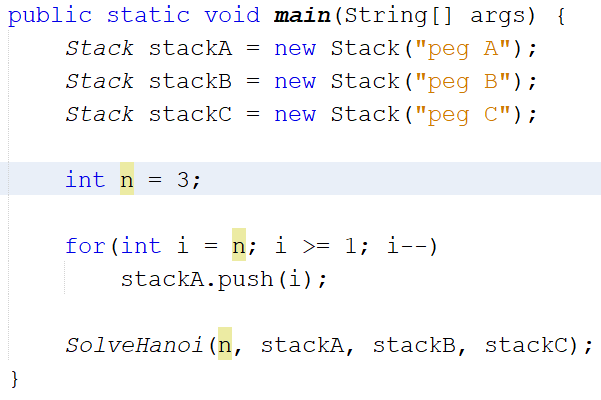
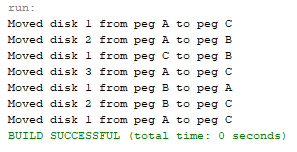
**Program 1**

This program allows for the Tower of Hanoi puzzle to be solved recursively.

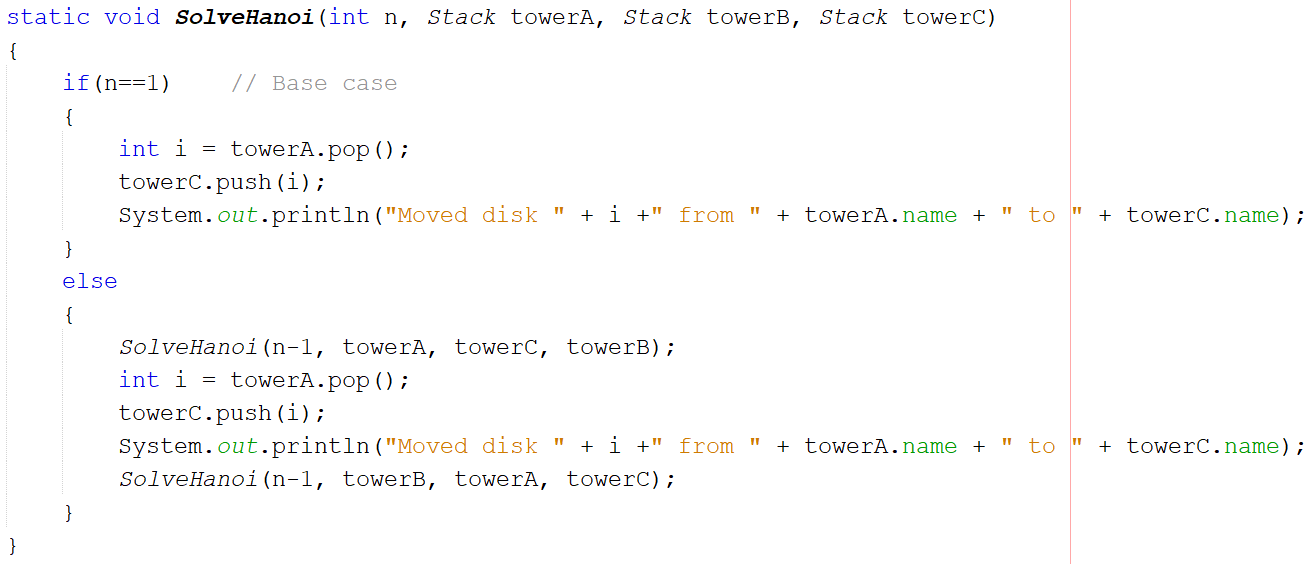




The program requires the following *Stack* class.

|  |  |
| --- | --- |
| Class | Stack |
| Fields | data, name |
| Behaviors | push(i), pop() |

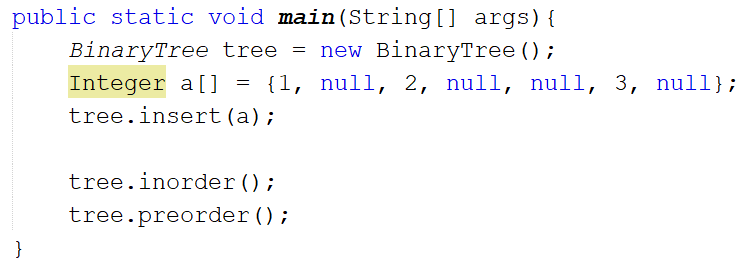
3 stacks must be constructed, and the first stack must have n discs. Then, the *SolveHanoi()* method may be called.



*SolveHanoi()* calls itself recursively until n is equal to 1. At this base case, a disk is moved from towerA to towerC. Otherwise, a recursive call is made to move a disk from towerA to towerB, a disk is moved from towerA to towerC, and a recursive call is made to move a disk from towerB to towerC.

