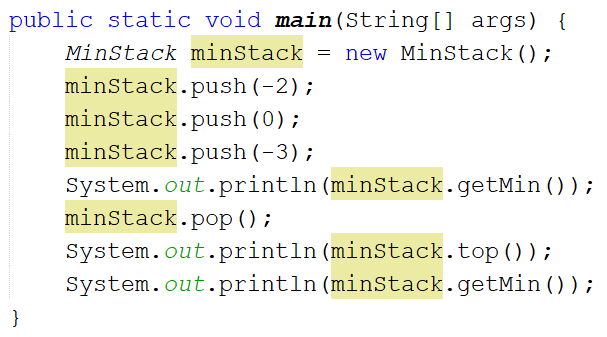
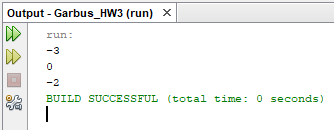
**Program 1**

This program utilizes a class – *MinStack* – that allows the user to store integers and keep track of the minimum values within their data.

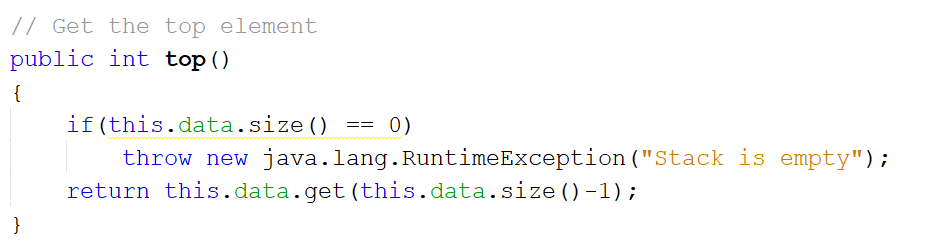




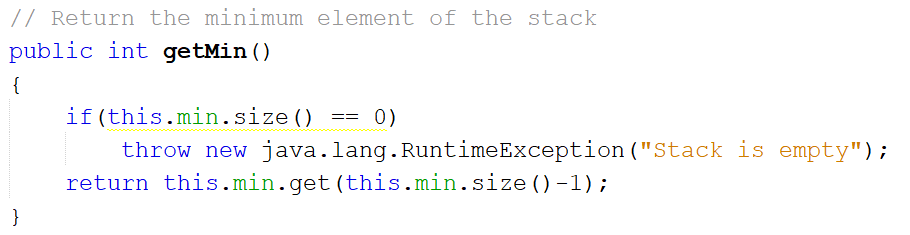
The *MinStack* class works by establishing 2 lists of integers: one for the main data and one for keeping track of minimum values. When a pushed value is less than the current minimum, it is pushed to the *min* stack. When this minimum value is popped from *data*, it is also popped from *min*. The user may call4 methods to handle data: *push()*, *pop(), top(),* and *getMin().*



*push()* and *pop()* function as expected of a stack: *push()* adds an element to the data and *pop()* removes and returns the top element of the stack.

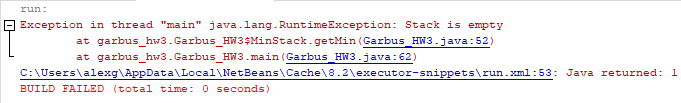


*top()* returns the top element of the stack without removing it.

