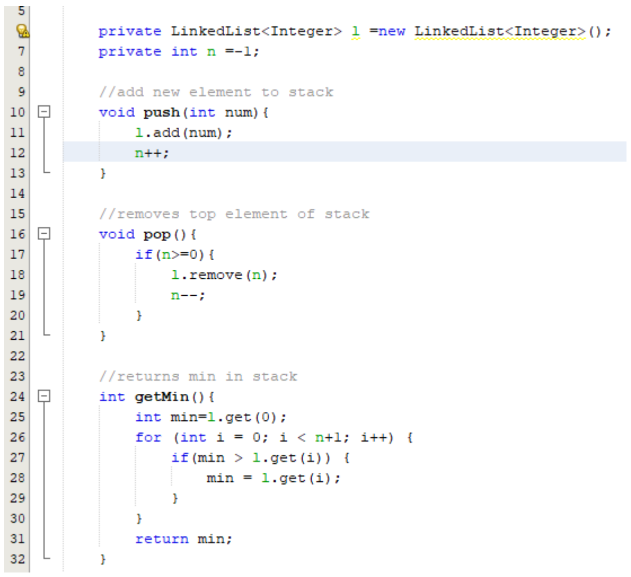
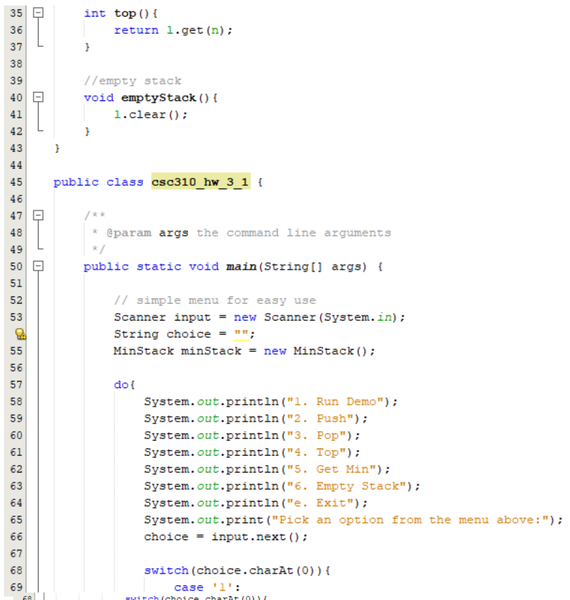
CSC310_HW_3_1



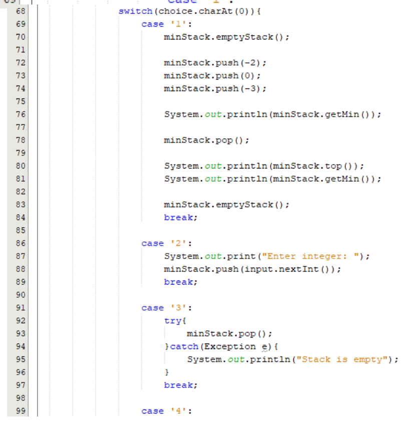
```java

    private LinkedList<Integer> l =new LinkedList<Integer>();
    private int n =-1;

    //add new element to stack
    void push(int num){
        l.add(num);
        n++;
    }

    //removes top element of stack
    void pop(){
        if(n>=0){
            l.remove(n);
            n--;
        }
    }

    //returns min in stack
    int getMin(){
        int min=l.get(0);
        for (int i = 0; i < n+1; i++) {
            if(min > l.get(i)) {
                min = l.get(i);
            }
        }
        return min;
    }
```

In this program I made a class called minstack and its meant to manage a int value with the option to find the min value in the stack. I decided to make it an int because it wasn’t specific and integers are easy to work with and small data type. I decided to make the min value return a int because that is what it ask for. I also included the basic stack functions push and pop to add and remove from the stack.



