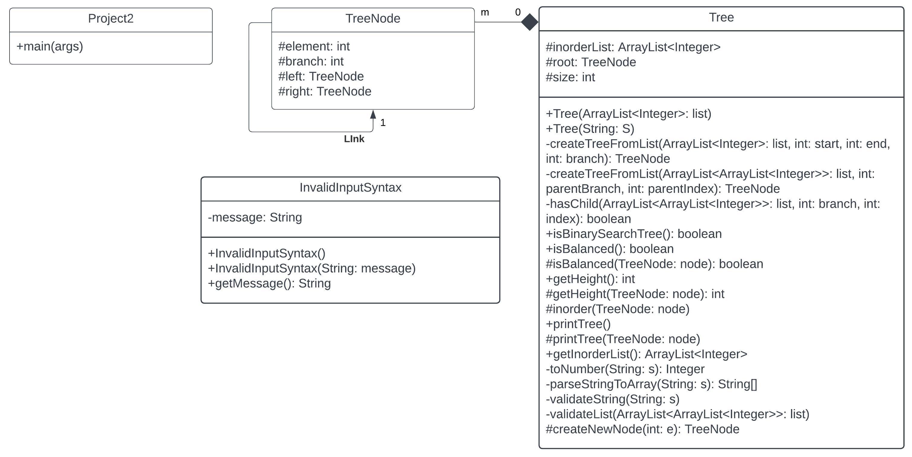
CMSC 315

Programming Project 3 – Binary Search Trees

Daniel Smolsky

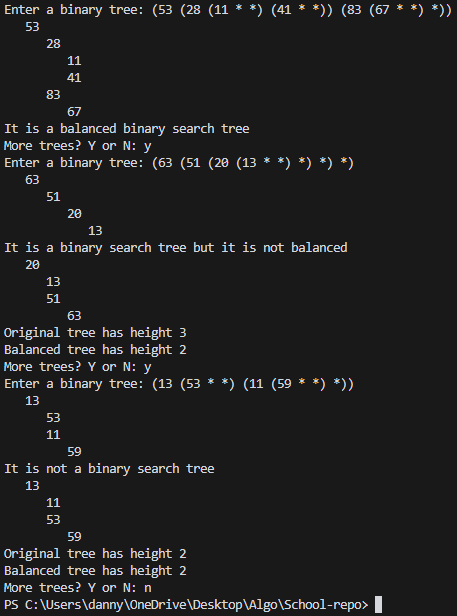
**UML Diagram:**



**Test Plan:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test # | Purpose | Pos/Neg Test | Input Values | Expected Result | Pass/Fail |
| 1 | Check program output matches examples | Pos | (53 (28 (11 \* \*) (41 \* \*)) (83 (67 \* \*) \*))  (63 (51 (20 (13 \* \*) \*) \*) \*)  (13 (53 \* \*) (11 (59 \* \*) \*)) | Same as program guidelines | pass |
| 2 | Check invalid string inputs return exceptions with description of the issue | Pos | (53 (28 (11 \* \*) (41 \* \*)) (83 (67 \* \*))) | Branch 2 is missing nodes | pass |
| 3 | Check invalid string inputs return exceptions with description of the issue | Pos | (53 (28 (11 \* \*) (41 \* \*)) (83 (r \* \*) \*)) | Data is not an integer | pass |
| 4 | Check invalid string inputs return exceptions with description of the issue | Pos | (63 (51 (20 (13 \* \*) \*) \*) \* | Missing right parenthesis | pass |
| 5 | Check trees with no left subtrees are draw correctly | Pos | (53 \* (83 (67 \* \*) \*)) | Unbalanced tree is drawn | pass |

Test plan 1:

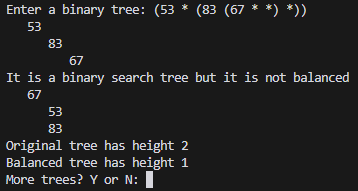


Test plan 2:



Test plan 3:  


Test plan 4:  


Test plan 5:  


**Lessons Learned:**

Most of the methods in my tree class were recursive, and originally it was hard to follow the stack execution as recursive methods would call other recursive methods. I significantly improved on my recursive logic programming skills due to this, and as a result my binary search tree creation process is clean and efficient. Additionally, I improved on my exception handling, string parsing and validation, and documentation skills. One goal I had for this project was for my code to be easily understandable by someone looking at it their first time, and I feel due to all of the descriptive comments I’ve accomplished this.