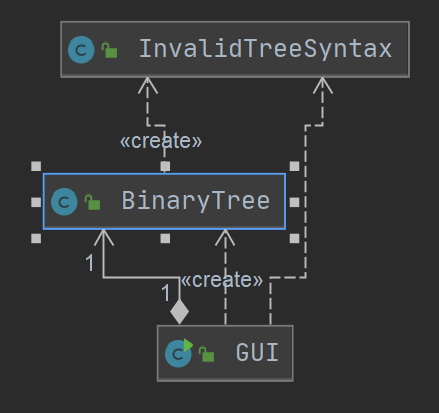
# UML:



# Test cases:

Test Case 1:

* Balanced (run 1 to 3) and unbalanced (run 4 to 6) trees:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Run | Input | Expected Output | Actual Output | Pass/Fail |
| 1 | (F(B)(1(C)(4))) | Make Tree: No Error  Is Balanced: true  Height: 3  Nodes: 5  In Order: ((B)F((C)1(4))) | Make Tree: No Error  Is Balanced: true  Height: 3  Nodes: 5  In Order: ((B)F((C)1(4))) | Pass |
| 2 | (B(1(A)(D))(2(3)(4))) | Make Tree: No Error  Is Balanced: true  Height: 3  Nodes: 7  In Order: (((A)1(D))B((3)2(4))) | Make Tree: No Error  Is Balanced: true  Height: 3  Nodes: 7  In Order: (((A)1(D))B((3)2(4))) | Pass |
| 3 | (A(B(2(1))(8))(C(4)(5))) | Make Tree: No Error  Is Balanced:  Height: 4  Nodes: 8  In Order: ((((1)2)B(8))A((4)C(5))) | Make Tree: No Error  Is Balanced:  Height: 4  Nodes: 8  In Order: ((((1)2)B(8))A((4)C(5))) | Pass |
| 4 | (F(A)(G(B)(H(I)))) | Make Tree: No Error  Is Balanced: false  Height: 4  Nodes: 6  In Order: ((A)F((B)G((I)H))) | Make Tree: No Error  Is Balanced: false  Height: 4  Nodes: 6  In Order: ((A)F((B)G((I)H))) | Pass |
| 5 | (F(E(1(2)))(B)) | Make Tree: No Error  Is Balanced: false  Height: 4  Nodes: 5  In Order: ((((2)1)E)F(B)) | Make Tree: No Error  Is Balanced: false  Height: 4  Nodes: 5  In Order: ((((2)1)E)F(B)) | Pass |
| 6 | (F(A(1)(2))(B(f(1(c)))(g))) | Make Tree: No Error  Is Balanced: false  Height: 5  Nodes: 9  In Order: (((1)A(2))F((((c)1)f)B(g))) | Make Tree: No Error  Is Balanced: false  Height: 5  Nodes: 9  In Order: (((1)A(2))F((((c)1)f)B(g))) | Pass |

Test Case 2:

* Full (run 1 & 2) and not full (run 3 & 4) trees:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Run | Input | Expected Output | Actual Output | Pass/Fail |
| 1 | (A(B)(C)) | Make Tree: No Error  Is full: true  Height: 2  Nodes: 3  In Order: ((B)A(C)) | Make Tree: No Error  Is full: true  Height: 2  Nodes: 3  In Order: ((B)A(C)) | Pass |
| 2 | (8(A(B)(C))(9(2)(1))) | Make Tree: No Error  Is Full: true  Height: 3  Nodes: 7  In Order: (((B)A(C))8((2)9(1))) | Make Tree: No Error  Is Full: true  Height: 3  Nodes: 7  In Order: (((B)A(C))8((2)9(1))) | Pass |
| 3 | (A(B)) | Make Tree: No Error  Is Balanced: false  Height: 2  Nodes: 2  In Order: ((B)A) | Make Tree: No Error  Is Balanced: false  Height: 2  Nodes: 2  In Order: ((B)A) | Pass |
| 4 | (8(A(C))(9(2)(1))) | Make Tree: No Error  Is Balanced: false  Height: 3  Nodes: 6  In Order: (((C)A)8((2)9(1))) | Make Tree: No Error  Is Balanced: false  Height: 3  Nodes: 6  In Order: (((C)A)8((2)9(1))) | Pass |

Test Case 3:

* Proper (run 1 & 2) and improper (run 3 & 4) trees:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Run | Input | Expected Output | Actual Output | Pass/Fail |
| 1 | (A(B)(C(1)(2))) | Make Tree: No Error  Is Proper: true  Height: 3  Nodes: 5  In Order: ((B)A((1)C(2))) | Make Tree: No Error  Is Proper: true  Height: 3  Nodes: 5  In Order: ((B)A((1)C(2))) | Pass |
| 2 | (8(A)(9(2)(1))) | Make Tree: No Error  Is Proper: true  Height: 5  Nodes: 3  In Order: ((A)8((2)9(1))) | Make Tree: No Error  Is Proper: true  Height: 5  Nodes: 3  In Order: ((A)8((2)9(1))) | Pass |
| 3 | (F(A(1)(2))(B(f(1(c)))(g))) | Make Tree: No Error  Is Proper: false  Height: 5  Nodes: 9  In Order: (((1)A(2))F((((c)1)f)B(g))) | Make Tree: No Error  Is Proper: false  Height: 5  Nodes: 9  In Order: (((1)A(2))F((((c)1)f)B(g))) | Fail |
| 4 | (F(A)(G(B)(H(I)))) | Make Tree: No Error  Is Proper:  Height: 4  Nodes: 6  In Order: ((A)F((B)G((I)H))) | Make Tree: No Error  Is Proper:  Height: 4  Nodes: 6  In Order: ((A)F((B)G((I)H))) | Fail |

Test Case 4:

* Syntax Errors:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Run | Input | Expected Output | Actual Output | Pass/Fail |
| 1 | F(B)(1(C)(4))) | Make Tree: Syntax Error | Syntax Error popup window. | Pass |
| 2 | (B(1(A(D))(2(3)(4)) | Make Tree: Syntax Error | Syntax Error popup window. | Pass |
| 3 | (A (B)(C(1)(2))) | Make Tree: Syntax Error | Syntax Error popup window. | Pass |

# Lessons Learned:

* The most prevalent issue I can tell that I have is error checking. I get in a time crunch, code without even thinking of how I will check for errors, and then at the last-minute figure it out. I think error checking could be much easier if I started coding with it in mind, because I would see that the path I am going down will make error checking in that code block more difficult later. For this reason, I ended up with an inelegant way of checking for errors and most likely not checking for all the errors the program would be thrown. However, the program does handle improper parentheses and also handles random characters in between parentheses ( as in (A (B)asdaf(C(1)(2))), but not by error checking, it just ignores these mistakes.
* This program also does not handle more than one digit nodes. I should have thought of this before, but just like error checking, a mistake I make often is not thinking of all the cases the program will face.

## Extra note

These mistakes that I commonly make, along with the others that I can not see the impact of immediately, worry me. I am used to environments where mistakes get people hurt in my previous jobs, and remembering these experiences causes me to wonder what a mistake in programming (especially error checking) could cause. I am starting to worry more about things like this because I need to break the habit. What happens when I go out into the workforce with a CS and EE degree and an employer has me work both sides of some robotics project and the programming side of said project doesn’t check for some circumstance, moves out of control, and hits somebody? I apologize if this extra lessons learned point is too much, I keep all of my notes and projects for future reference and sometimes like to include things to look back on.