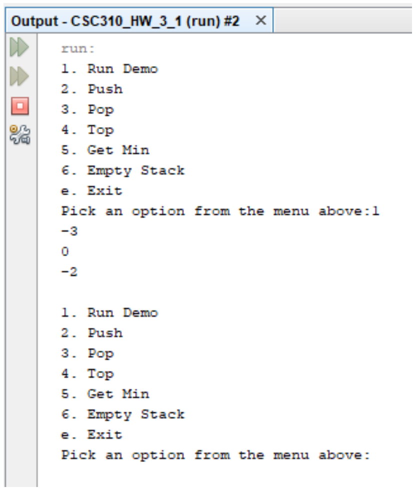
```java
    int top(){
        return l.get(n);
    }

    //empty stack
    void emptyStack(){
        l.clear();
    }
}

public class csc310_hw_3_1 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        // simple menu for easy use
        Scanner input = new Scanner(System.in);
        String choice = "";
        MinStack minStack = new MinStack();

        do{
            System.out.println("1. Run Demo");
            System.out.println("2. Push");
            System.out.println("3. Pop");
            System.out.println("4. Top");
            System.out.println("5. Get Min");
            System.out.println("6. Empty Stack");
            System.out.println("e. Exit");
            System.out.print("Pick an option from the menu above:");
            choice = input.next();

            switch(choice.charAt(0)){
                case '1':
```

I have another basic stack function top and added a emptyStack function to work better with the menu that you can see it’s a basic and simple menu. I chose to ad a menu to easily use the function and to quickly to run the Demo with example in it.



```java
            switch(choice.charAt(0)){
                case '1':
                    minStack.emptyStack();

                    minStack.push(-2);
                    minStack.push(0);
                    minStack.push(-3);

                    System.out.println(minStack.getMin());

                    minStack.pop();

                    System.out.println(minStack.top());
                    System.out.println(minStack.getMin());

                    minStack.emptyStack();
                    break;

                case '2':
                    System.out.print("Enter integer: ");
                    minStack.push(input.nextInt());
                    break;

                case '3':
                    try{
                        minStack.pop();
                    }catch(Exception e){
                        System.out.println("Stack is empty");
                    }
                    break;

                case '4':
```

This an continuation of the menu with the first 3 cases

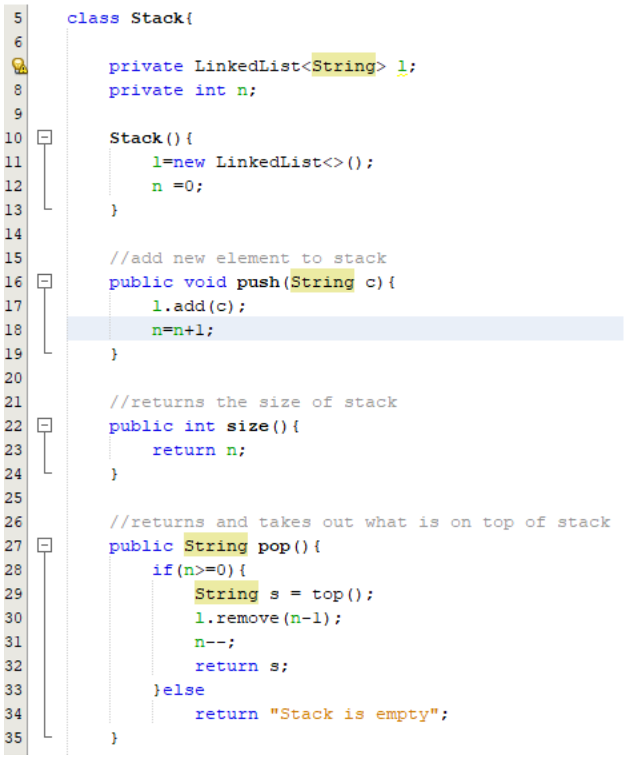


Output - CSC310_HW_3_1 (run) #2

```
run:
1. Run Demo
2. Push
3. Pop
4. Top
5. Get Min
6. Empty Stack
e. Exit
Pick an option from the menu above:1
-3
0
-2

1. Run Demo
2. Push
3. Pop
4. Top
5. Get Min
6. Empty Stack
e. Exit
Pick an option from the menu above:
```

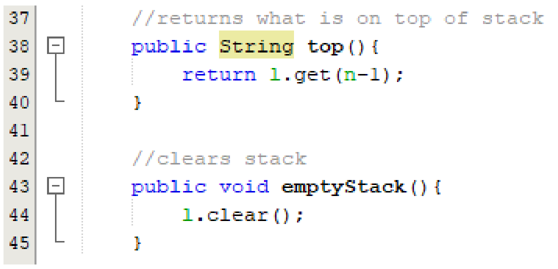
This is an example of the inputs and out puts for the program and how the menu works.