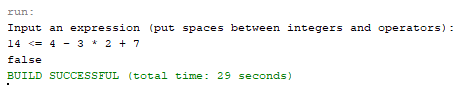
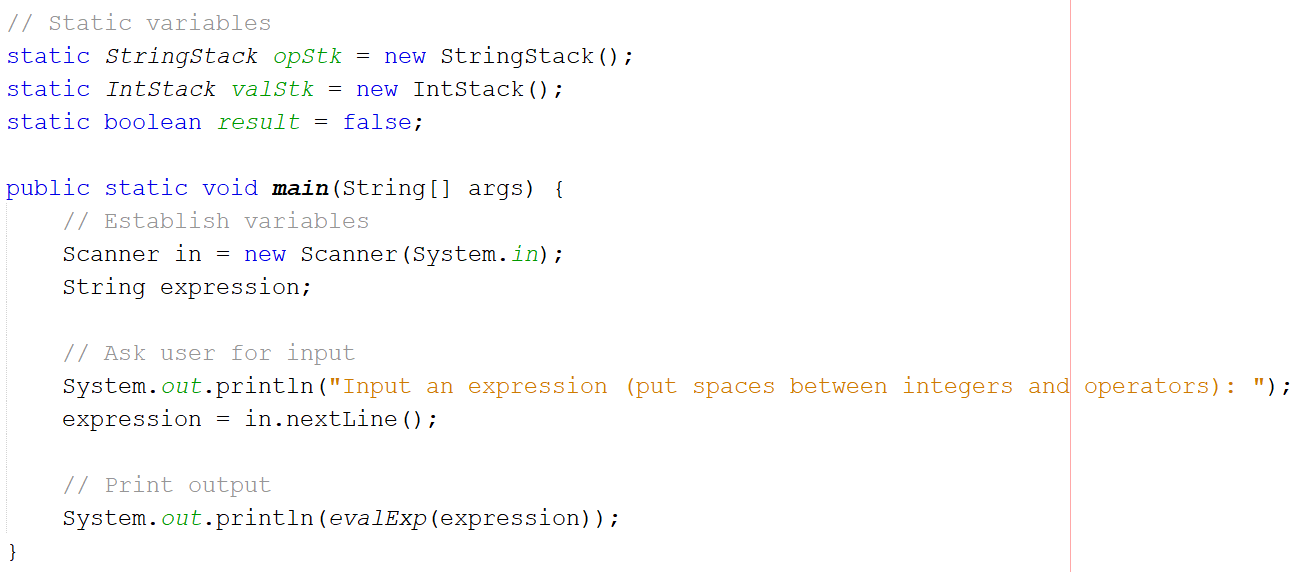


*getMin()* returns the current minimum value of the data.

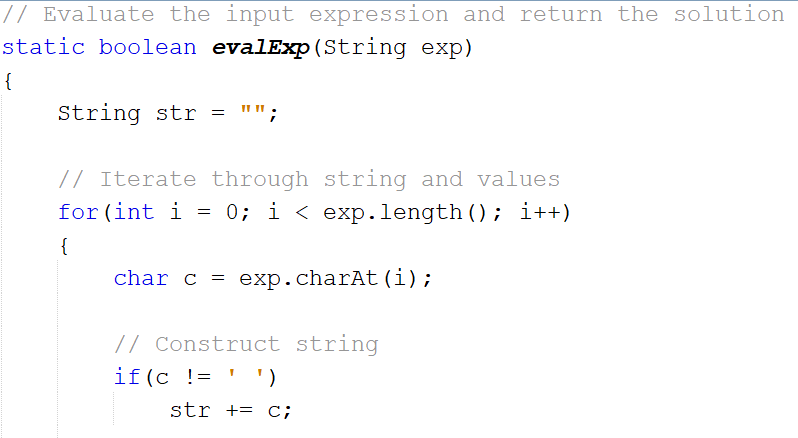
If at any point the user tries to access data from an empty stack, an exception is thrown.

