
+inOrder(): void
-findIndex(String str, int start, int end): int
-constructTree(String str, int start, int end): Node
-height(Node node): int
+height(): int
-numNodes(Node node): int
+numNodes(): int
-isBalanced(Node node): boolean
+isBalanced(): boolean
-isFull(Node root): boolean
+isFull(): boolean
-isProper(Node node): boolean
+isProper(): boolean
project3::GUI
~binaryTree = new BinaryTree(): BinaryTree
+main(String[] args): void
project3::InvalidTreeSyntax
~message: String
-serialVersionUID = 1L: long
+InvalidTreeSyntax(String message): ctor
+toString(): String

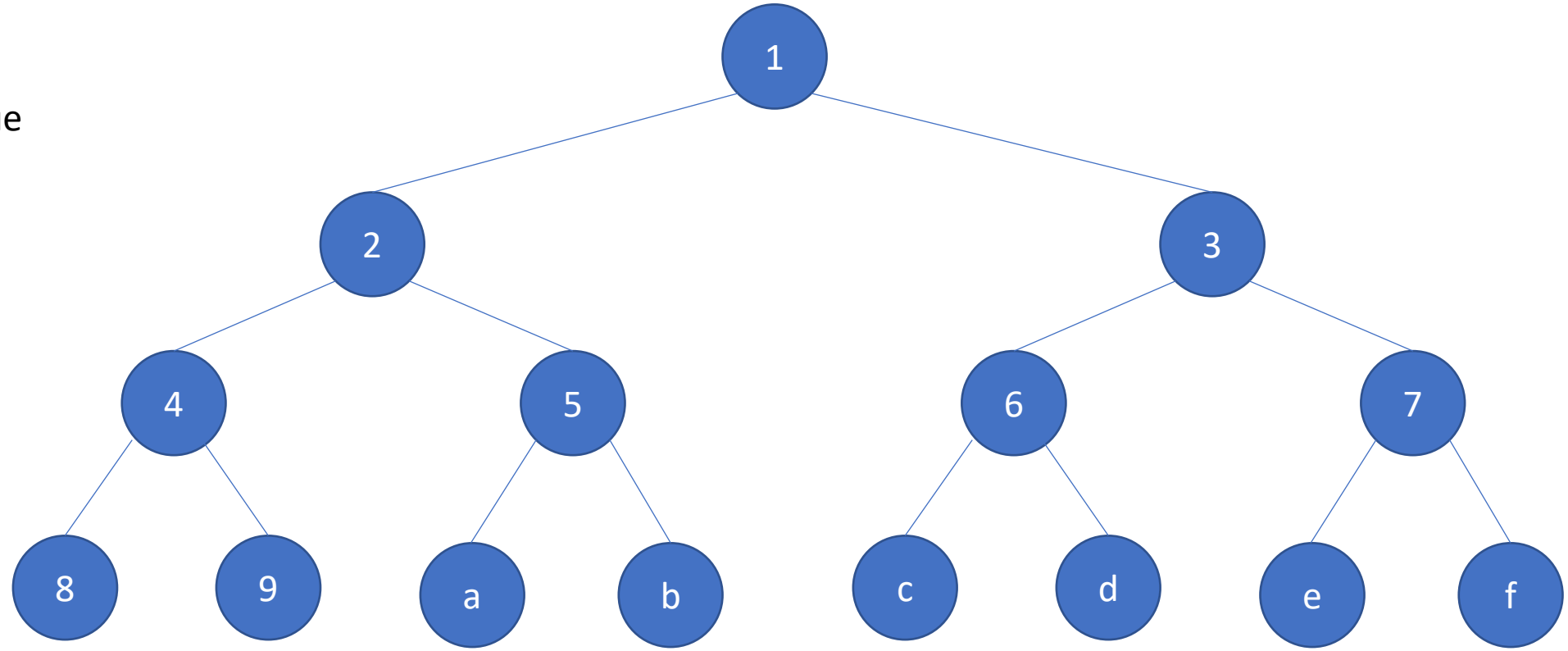
Test Case #1:

Binary Tree String:

(1(2(4(8)(9))(5(a)(b)))(3(6(c)(d))(7(e)(f))))

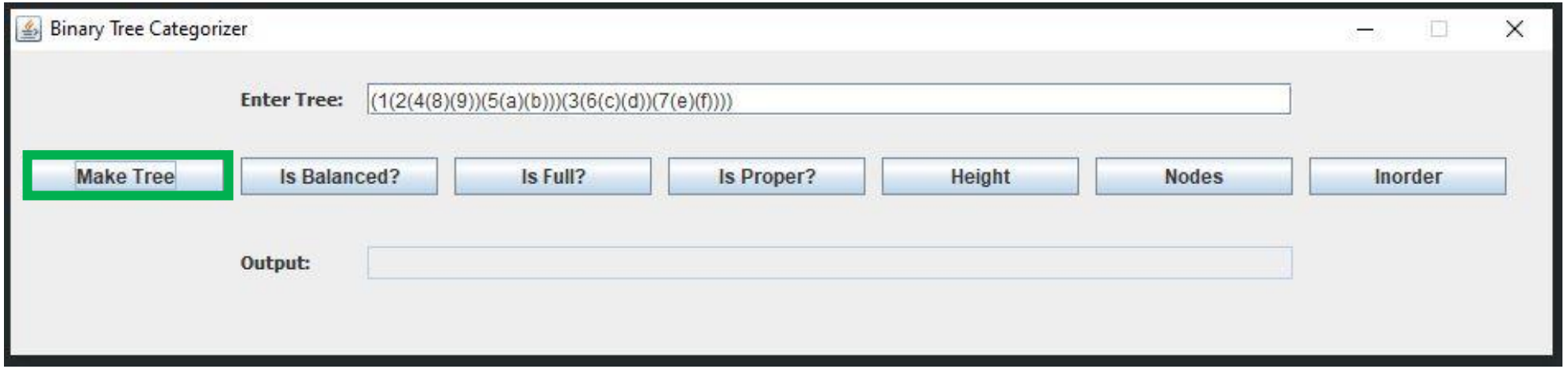
Expected Output:

