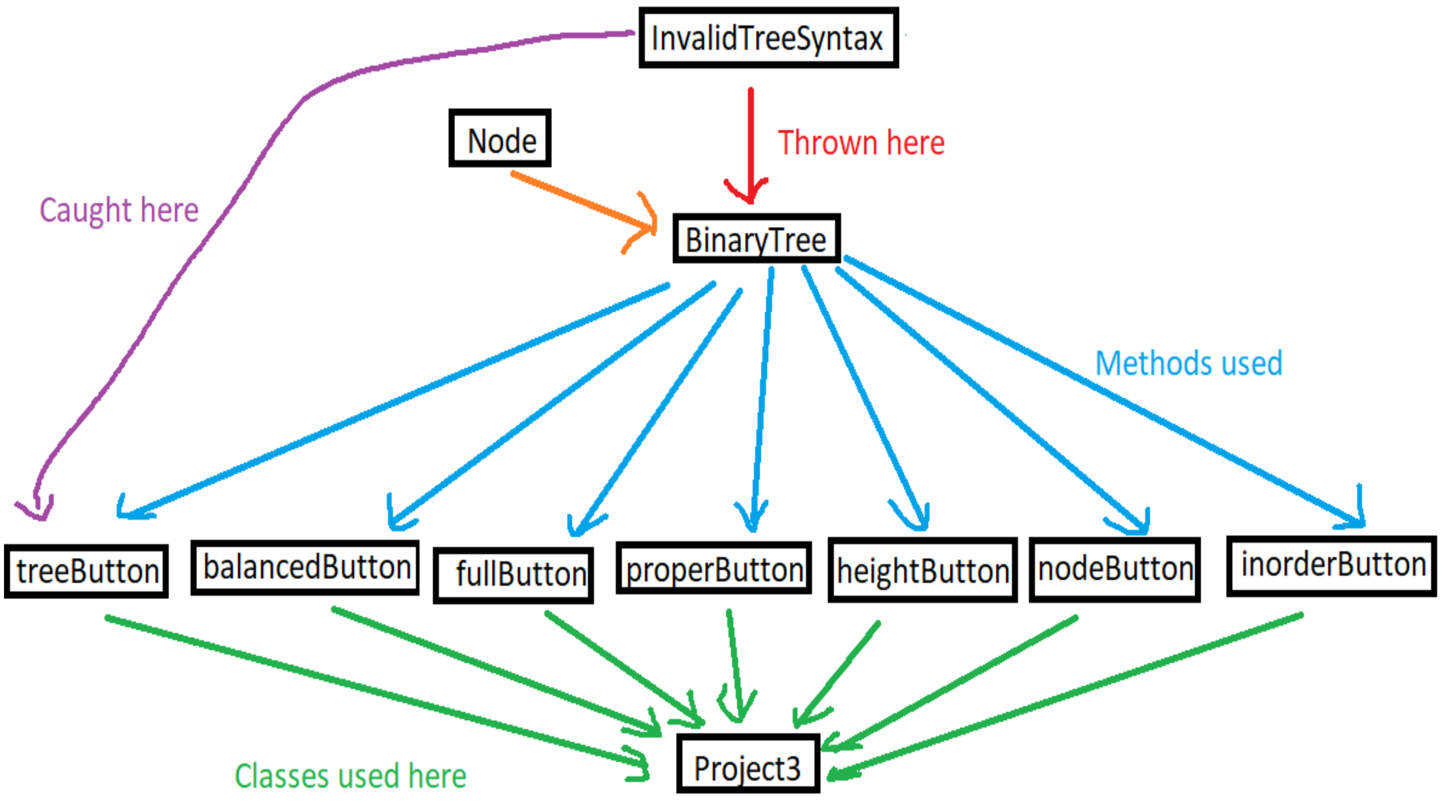
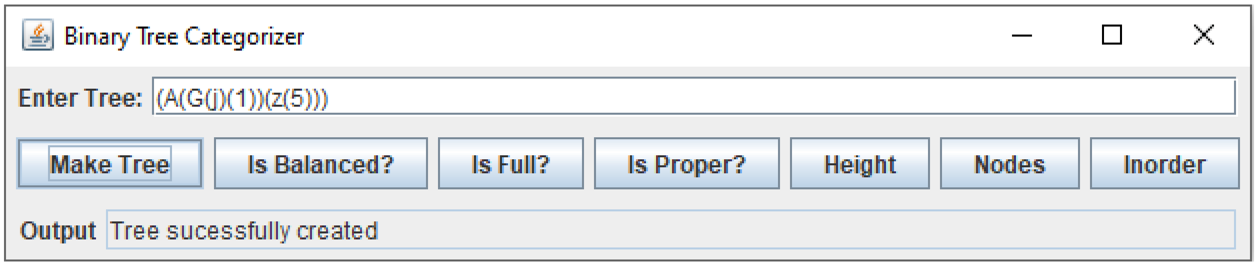