CSC310_HW_3_2

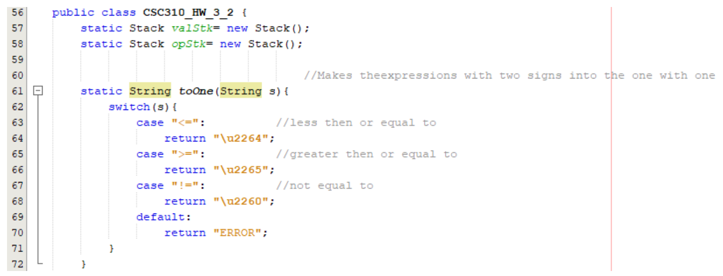


```java
class Stack{

    private LinkedList<String> l;
    private int n;

    Stack(){
        l=new LinkedList<>();
        n =0;
    }

    //add new element to stack
    public void push(String c){
        l.add(c);
        n=n+1;
    }

    //returns the size of stack
    public int size(){
        return n;
    }

    //returns and takes out what is on top of stack
    public String pop(){
        if(n>=0){
            String s = top();
            l.remove(n-1);
            n--;
            return s;
        }else
            return "Stack is empty";
    }
```

I made String Stack for easier manipulation of the data and so that I can use it for both the value and operations stack. It has similar functions to the first program with the basics with an additional size function.

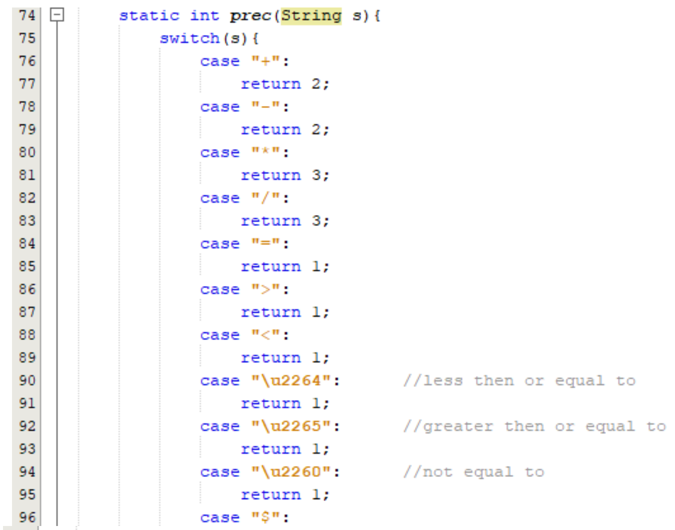


```java
    //returns what is on top of stack
    public String top(){
        return l.get(n-1);
    }

    //clears stack
    public void emptyStack(){
        l.clear();
    }
```