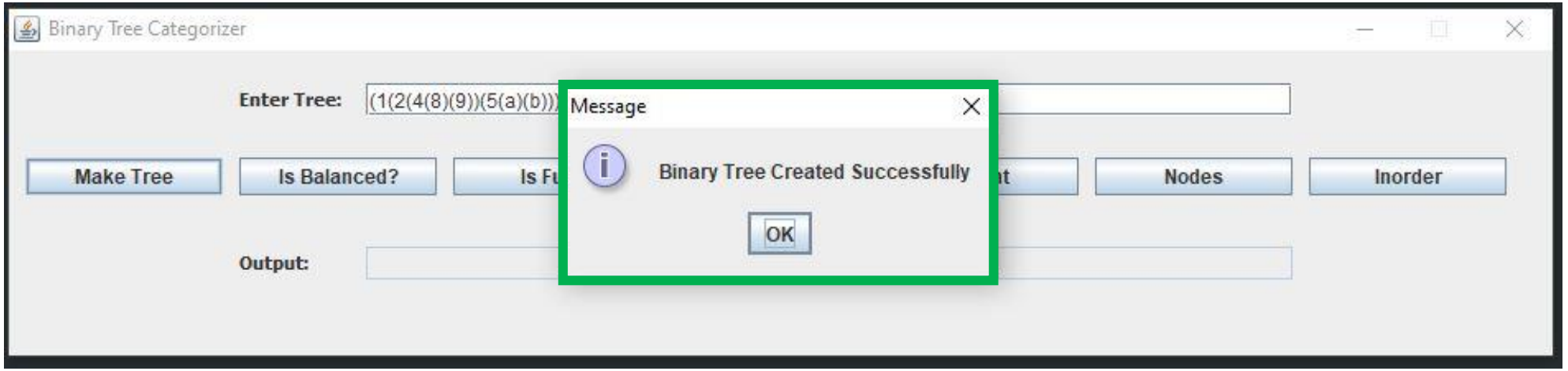
- Balanced = True
- Full = True
- Proper = True
- Height = 3
- Nodes = 11



Test Case #1:

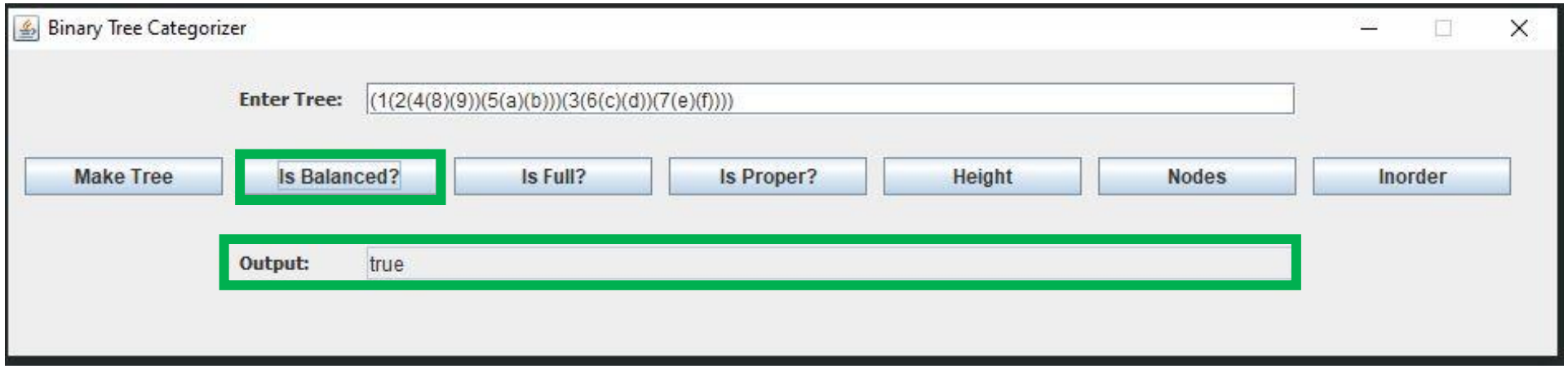
Action: Enter Tree text and click “Make Tree” Button.
Actual Output: JOptionsPane displays “Binary Tree Created Successfully” message with no errors.



