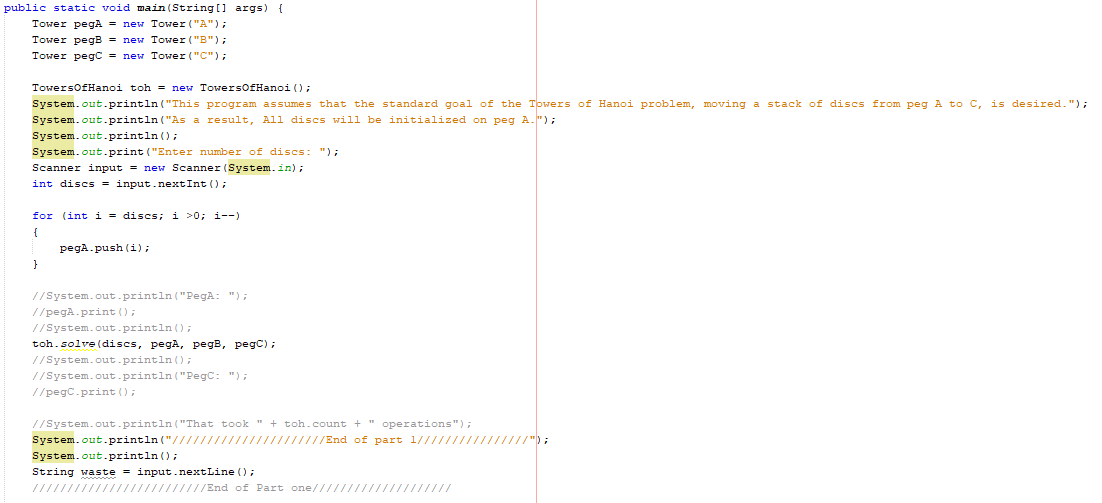
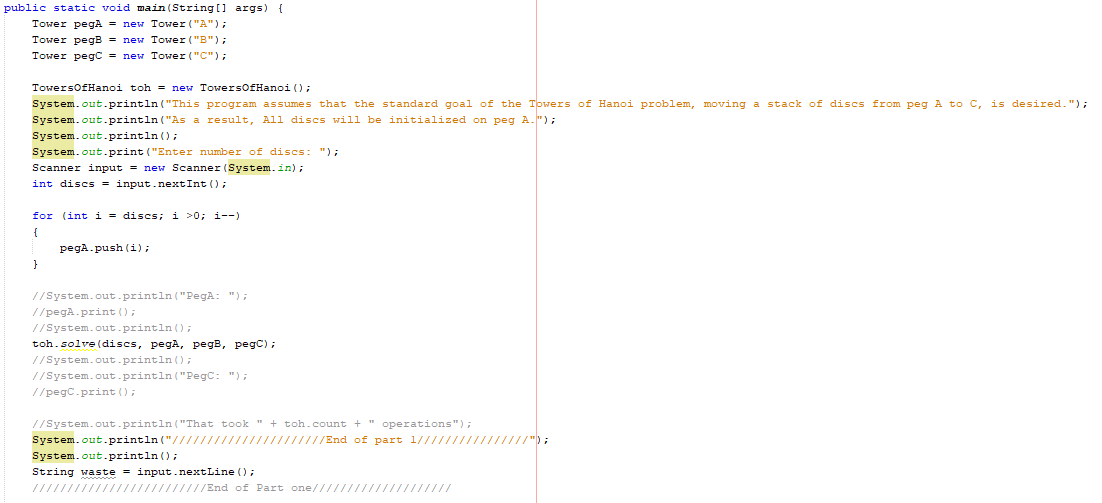
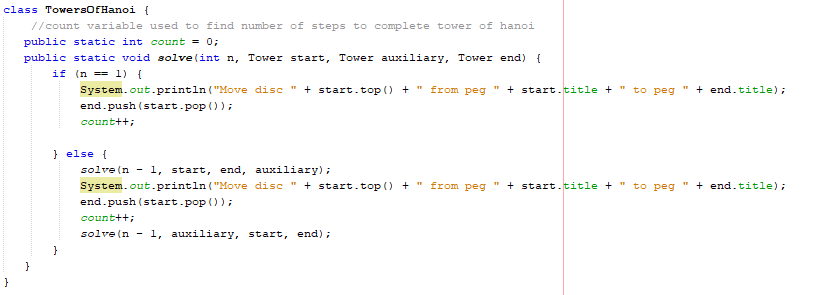
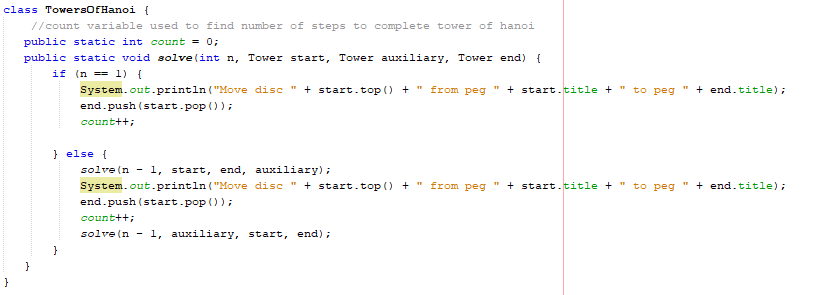
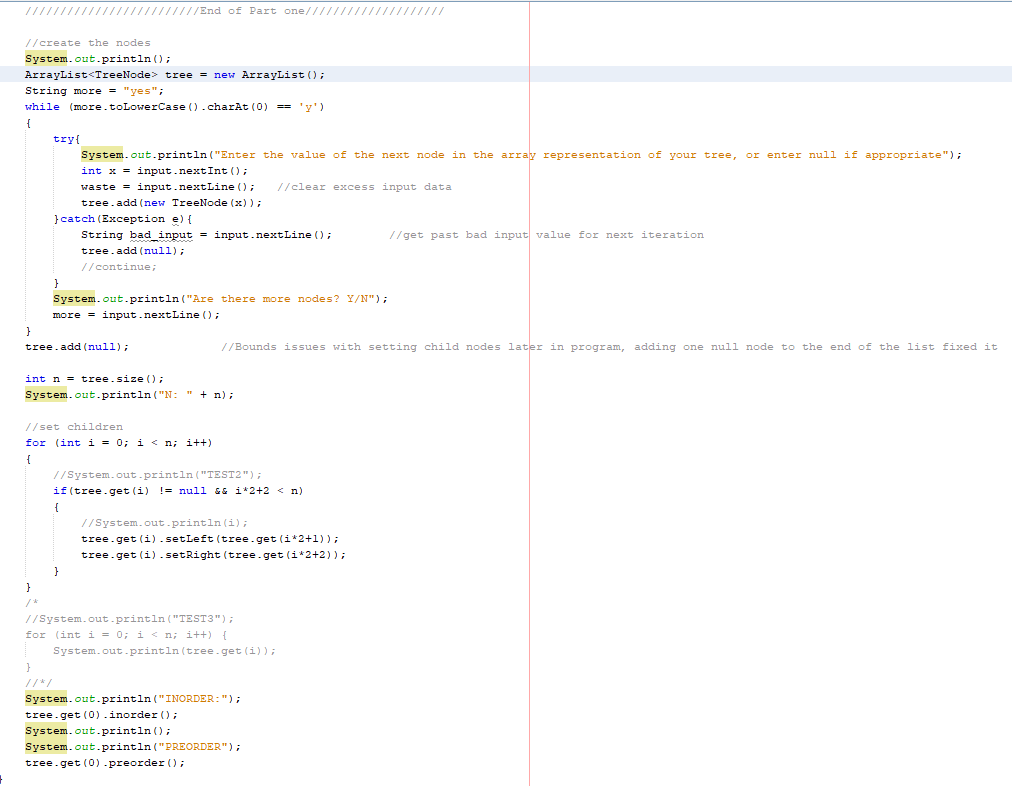
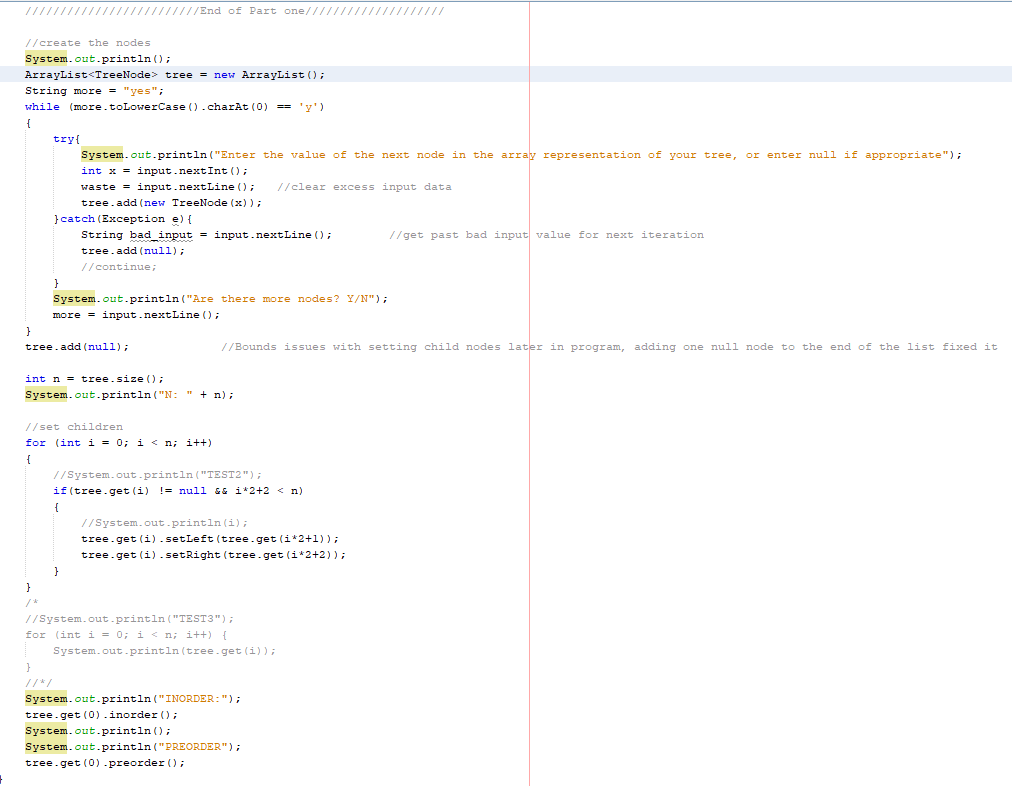
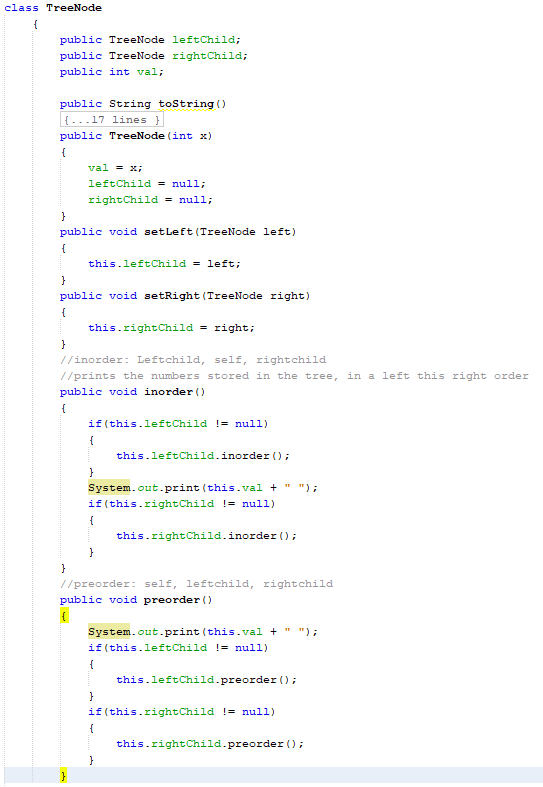
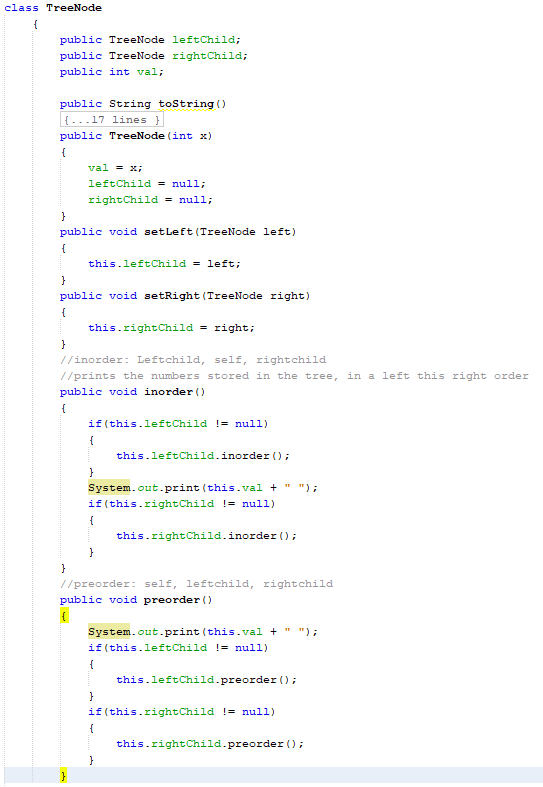
***well-documented summary that includes the annotation/justification of your algorithm and programming solution and the experimental results with the screenshots of your program testing you test. It should be at least 2 pages MS WORD or PDF file, (the more formal the better.) Note that screenshots should be readable to gain the full credits. The instructor will use the screenshots for student’s program demo.***

This document will rationalize the algorithms and functions used in my HW5 for CSC 310