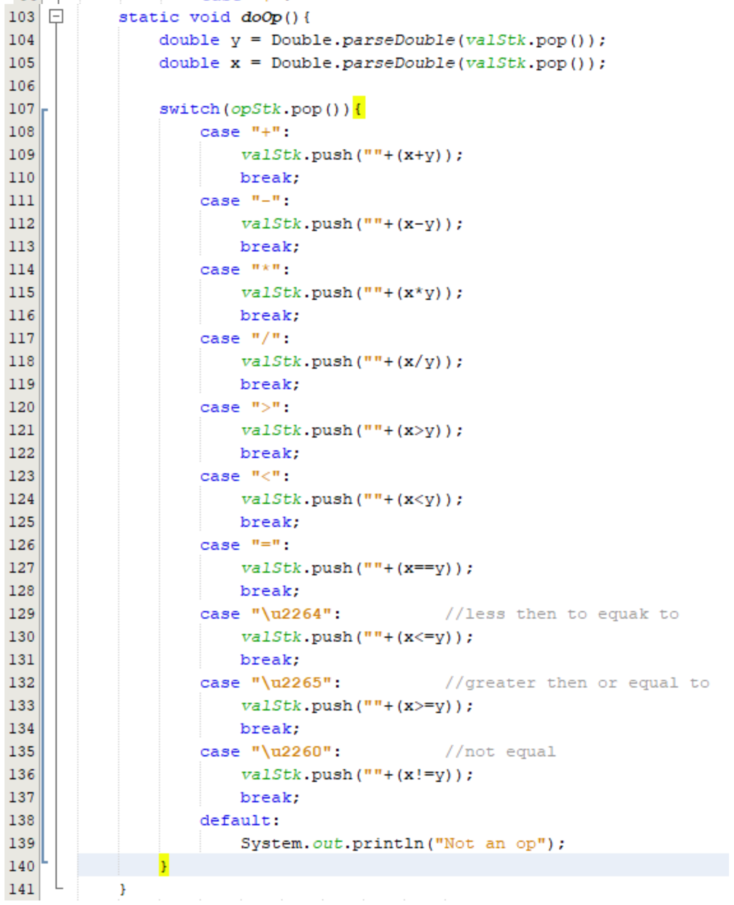
```java
public class CSC310_HW_3_2 {
    static Stack valStk= new Stack();
    static Stack opStk= new Stack();

                                        //Makes theexpressions with two signs into the one with one
    static String toOne(String s){
        switch(s){
            case "<=":          //less then or equal to
                return "\u2264";
            case ">=":          //greater then or equal to
                return "\u2265";
            case "!=":          //not equal to
                return "\u2260";
            default:
                return "ERROR";
        }
    }
```

I decided to make a function to handle if the user inputs characters that doesn't work well in the stack such as <= or != and change it to a more stack friendly format.



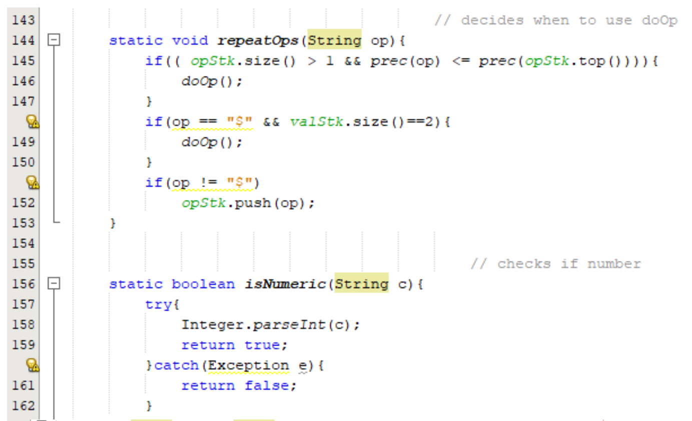
```java
    static int prec(String s){
        switch(s){
            case "+":
                return 2;
            case "-":
                return 2;
            case "*":
                return 3;
            case "/":
                return 3;
            case "=":
                return 1;
            case ">":
                return 1;
            case "<":
                return 1;
            case "\u2264":      //less then or equal to
                return 1;
            case "\u2265":      //greater then or equal to
                return 1;
            case "\u2260":      //not equal to
                return 1;
            case "$":
```