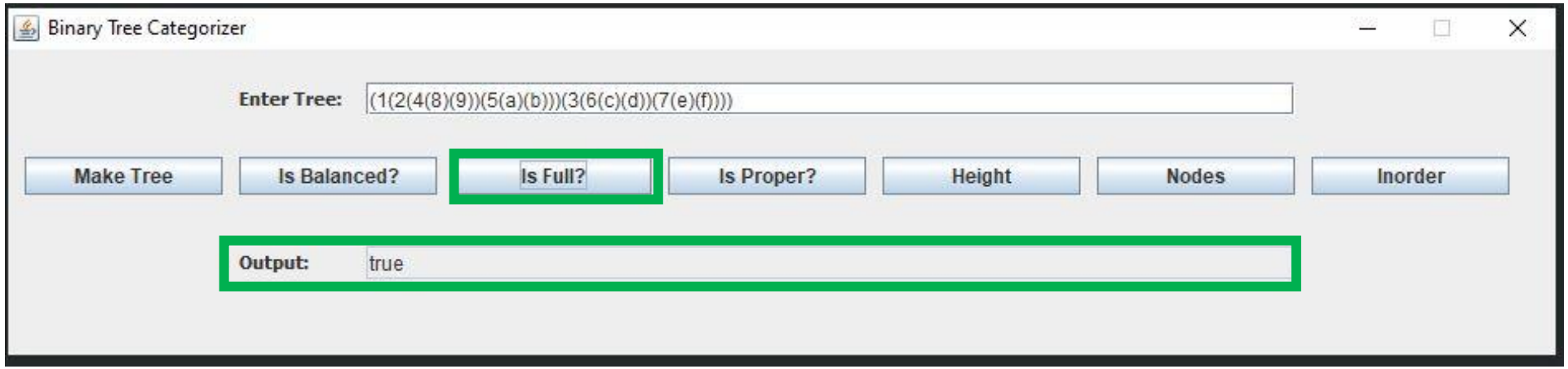


Test Case #1:

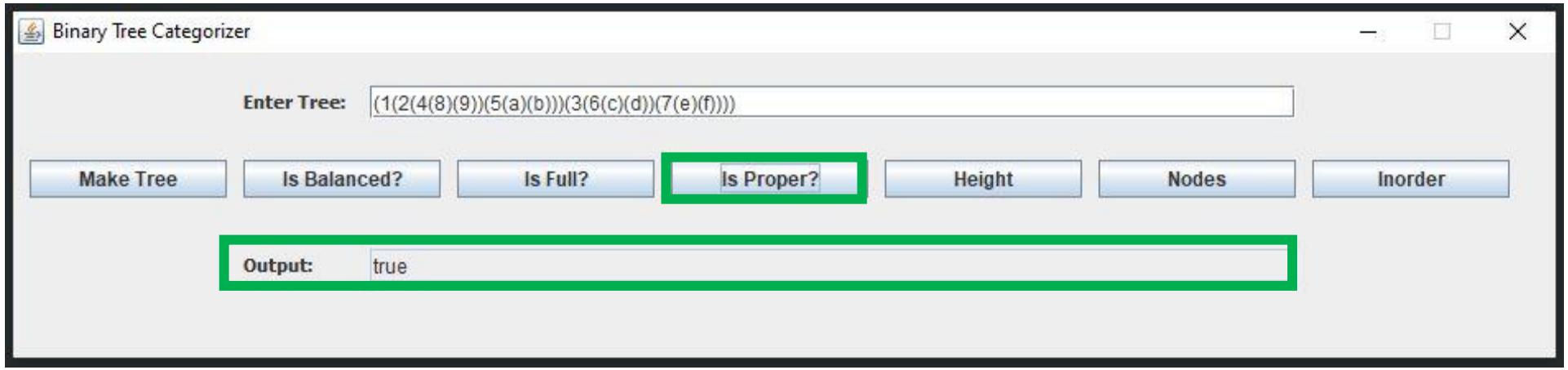
Action:
Click “Is Balanced?” Button.
Actual Output:
Output displays “true”.



Action:
Click “Is Full?” Button.
Actual Output:
Output displays “true”.

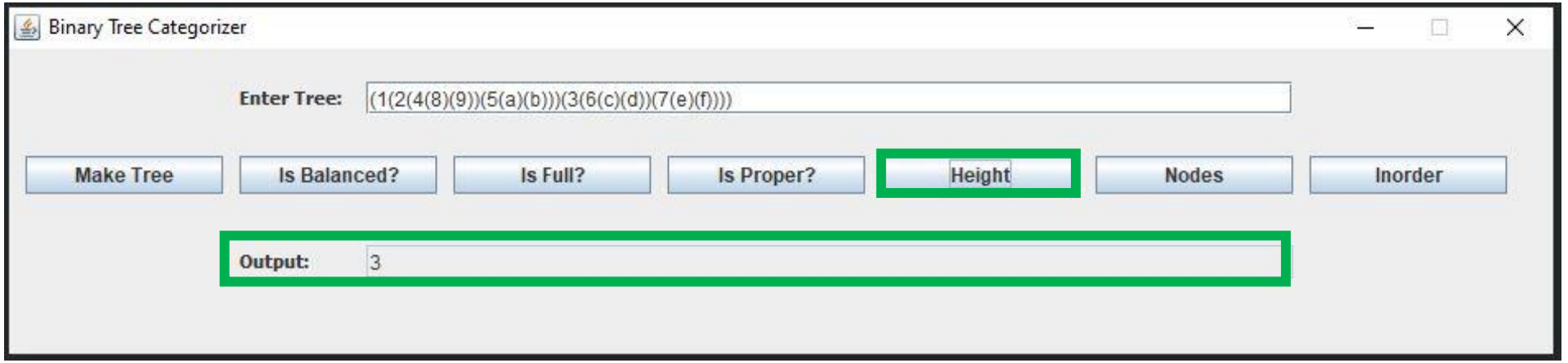


Action:
Click “Is Proper?” Button.
Actual Output:
Output displays “true”.