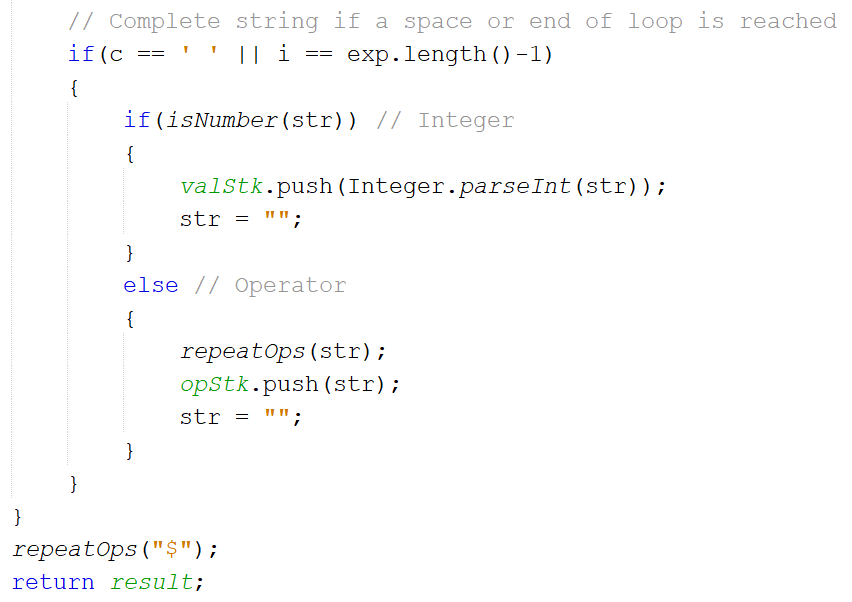


**Program 2**

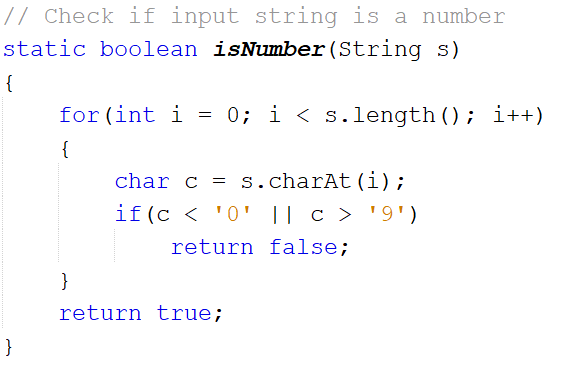


The program uses 2 stack classes – *StringStack* and *IntStack* – to read characters from an input expression and output whether the expression is true or false. Each of these classes has a *push(), pop(), top(),* and *size()* method; the only difference between the two is the type of data they hold: strings or integers.

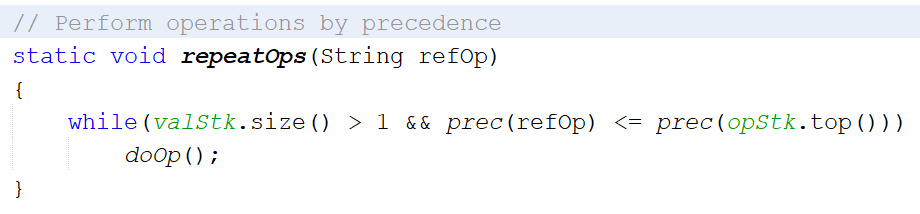




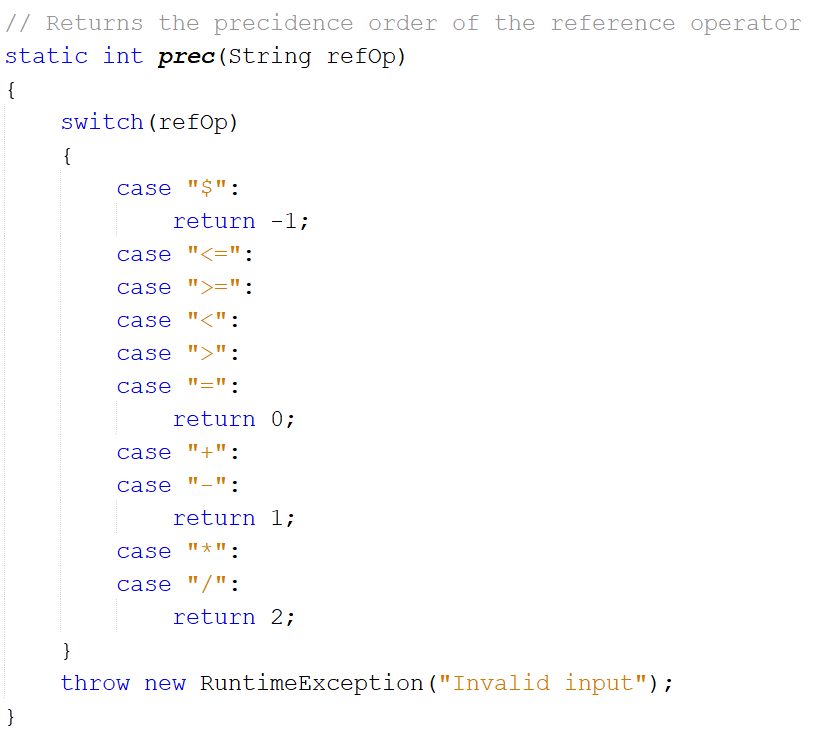
*evalExp()* is the method that handles reading the input and returning the output (a Boolean value). It loops through the expression and constructs strings for each integer and operator, using spaces as indication of when to push the current string and move to the next. When an operator is encountered, the program attempts to perform operations. Once the loop is finished, a $ is used as an end-of-input operator, and the result is then returned.