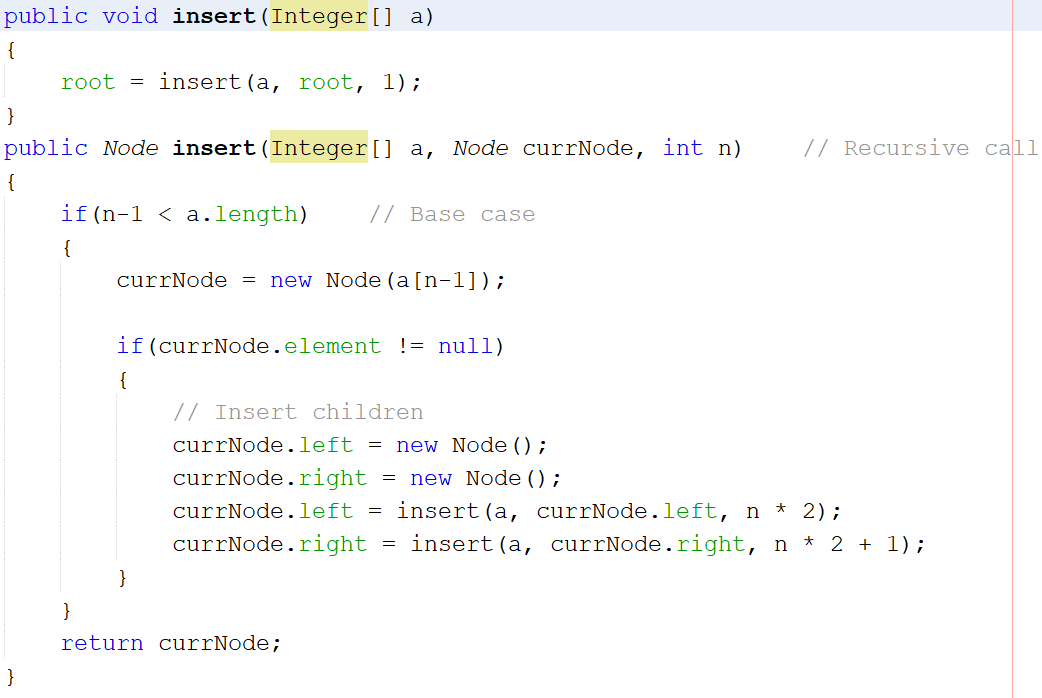
**Program 2**

This program allows for the creation and traversal of a binary tree.



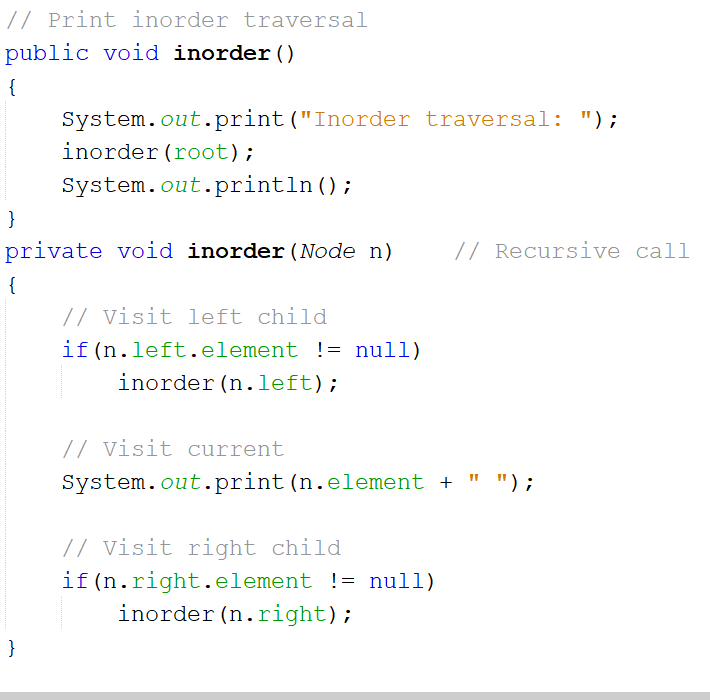
|  |  |
| --- | --- |
| Class | Node |
| Fields | element, left, right |
| Behaviors | n/a |

|  |  |
| --- | --- |
| Class | BinaryTree |
| Fields | root |
| Behaviors | insert(a), insert(a, currNode, n), inorder(), inorder(n), preorder(), preorder(n) |

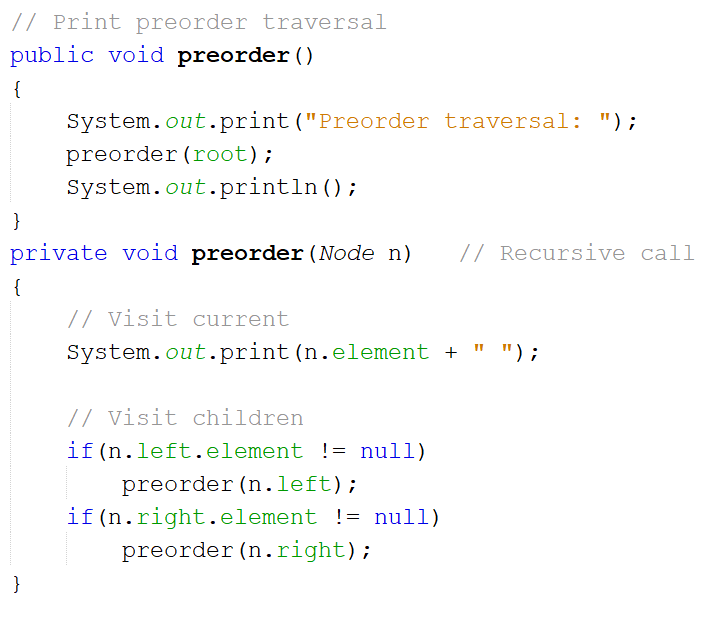


The *BinaryTree* is constructed using several *Nodes*. A root *Node* is established, with several left and right child *Nodes* branching off. The *insert*() method uses the rank formula to make recursive calls, although in this instance the formula assumes all rows of the tree up to the final row will be completely filled with nodes. As such, the input array must be constructed with the following conditions:

* Elements should be ordered based on position from top to bottom, left to right
* If a row has any external nodes above it, null values must represent that row’s empty positions



*inorder()* makes a recursive call for the left child, prints the current *Node*, and then makes a recursive call for the right child.



*preorder()* prints the current *Node* before making recursive calls for the left and then right children.