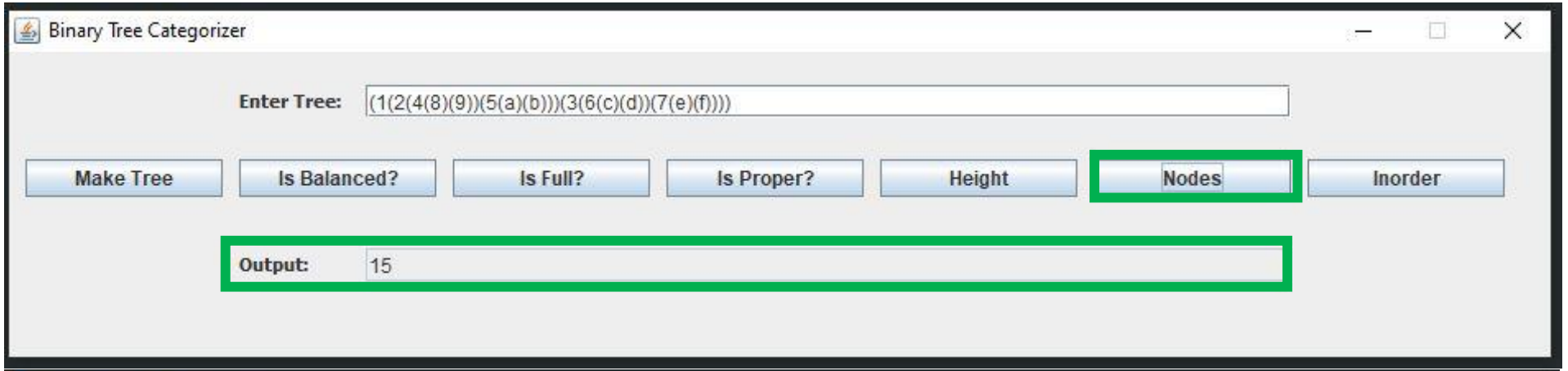


Test Case #1:

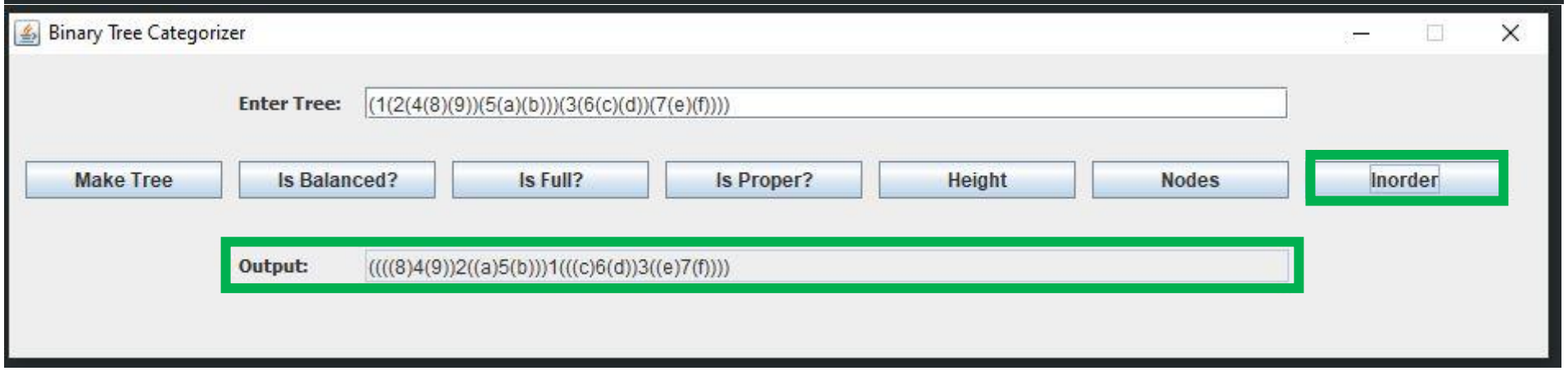
Action:
Click “Height” Button.
Actual Output:
Output displays “3”.



Action:
Click “Nodes” Button.
Actual Output:
Output displays “15”.



Action:
Click “Inorder” Button.
Actual Output:
Output displays
“((((8)4(9))2((a)5(b)))1(((c)6(d))3((e)7(f))))”.