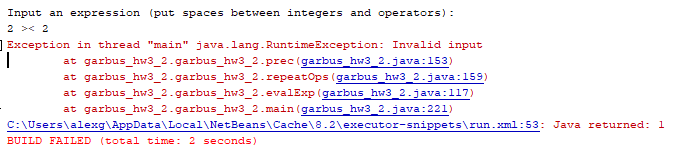
*isNumber()* simply checks whether the characters of a string can be converted to integers.



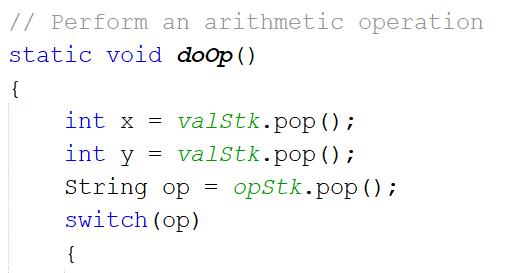
*repeatOps()* ensures that operations are performed in order of precedence, and that operations are only performed if the stack has more than 1 value.



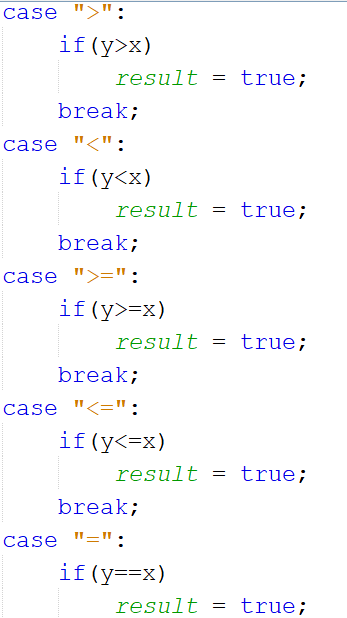
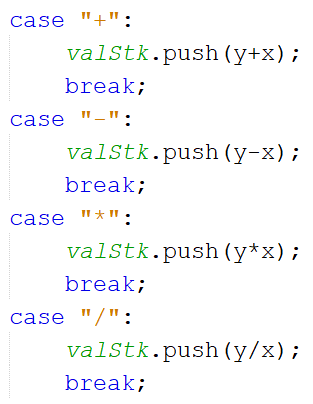
*prec()* returns the precedence order of an operator.



If an unidentifiable operator is detected, an exception is returned.