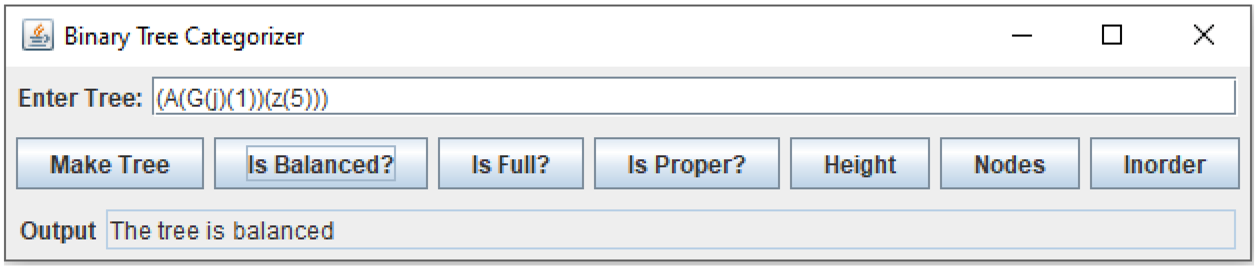
Test Case 1: GUI displayed with no problems, and confirmation upon successful tree creation



Enter Tree: (A(G(j)(1))(z(5)))

Output Tree sucessfully created

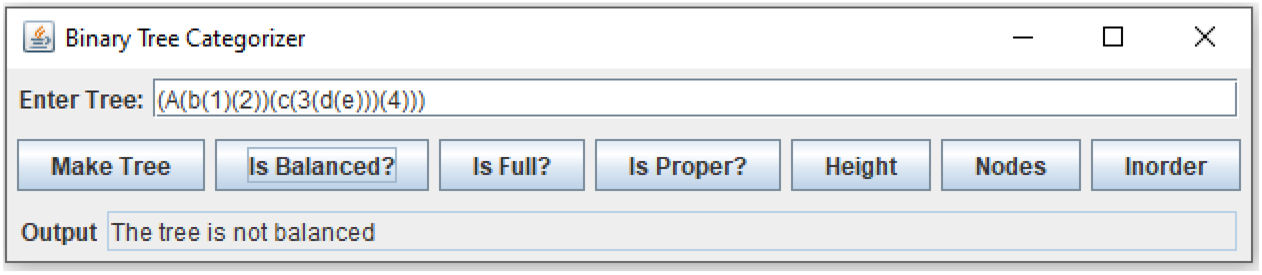
Test Case 2: Balanced tree.



Enter Tree: (A(G(j)(1))(z(5)))

Output The tree is balanced

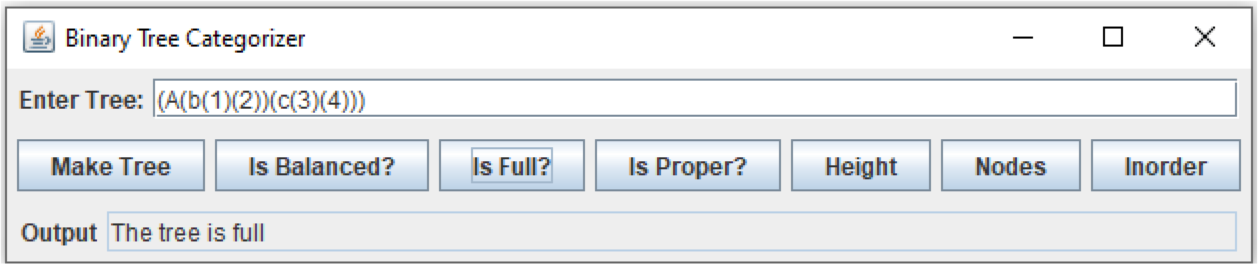
Test Case 3: Unbalanced tree.