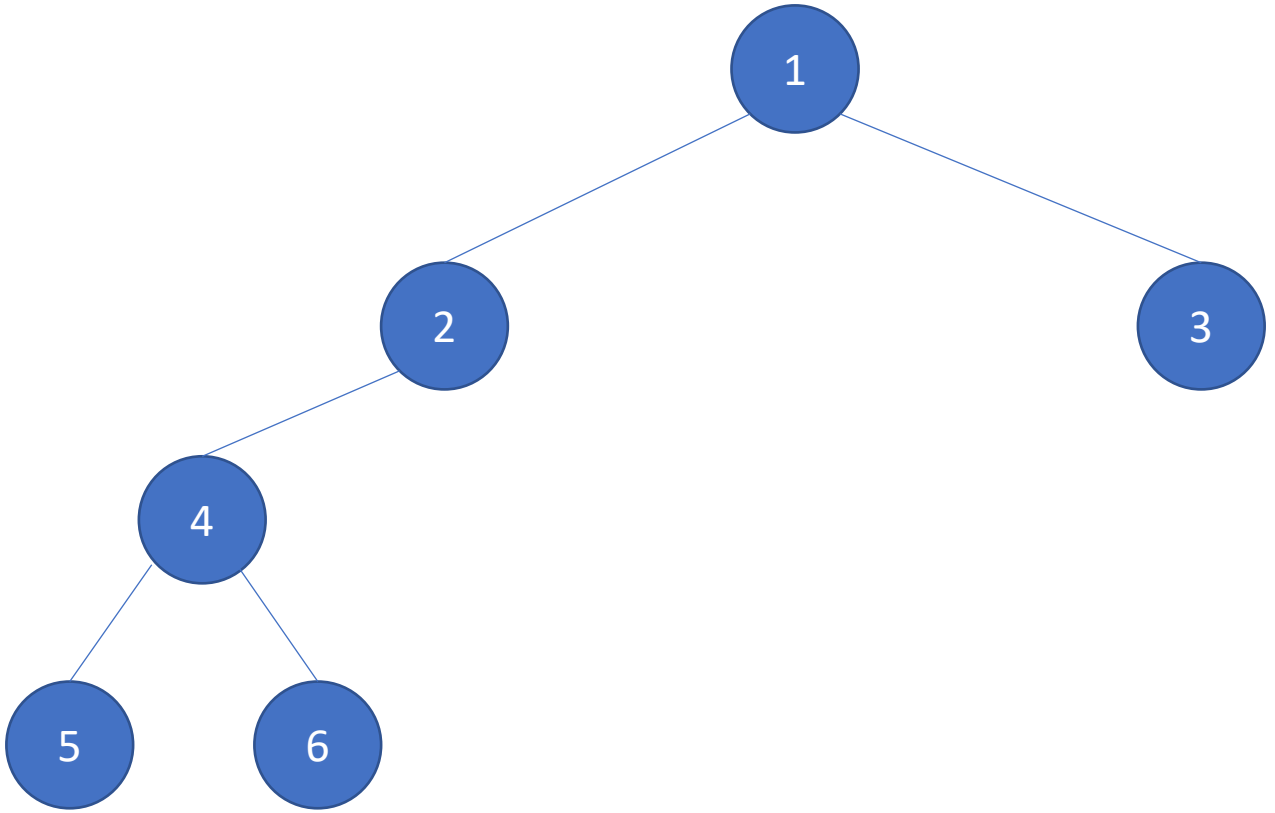


Test Case #2:

Binary Tree String:
(1(2(4(5)(6)))(3))

Expected Output:

- Balanced = False
- Full = False
- Proper = False
- Height = 3
- Nodes = 6

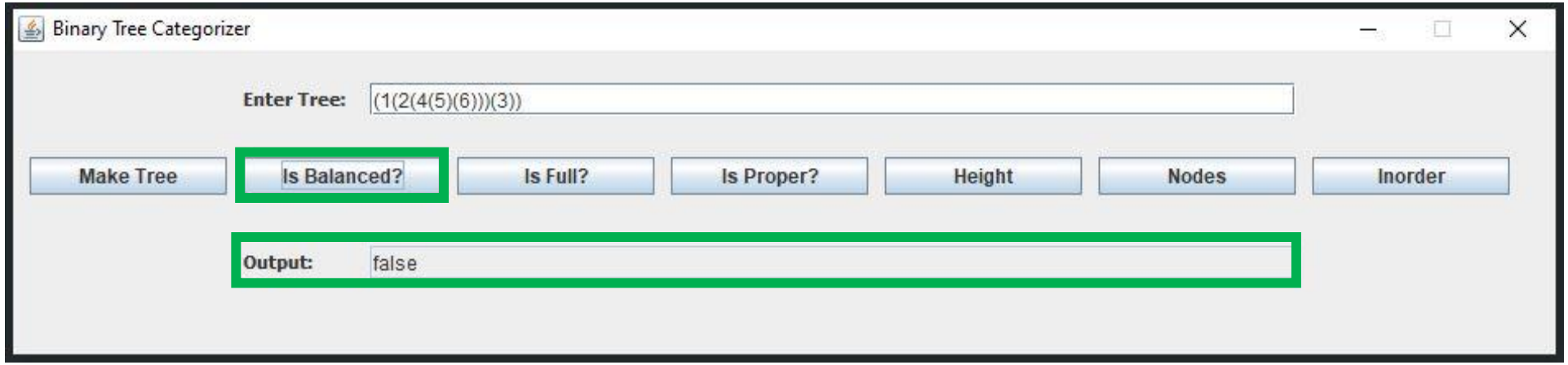


Test Case #2:

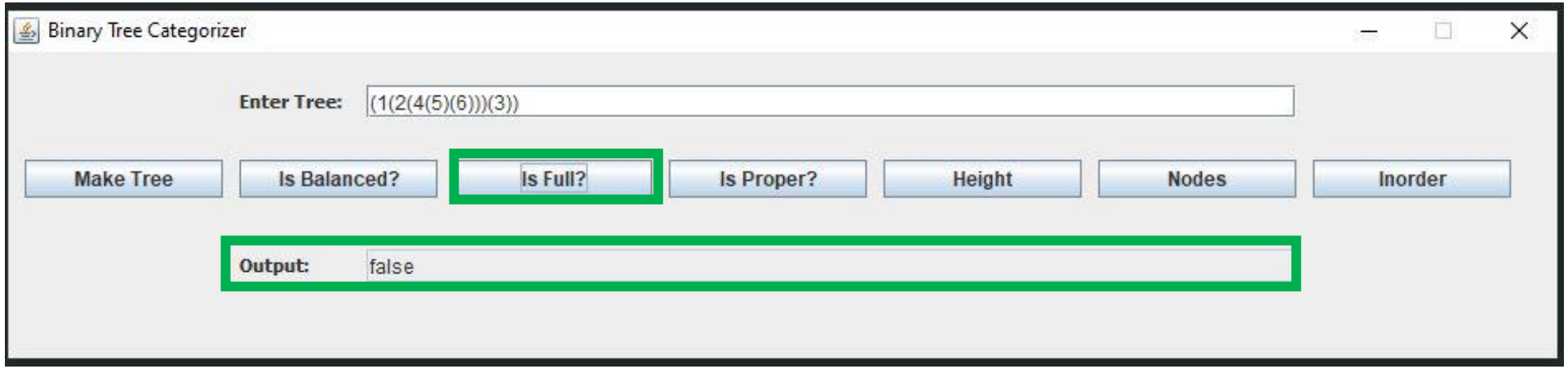
Action: Enter Tree text and click “Make Tree” Button.
Actual Output: JOptionsPane displays “Binary Tree Created Successfully” message with no errors.