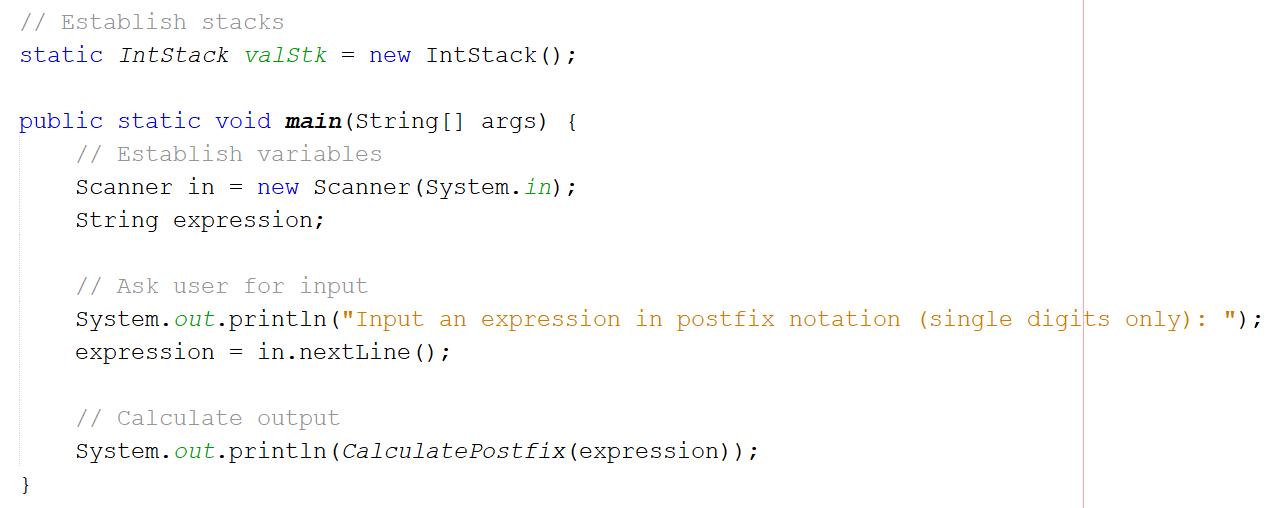


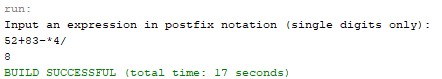
*doOp()* actually performs the operation between 2 integers.



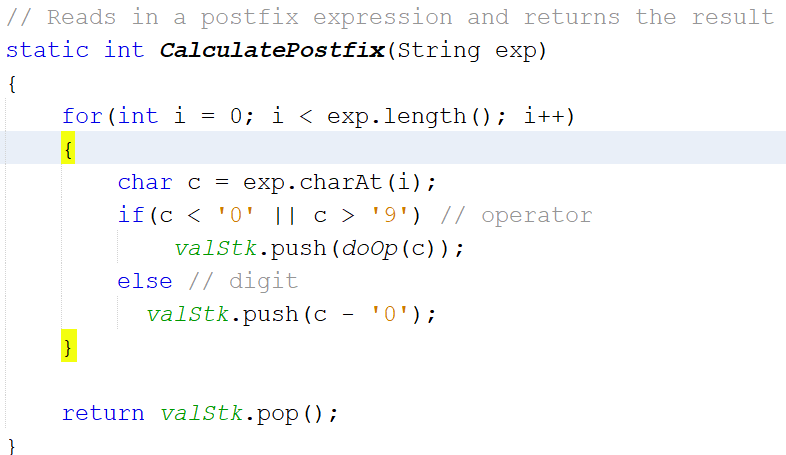
If an arithmetic operator is used, the result of the corresponding operation is pushed to the data. Otherwise, if an equality operator is used, the equality is evaluated and the result saved.

**Program 3**





The program uses an *IntStack* class for storing the numbers found within the postfix expression. *IntStack* has methods *push()* and *pop()* for handling data.



The *CalculatePostfix()* method handles the calculation of the postfix expression’s value. It loops through the expression, pushing any integers into the stack. Once it reaches an operator, it calls *doOp()*, pushing the result to the stack. Once the loop finishes, 1 value should remain in the stack; this value is returned as the result.



*doOp()* takes the previous 2 integers and performs the appropriate arithmetic operation on them. If the input character happens to not be an arithmetic operator, an exception is thrown.