I made this function to keep track of the precedence of each operation and to help compare them better in if and loops.



```java
    static void doOp(){
        double y = Double.parseDouble(valStk.pop());
        double x = Double.parseDouble(valStk.pop());

        switch(opStk.pop()){
            case "+":
                valStk.push(""+(x+y));
                break;
            case "-":
                valStk.push(""+(x-y));
                break;
            case "*":
                valStk.push(""+(x*y));
                break;
            case "/":
                valStk.push(""+(x/y));
                break;
            case ">":
                valStk.push(""+(x>y));
                break;
            case "<":
                valStk.push(""+(x<y));
                break;
            case "=":
                valStk.push(""+(x==y));
                break;
            case "\u2264":          //less then to equak to
                valStk.push(""+(x<=y));
                break;
            case "\u2265":          //greater then or equal to
                valStk.push(""+(x>=y));
                break;
            case "\u2260":          //not equal
                valStk.push(""+(x!=y));
                break;
            default:
                System.out.println("Not an op");
        }
    }
```

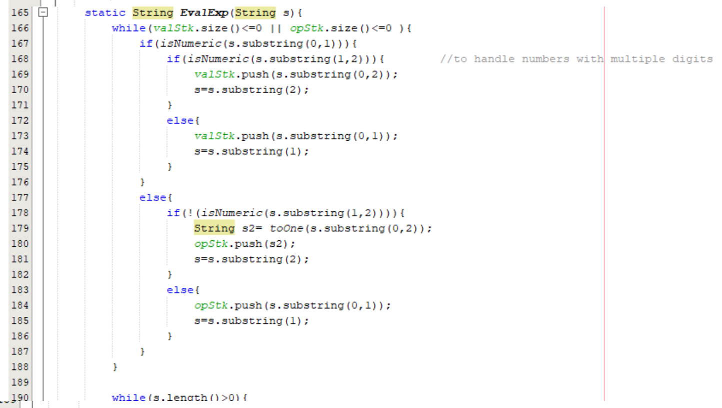
I made this function to do the actual operations associated with the character. I felt like a switch statement was the best and most efficient way of handling this.



```
                                        // decides when to use doOp
    static void repeatOps(String op){
        if(( opStk.size() > 1 && prec(op) <= prec(opStk.top()))){
            doOp();
        }
        if(op == "$" && valStk.size()==2){
            doOp();
        }
        if(op != "$")
            opStk.push(op);
    }


                                            // checks if number
    static boolean isNumeric(String c){
        try{
            Integer.parseInt(c);
            return true;
        }catch(Exception e){
            return false;
        }
```

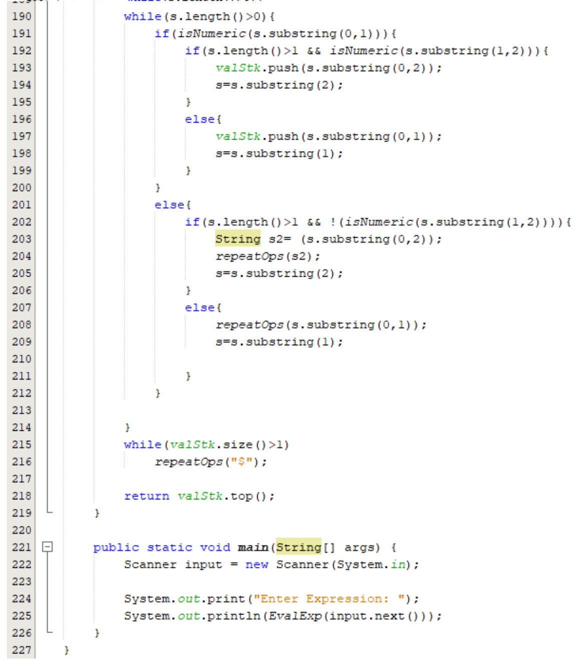
This function was made to handle when to do the operations

This function I created to check and to if the string is a number or not and returns a Boolean to easily sort between number and operation