Test Case #2:

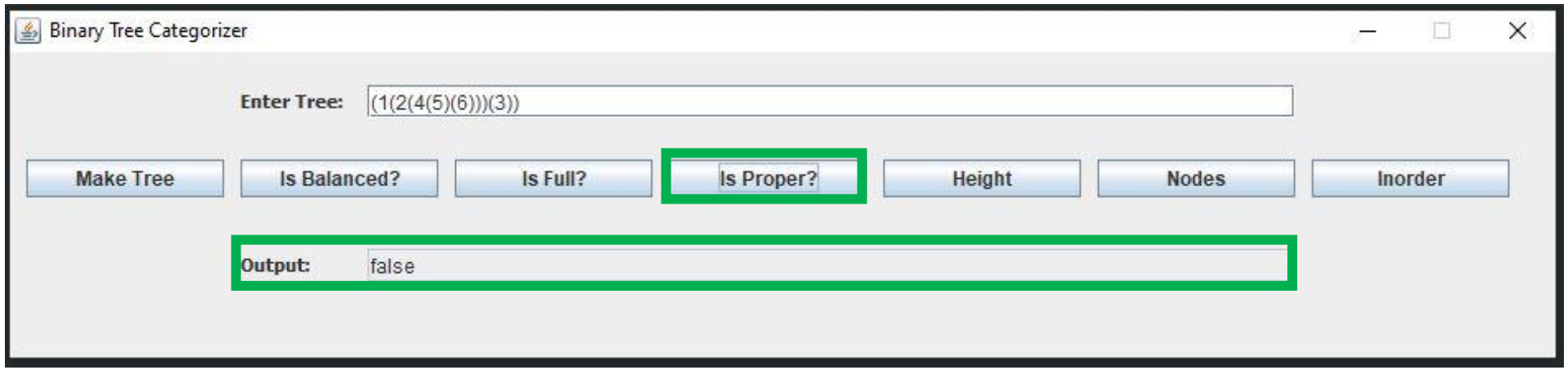
Action:
Click “Is Balanced?” Button.
Actual Output:
Output displays “false”.



Action:
Click “Is Full?” Button.
Actual Output:
Output displays “false”.

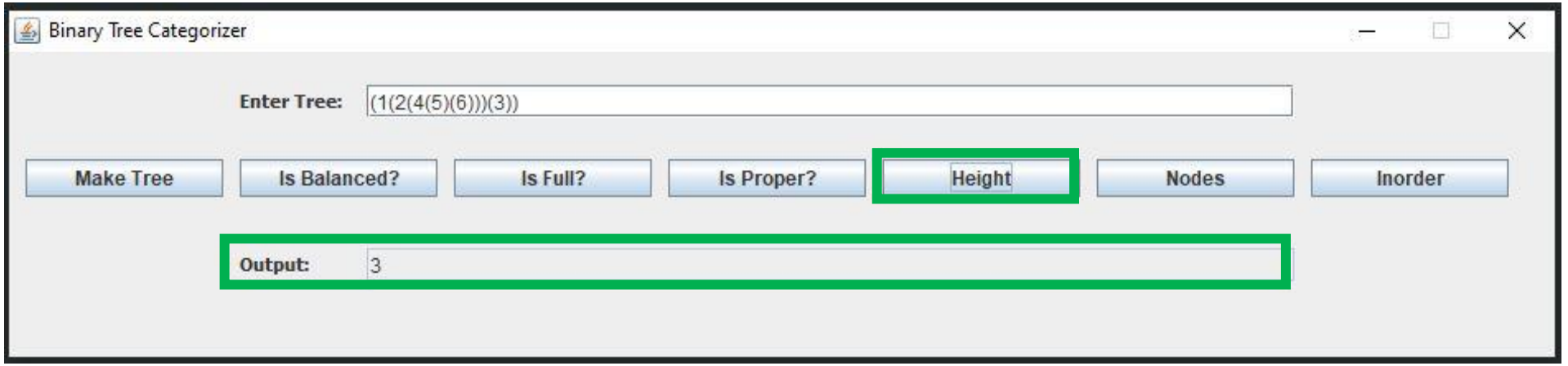


Action:
Click “Is Proper?” Button.
Actual Output:
Output displays “false”.

