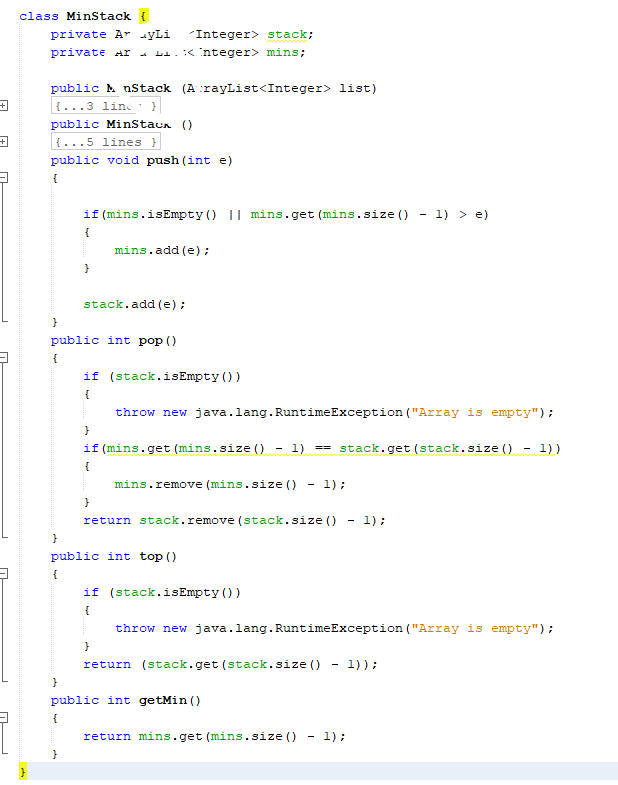
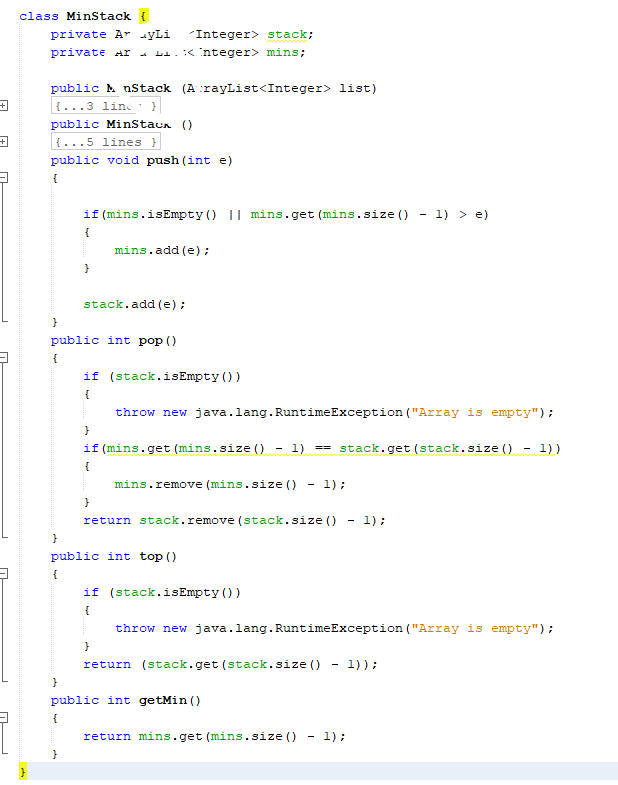
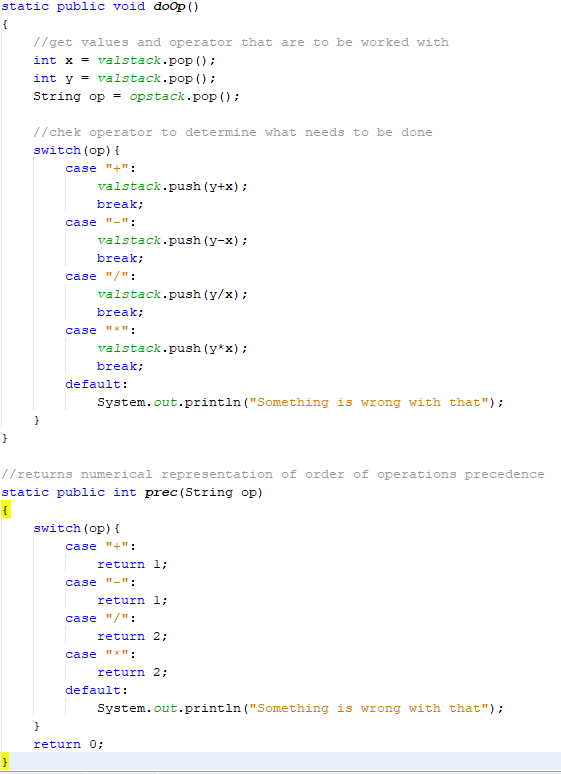
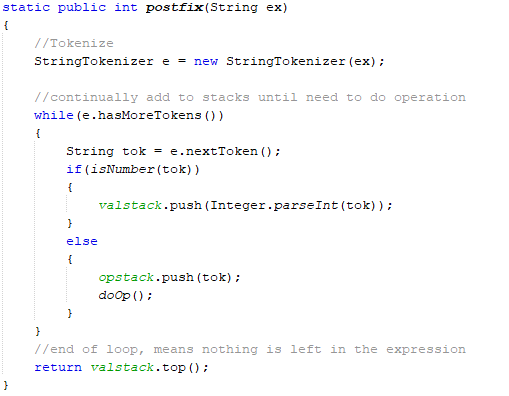
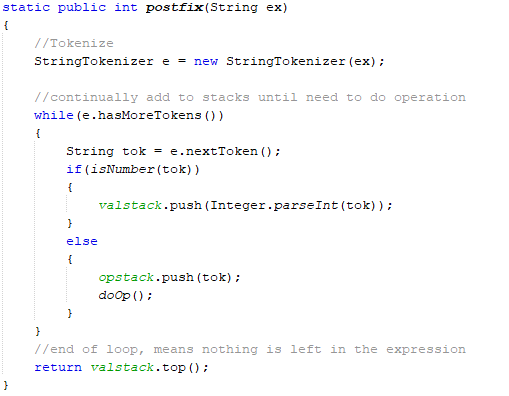
***well-documented summary that includes the annotation/justification of your algorithm and programming solution and the experimental results with the screenshots of your program testing you test. It should be at least 2 pages MS WORD or PDF file, (the more formal the better.) Note that screenshots should 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 HW1 for CSC 310

1. MinStack has two constructors allowing a premade arraylist<integer> to be given to populate the minStack, or for an empty minStack to be made   
   It also has two arraylists to help in maintaining constant time. One for every new value that is added to the minStack, and one to hold any new minimum values that are added.  
   Push simply adds the indicated value to the end of the arraylist  
   Pop returns the last value to be added, but also removes it from the main arraylist, and from the minimums list if it matches the top value of that list. Returns an error if the stack is not yet populated  
   Top returns the last value to be added to the stack  
   GetMin returns the top value of the minimums list, which is the smallest value contained within the stack
2. EvalExpression utilizes two custom stack classes of different data types, one for values and one for the string representations of operators. It also uses a prec method to determine order of operations precedence, a repOp method to use that precedence, and a doOp method to actually do operations  
   Takes in a String representation of an arithmetic expression, in infix notation, and returns the answer. Expression must have spaces between individual values and operators (ie. 1 + 2 / 3 rather than 1+2/3)
3. Postfix takes an arithmetic expression in postfix notation and returns the answer.  
   Does exactly the same thing as evalExpression, except that in postfix operators are already ordered by order of operations, so all that needs to be done is put tokens in their respective stacks and then use doOp until all values and operators have been used.