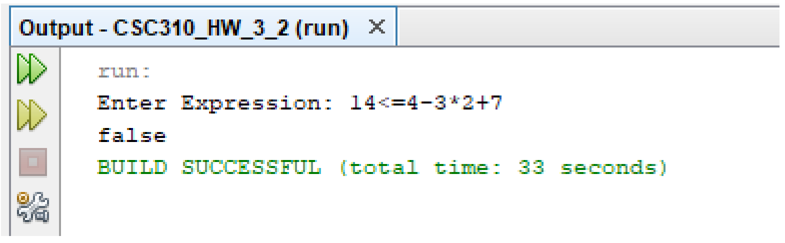
```
    static String EvalExp(String s){
        while(valStk.size()<=0 || opStk.size()<=0 ){
            if(isNumeric(s.substring(0,1))){
                if(isNumeric(s.substring(1,2))){        //to handle numbers with multiple digits
                    valStk.push(s.substring(0,2));
                    s=s.substring(2);
                }
                else{
                    valStk.push(s.substring(0,1));
                    s=s.substring(1);
                }
            }
            else{
                if(!(isNumeric(s.substring(1,2)))){
                    String s2= toOne(s.substring(0,2));
                    opStk.push(s2);
                    s=s.substring(2);
                }
                else{
                    opStk.push(s.substring(0,1));
                    s=s.substring(1);
                }
            }
        }

        while(s.length()>0){
```

This function handles the full expression and calls other functions that does the operations it also I decided to make sure that it can tell when it's a multidigit number or a two part operation such as!=.



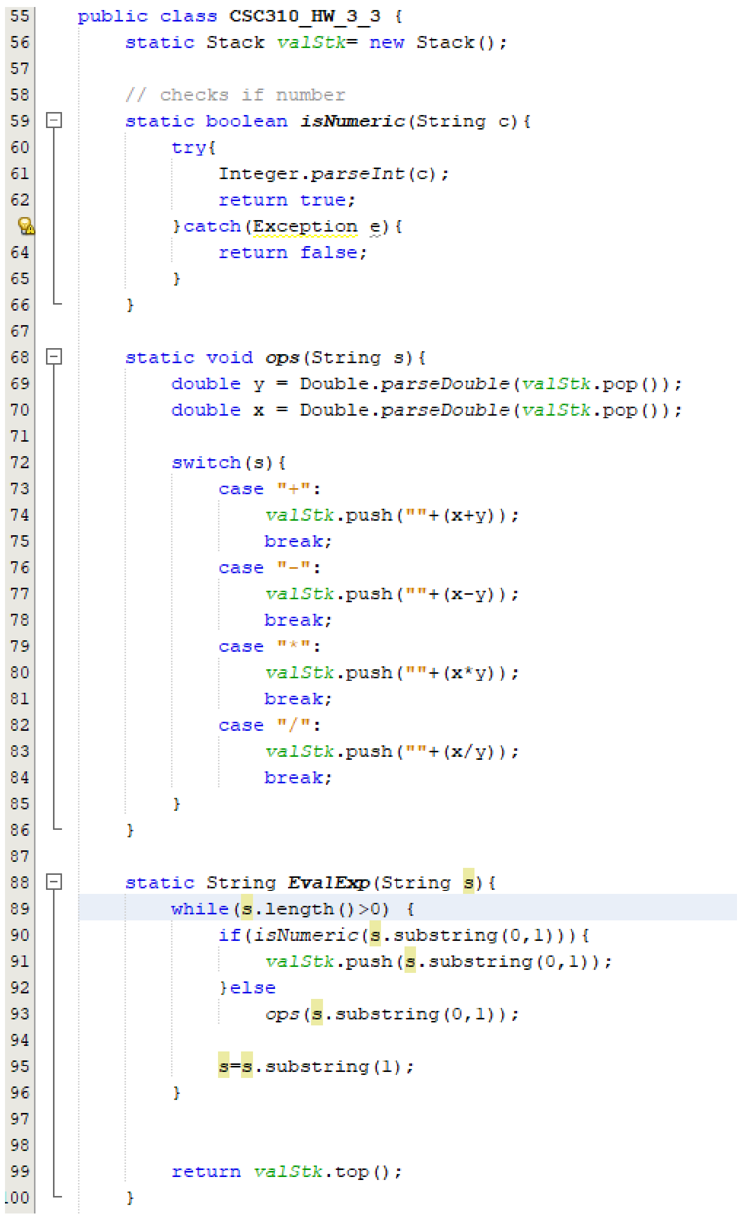
```
        while(s.length()>0){
            if(isNumeric(s.substring(0,1))){
                if(s.length()>1 && isNumeric(s.substring(1,2))){
                    valStk.push(s.substring(0,2));
                    s=s.substring(2);
                }
                else{
                    valStk.push(s.substring(0,1));
                    s=s.substring(1);
                }
            }
            else{
                if(s.length()>1 && !(isNumeric(s.substring(1,2)))){
                    String s2= (s.substring(0,2));
                    repeatOps(s2);
                    s=s.substring(2);
                }
                else{
                    repeatOps(s.substring(0,1));
                    s=s.substring(1);

                }
            }

        }
        while(valStk.size()>1)
            repeatOps("$");

        return valStk.top();
    }

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter Expression: ");
        System.out.println(EvalExp(input.next()));
    }
}
```