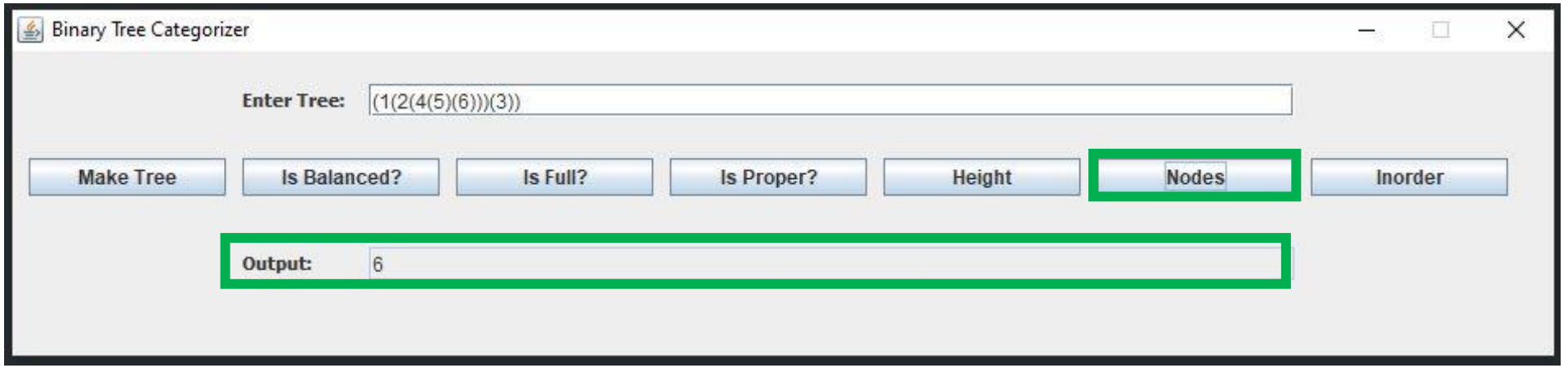


Test Case #2:

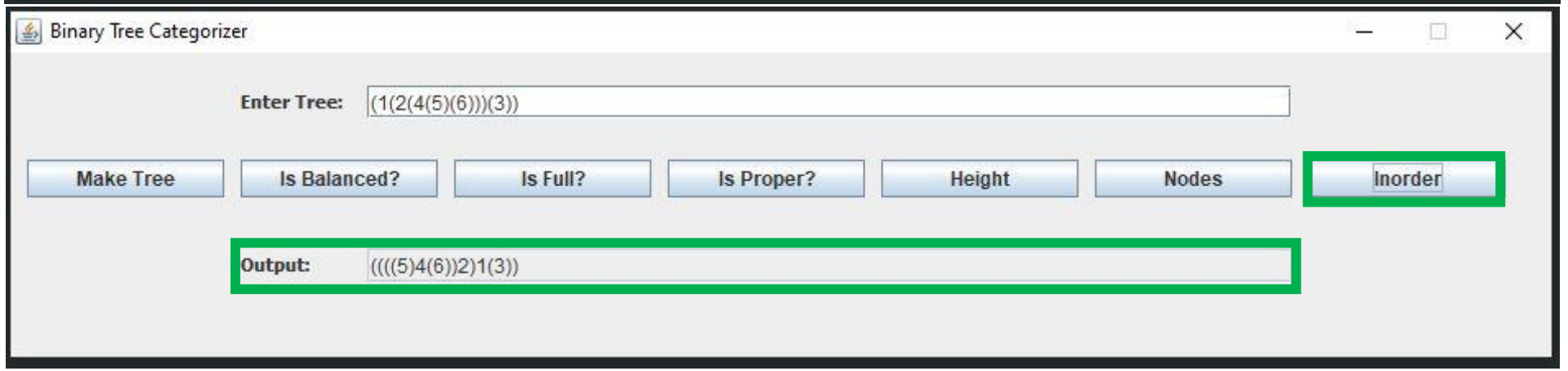
Action:
Click “Height” Button.
Actual Output:
Output displays “3”.



Action:
Click “Nodes” Button.
Actual Output:
Output displays “6”.



Action:
Click “Inorder” Button.
Actual Output:
Output displays “((((5)4(6))2)1(3))”.



Test Case #3:

Invalid Binary Tree Strings:

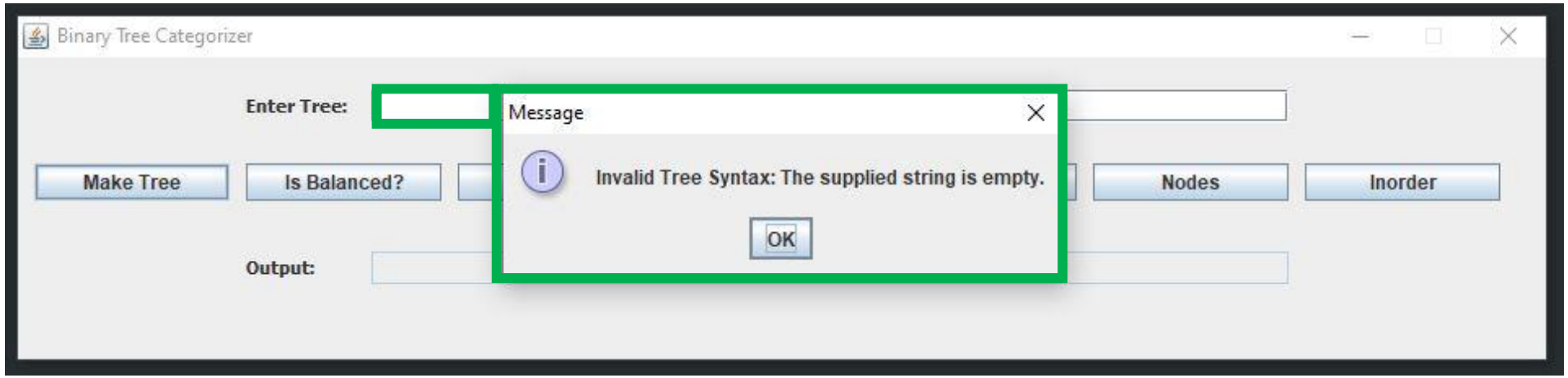
1. <empty>
2. (1(2)(3))) <- Improper brackets
3. ($(2)(3)) <- Invalid Character

