Assignment 2

Problem Statement

We need to make a program that is able to take input from the user, convert it to postfix and once converted, evaluate the postfix from a stack and give the answer back to the user. We will need to push and pop from and to the stack in the right order back on precedence of the operator and operator in the stack and whether it is a bracket or number we are pushing or popping.

Analysis and Design Notes

I need to make a method that is able to get input from the user so that I can parse every character to check that is either a number, bracket or operator. I'll need a method that is able to return a value of precedence based on an operator passed in and the operator on the top of stack. I'll need a method that is able to convert infix to postfix by scanning character by character, pushing all numbers to the stack. If it's a '(' push to the stack. Push operator to stack if it has higher precedence than, pop from stack until you reach something that has the same or higher precedence or a open bracket.

- Create ArrayStack
- Get input from user and check that input is valid and return to character array
- Call infix to postfix function with character array
- Loop character array, pushing to stack & output string in correct order based on whether they are numbers, brackets or operators and precedence.
- Call postfix evaluator function which will take two numbers from the stack and an operator, evaluate them and return the result to stack.

PseudoCode

```
ArrayStack s = new ArrayStack();
char [] in = getInput();

For (char x : in) {
        If ('0' >= x <= '9') out.append(x);
        Else If (s.isEmpty || pLevel == 2 || s.top == '(') s.push(x);
        Else while (!isEmpty && pLevel >= 1 && s.top != '(') {
                Out += s.top();
                s.pop();
        }
        s.push(x);
}

If (!isEmpty()) {

If (x== ‘(‘) s.push(x);
        If (s.top == ‘)’) {
                s.pop();
                While (s.top != ‘(‘) {
                        Out += s.top;
                        s.pop();
                }
        }
}
// end for loop

// Take everything off stack and put into string
While (!isEmpty()) {
        If (s.top() != ‘(‘) {
                Out += s.top;
        }
        s.pop()
}

For (char x : out) {
        If (‘0’ >= x <= 9) s.push(x);
        Else if (isOperator(x)) {
                Z = s.top;
                S.pop;
                Y = s.top;
                S.pop;

                if (x == '*') s.push(z*y);
                else if (x == '/') s.push(y/z);
                else if (x == '+') s.push(z+y);
                else if (x == '-') s.push(y-z);
                else s.push(Math.pow(y, z));
        }
}

Out = s.pop();
```