Binary Tree Categorizer

Enter Tree: (A(b(1)(2))(c(3(d(e)))(4)))

Output The tree is not balanced

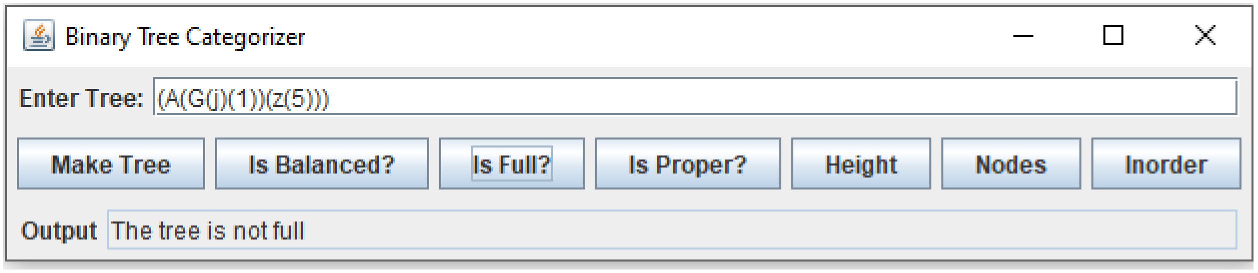
Test Case 4: Full tree.


Binary Tree Categorizer

Enter Tree: (A(b(1)(2))(c(3)(4)))

Output The tree is full

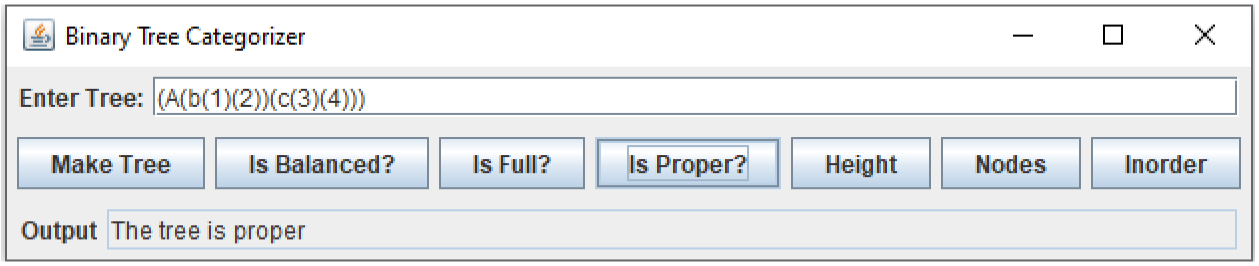
Test Case 5: Not full tree.


Binary Tree Categorizer

Enter Tree: (A(G(j)(1))(z(5)))

Output The tree is not full

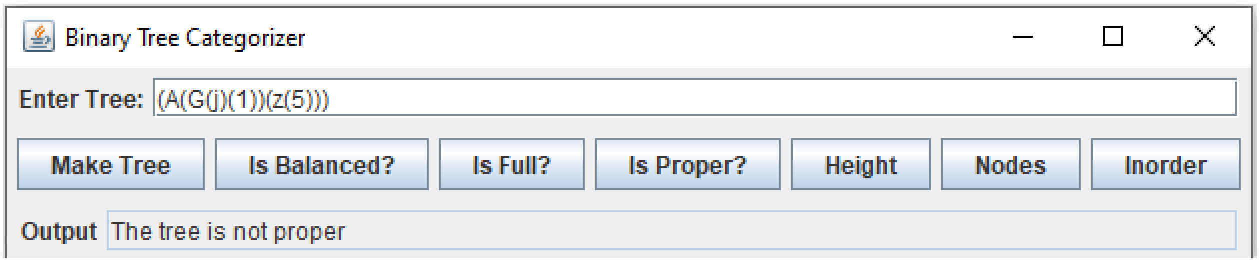
Test Case 6: Proper tree.


Binary Tree Categorizer

Enter Tree: (A(b(1)(2))(c(3)(4)))