The screenshot below is an example input and output of the program.



```
run:
Enter Expression: 14<=4-3*2+7
false
BUILD SUCCESSFUL (total time: 33 seconds)
```

CSC310_HW_3_3

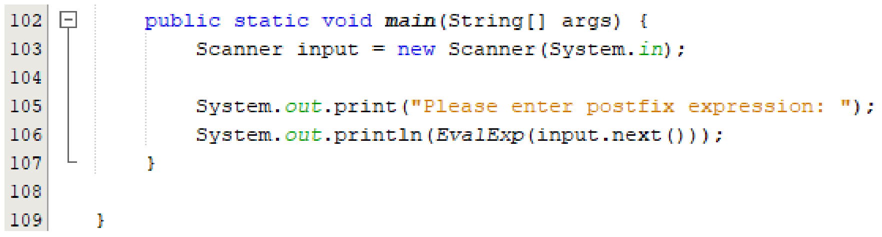


```
public class CSC310_HW_3_3 {
    static Stack valStk= new Stack();

    // checks if number
    static boolean isNumeric(String c){
        try{
            Integer.parseInt(c);
            return true;
        }catch(Exception e){
            return false;
        }
    }

    static void ops(String s){
        double y = Double.parseDouble(valStk.pop());
        double x = Double.parseDouble(valStk.pop());

        switch(s){
            case "+":
                valStk.push(""+(x+y));
                break;
            case "-":
                valStk.push(""+(x-y));
                break;
            case "*":
                valStk.push(""+(x*y));
                break;
            case "/":
                valStk.push(""+(x/y));
                break;
        }
    }

    static String EvalExp(String s){
        while(s.length()>0) {
            if(isNumeric(s.substring(0,1))){
                valStk.push(s.substring(0,1));
            }else
                ops(s.substring(0,1));

            s=s.substring(1);
        }


        return valStk.top();
    }
```

For this program I was able to reuse a lot of the code from the last program it work very well such as the string stack class and the is numeric.

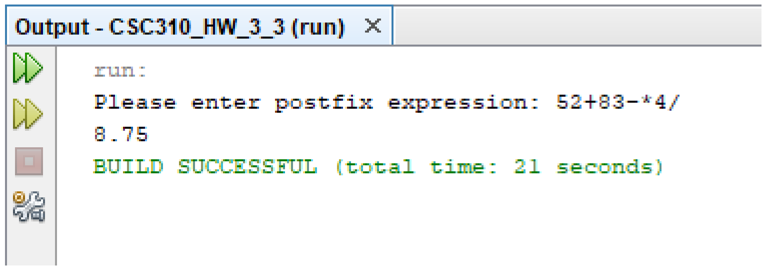
I was able to bacially sue all the code from the from the doOp function and renamed ops for operations which this function handles all the operations section

This function serves the same purpose as from the last program but I was able to streamline it with one stack and with a middle function I was able to send it straight to the ops function.



```
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Please enter postfix expression: ");
        System.out.println(EvalExp(input.next()));
    }

}
```

This is the cod that prompts the user for input.



Output - CSC310_HW_3_3 (run)

```
run:
Please enter postfix expression: 52+83-*4/
8.75
BUILD SUCCESSFUL (total time: 21 seconds)
```

This is example of the postfix expression input and the out from it.