Flow

Code

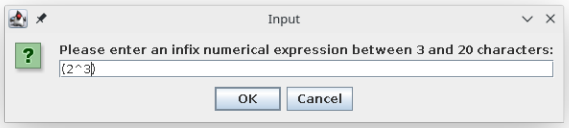
```
import javax.swing.*;

public class MyApp {
    // Init Array Stack
    private static ArrayStack s = new ArrayStack();

    public static void main(String [] args) {
    char[] in = getInput();
    String postfix = infixToPostfix(in);
    JOptionPane.showMessageDialog(null, "Postfix: " + postfix);
    double ans = postfixEvaluator(postfix);
    JOptionPane.showMessageDialog(null, "Answer: " + ans);
    }

    public static String infixToPostfix(char[] in) {
    StringBuilder out = new StringBuilder();

    // Loop until reach the end of string.
    for (char x : in) {
    // If character is number, append to string
    if (x >= '0' && x <= '9') {
        out.append(x);
        // If stack is empty, precedence of character is higher than character in stack, or
the stack has an open bracket, push x
    } else if (s.isEmpty() || precedenceLevel(x) == 2 || (char)s.top() == '(') {
        s.push(x);
        // break out of loop for iteration so character is not pushed twice
        continue;
    } else {
        // While x has a precedence level higher or equal to character in stack, stack is
not empty, and top of stack is not an open bracket, append x to string, pop from stack
        while (!s.isEmpty() && precedenceLevel(x) >= 1 && (char)s.top() != '(') {
        out.append((char) s.top());
        s.pop();
        }
        // Push character x of high precedence
```

```
                s.push(x);
                // Break so x is not pushed twice
                if ((char)s.top() == '(') continue;
        }

        if (!s.isEmpty()) {
                // If stack top is open bracket, push x
                if (x == '(') s.push(x);
                /* if stack top is close bracket, pop ')' from stack
                *  While stack top is not '(', pop everything from stack and append to string*/
                if ((char) s.top() == ')') {
                s.pop();
                while ((char) s.top() != '(') {
                out.append((char) s.top());
                s.pop();
                }
                // Get rid of '(' from stack
                s.pop();
                }
        }
        }

        // Once the whole string has been scanned, pops all operators on the stack to output
string
        while (!s.isEmpty()) {
        if ((char)s.top() != '(') {
                out.append((char) s.top());
        }
        s.pop();
        }
        return out.toString();
        }

        /* Scan input character by character
        *  If character x is a number, push to stack, if its an operator, pop two numbers from the
stack and evaluate with the operator.
        *  Push the answer to the stack, repeat until reach the end of input */
        public static double postfixEvaluator(String in) {
        double out,z,y;
        for (int i = 0; i < in.length(); i++) {
        char x = in.charAt(i);
        if (x >= '0' && x <= '9') {
                s.push((double)Character.getNumericValue(x));
        } else if (isOperator(x)) {
                z = (double)s.top();
                s.pop();
                y = (double)s.top();
                s.pop();

                if (x == '*') s.push(z*y);
                else if (x == '/') s.push(y/z);
                else if (x == '+') s.push(z+y);
                else if (x == '-') s.push(y-z);
                else s.push(Math.pow(y, z));
        }
        }

        out = (double)s.pop();
        return out;
        }

        // returns true if character is an operator
        public static boolean isOperator(char x) {
        return x == '+' || x == '-' || x == '*' || x == '/' || x == '^';
        }

        // As long as stack is not empty, return level of precedence of x comparable to stack top.
        public static int precedenceLevel(char x) {
        if (!s.isEmpty()) {
        if (x == '^') {
                if ((char) s.top() == '^') return 1;
                else return 2;
        } else if (x == '/' || x == '*') {
                if ((char) s.top() == '+' || (char) s.top() == '-') return 2;
                else if ((char) s.top() == '/' || (char) s.top() == '*') return 1;
                else return 0;
        } else {
                return 0;
        }
        }
        return 0;
        }

        // Constructor not needed as testing is done in main
        public MyApp() {
        	}

        	public static char[] getInput() {
        	// Taking input from the user;
        	String input = JOptionPane.showInputDialog(null,"Please enter an infix numerical
        expression between 3 and 20 characters:");
        	char[] in = new char[input.length()];
        	int openBracketCounter = 0, closeBracketCounter = 0;

        	// Looping through input checking that character in input is in-between 0-9, an *, /, ^, -, or
        + and that the character being scanned, has no operator before or after it.
        	for (int i = 0; i < input.length(); i++) {
        	// Outer if just checks that character in input is in-between 0-9, an *, /, ^, -, or +
        	char x = input.charAt(i), xMinus = 0, xPlus = 0;
        	if (i > 0) xMinus = input.charAt(i-1);
        	if (i <= input.length() - 2) xPlus = input.charAt(i+1);

        	// Outer if statement checking that input is within length boundary & that is only uses
        selected characters
        	if ((input.length() <= 20 && input.length() >= 3) && (x >= '0' && x <= '9') || x == '(' || x == ')'
        || x == '+' || x == '-' || x == '/' || x == '*' || x == '^') {
        		// If bracket is entered, we want to make sure it is accounted for and cancelled
        out with another bracket.
        		if (x == '(') openBracketCounter++;
        		else if (x == ')') closeBracketCounter++;

        		// If i = 0, we don't want to be referencing x - 1, When i = 0, the only characters
        that can be there are an open bracket and number
        		if (i == 0 && ((x == '(' && !isOperator(xPlus)) || (!isOperator(x) &&
        isOperator(xPlus)))) {
        		in[i] = x;
        		}

        		// When at the last character, all we have to worry about is it not being an open
        bracket and it not being an operator
        		else if (i == input.length() - 1 && x != '(' && !isOperator(x)) {
        		in[i] = x;
        		}

        		//  For every number in-between, we check if its an operator, that the character
        behind and ahead is not an operator. we check if its not a operator that either there is an
        operator ahead or behind.
```