Output The tree is proper

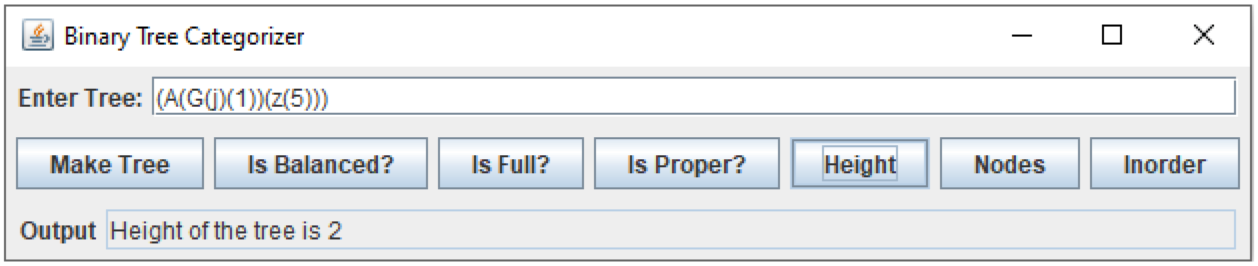
Test Case 7: Not proper tree



Enter Tree: (A(G(j)(1))(z(5)))

Output The tree is not proper

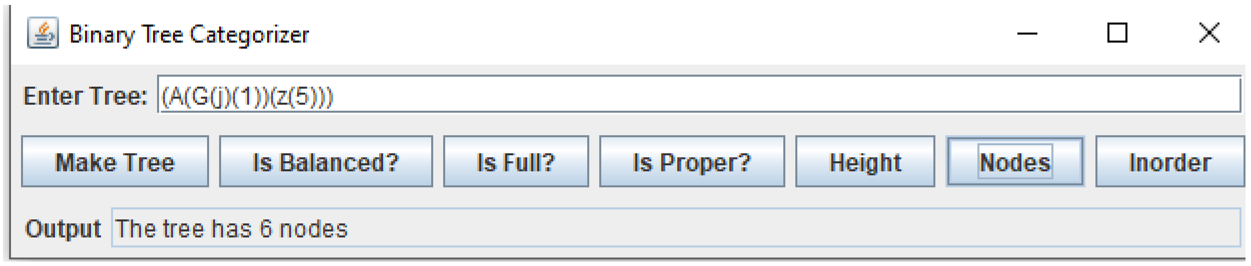
Test Case 8: Program can correctly determine height of tree.



Enter Tree: (A(G(j)(1))(z(5)))

Output Height of the tree is 2

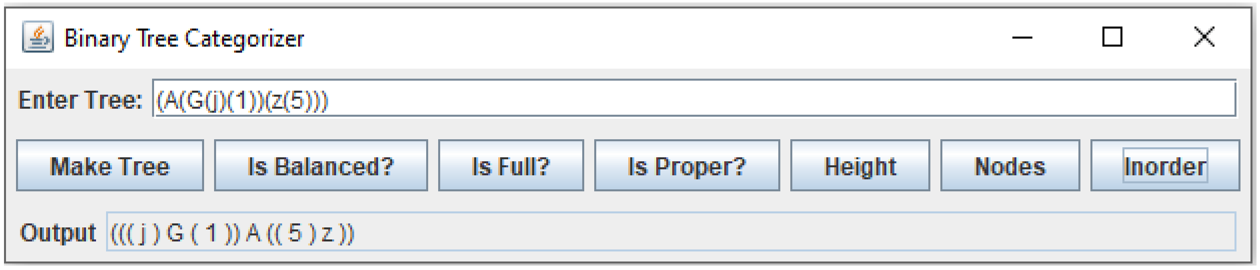
Test Case 9: Program can correctly determine how many nodes are in the tree.



Enter Tree: (A(G(j)(1))(z(5)))