1. For the first part of the Assignment, I used two custom classes. One was a customized stack class, which was a standard stack, but with an added String variable, which was the stacks name. This was needed for the output of the program, because if names were assigned in the solve method of the TowersOfHanoi class then they would be changing throughout the recursive calls. TowersOfHanoi had an Integer variable count, used to track the number of moves it would take to solve the puzzle, and had only one method. This method was the recursive method to solve the puzzle, which kept finding the solution for moving all discs except the bottom one from the start to the other peg which was also not the target peg, moved the bottom disc to the target peg, then moved all the other pegs to the target peg, on top of the bottom peg. It kept doing this with ever smaller numbers of discs, until only the last disc remained, which it then moved to the given target disc for that iteration through.
2. The second part of the assignment only utilized one Class, the TreeNode class, which consisted of an integer value with an associated TreeNode called leftChild, and another associated TreeNode called rightChild. The main method of the program prompted the user to build a list of these TreeNodes, until they had entered all their necessary nodes. One fluke in the program, which I suspect was an issue in my bounds testing during my loop, was that after all this input, the last value to be entered was not being assigned as a child. I fixed this by simply adding one final null value to the list. Due to my process behind setting children, this node would do nothing for the tree, aside from lengthening the ArrayList representation of it. I then called the inorder method of the root node and the preorder method of the root node. The inorder method recursively called the inorder method of the node’s leftChild, if it existed, before displaying the node’s own value and then calling the inorder method of the node’s rightChild, if it existed. Pre order did exactly the same thing, except the nodes own value was displayed first, then the preorder of the leftChild and then the rightChild were called.