Expected Output:

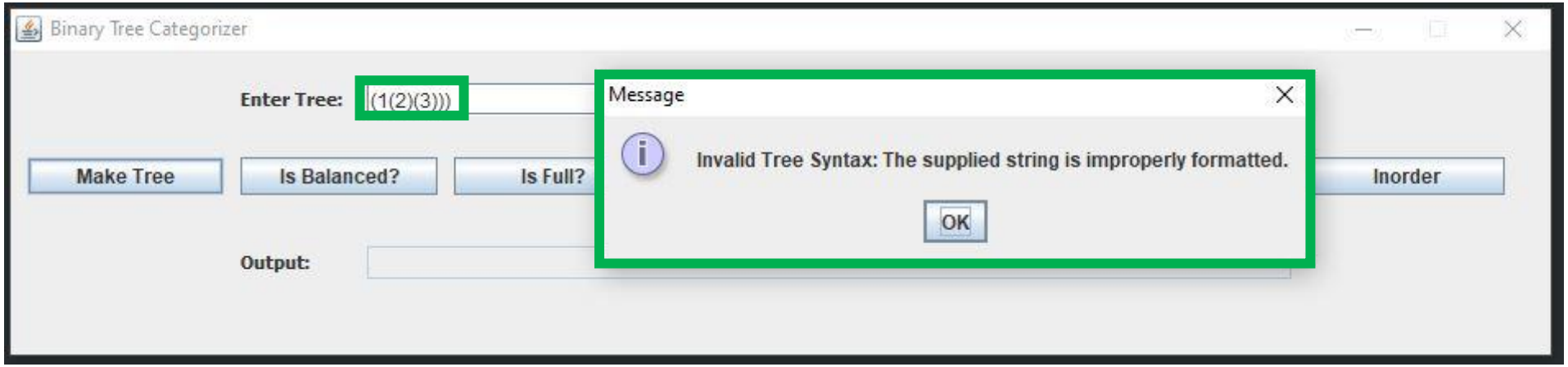
1. Invalid Tree Syntax: The supplied string is empty.
2. Invalid Tree Syntax: The supplied string is improperly formatted.
3. Invalid Tree Syntax: The supplied string contains invalid characters.

Test Case #3:

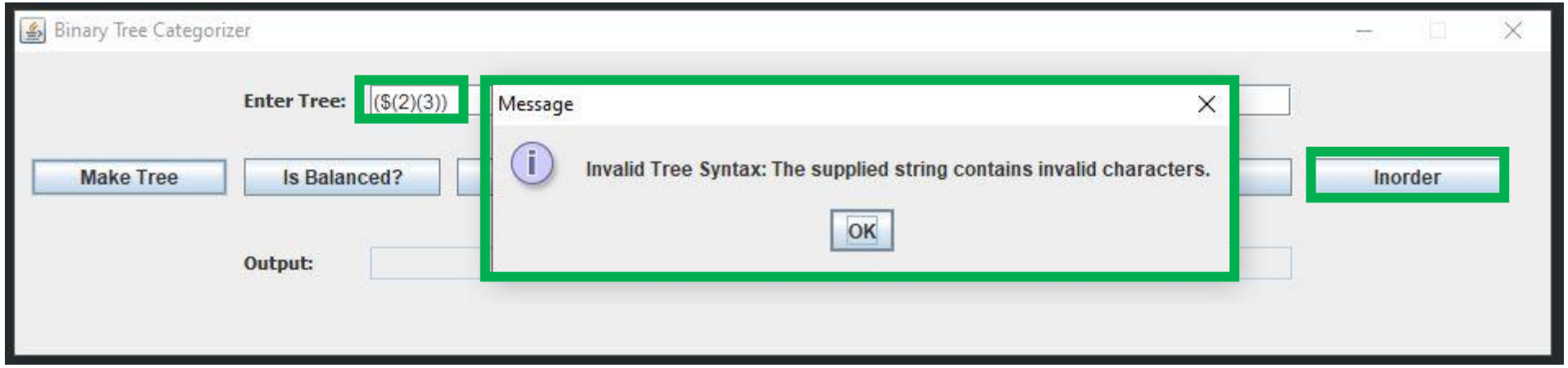
Action:
Enter empty string.
Actual Output:
JOptionPane displays “Invalid Tree Syntax: The supplied string is empty”.



Action:
Enter “(1(2)(3)))”.
Actual Output:
JOptionPane displays “Invalid Tree Syntax: The supplied string is improperly formatted”.



Action:
Enter “($(2)(3))”.
Actual Output:
JOptionPane displays “Invalid Tree Syntax: The supplied string contains invalid characters”.



Lessons Learn:

The most prominent lesson learned from this project was the use of recursion. Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve.

In my code, there are several uses of recursion. Below the **inOrder** method calls itself to iterate the nodes to the left and to the right, all while building the output string that will be displayed on the GUI. This method also employs the user of ternary operators, which are condensed versions of if/else statements and makes them into “one-liners”.

```
private static void inOrder(Node node) {
    if (node == null)
        return;
    outputString = PRINT_BRACKETS ? outputString + "(" : outputString + "";
    inOrder(node.left);
    outputString = outputString + node.data + " ";
    inOrder(node.right);
    outputString = PRINT_BRACKETS ? outputString + ")" : outputString + "";
}

public void inOrder() {
    inOrder(getRoot());
}
```

Overall, recursion is a powerful technique that allows programmers to efficiently code their programs and it was an critical part of this project.