Output The tree has 6 nodes

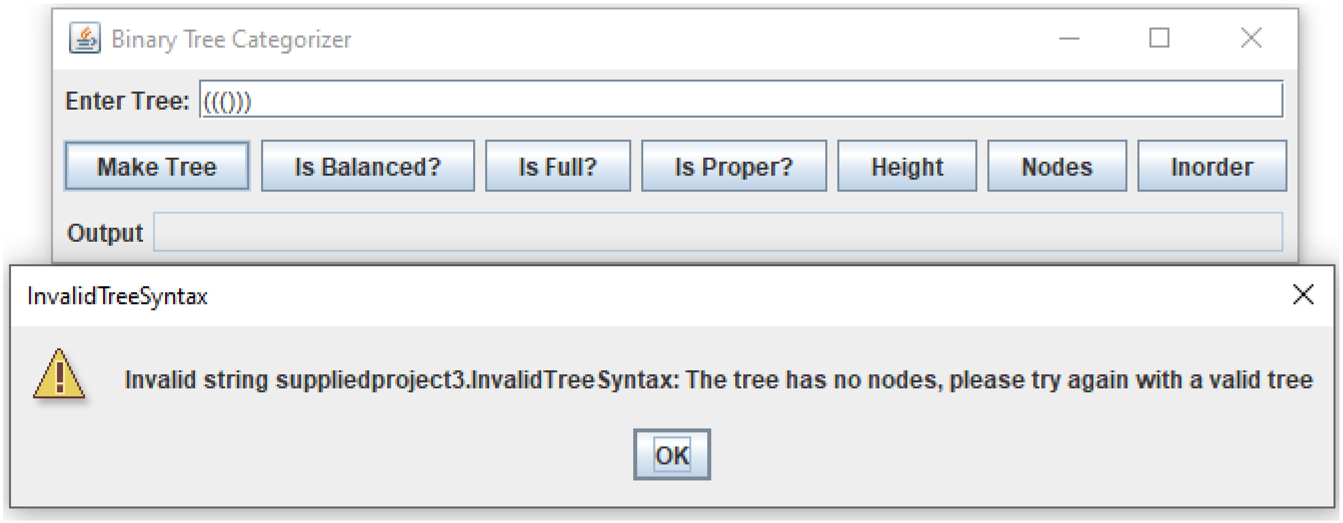
Test Case 10: Program can correctly display the inorder traversal.



Enter Tree: (A(G(j)(1))(z(5)))

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

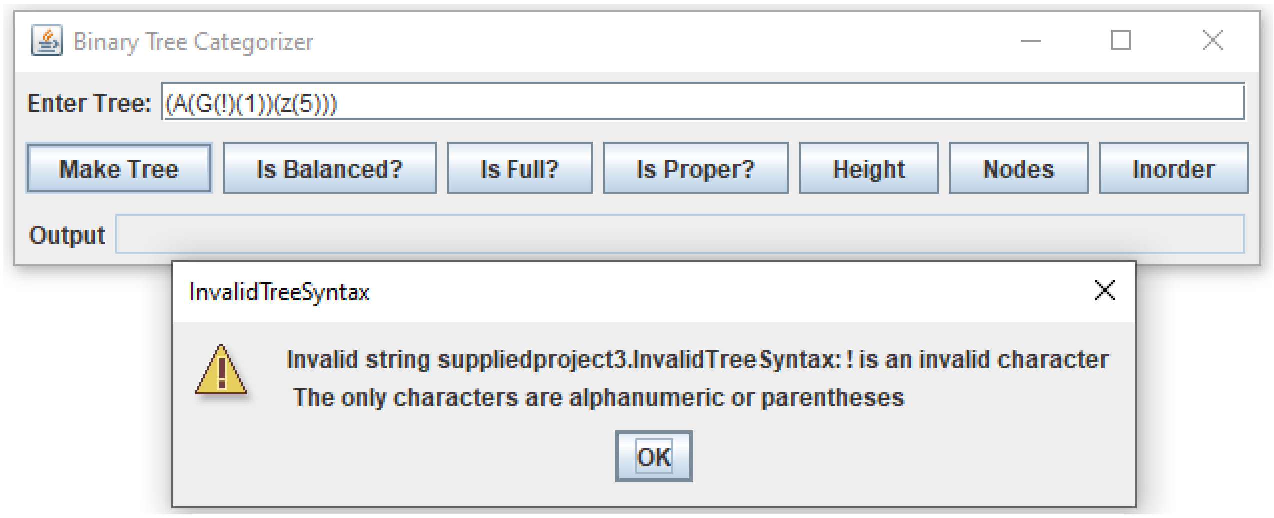
Test Case 11: JOptionPane created when no nodes are present.



Enter Tree: (((()))

Invalid string suppliedproject3.InvalidTreeSyntax: The tree has no nodes, please try again with a valid tree

Test Case 12: JOptionPane created when invalid characters are present for nodes



Enter Tree: (A(G(!)(1))(z(5)))

Invalid string suppliedproject3.InvalidTreeSyntax: ! is an invalid character
The only characters are alphanumeric or parentheses

I learned a lot about recursion and how many different ways it is applicable. Before this project, I'd only used recursion once before, so I'm sure it could've been used better/more efficiently. Because I'm still new to recursion, I had to reference code from other's quite a bit. But like you said, there's no need to reinvent the wheel. The most important part is to fully understand what the code you are referencing does before implementing and adapting it and reference it in the code. Working with recursion has made me understand how indispensable it can be. While it is much more complicated to work with (for now), in the long term I believe it will make things much simpler and concise.