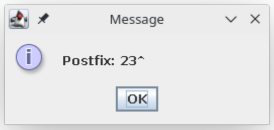
```
        else if ((isOperator(x) && !isOperator(xMinus) && !isOperator(xPlus)) ||
(!isOperator(x) && (isOperator(xPlus) || isOperator(xMinus)))) {
        in[i] = x;
        }

        // If input is invalid, print message and ask for input again.
        else {
        JOptionPane.showMessageDialog(null, "Only the following characters are valid:
+, -, *, /, ^, (, ) and numbers 0-9 in single use\n");
        return getInput();
        }
    } else {
        return getInput();
    }
    }
    // If input is valid, for loop will go through without errors and return character array.
    // If all brackets are opened and closed, openBracketCounter + closeBracketCounter %2
will return 0
    if ((openBracketCounter + closeBracketCounter) % 2 == 0) return in;
    else {
    JOptionPane.showMessageDialog(null, "Only the following characters are valid: +, -, *, /,
^, (, ) and numbers 0-9 in single use\n");
    return getInput();
    }
    }
}
```

Testing

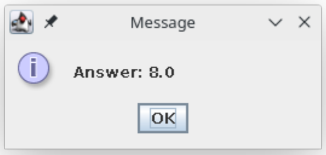
Test that normal input works for operators of different precedences:



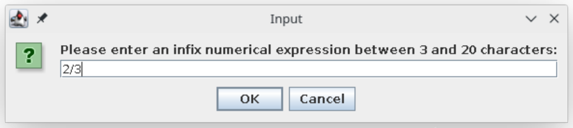
```
Please enter an infix numerical expression between 3 and 20 characters:
(2^3)
```



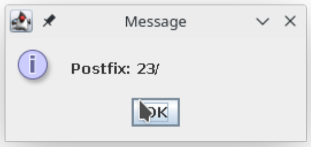
```
Postfix: 23^
```



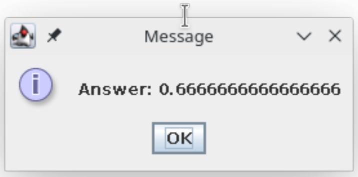
```
Answer: 8.0
```



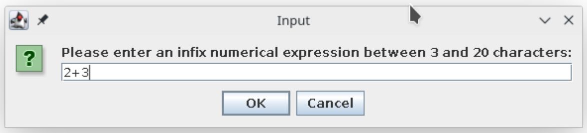
```
Please enter an infix numerical expression between 3 and 20 characters:
2/3
```



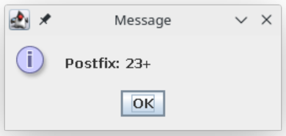
```
Postfix: 23/
```



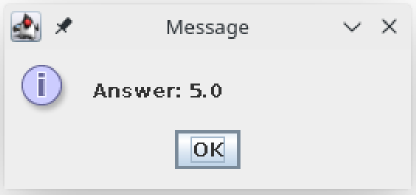
```
Answer: 0.6666666666666666
```



```
Please enter an infix numerical expression between 3 and 20 characters:
2+3
```

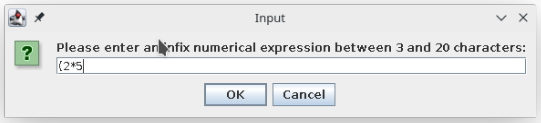


```
Postfix: 23+
```


Message

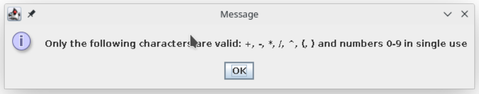
Answer: 5.0

Testing invalid input where a bracket is missing


Input

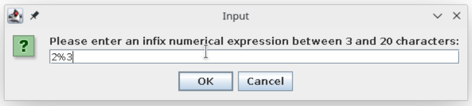
Please enter an infix numerical expression between 3 and 20 characters:

(2*5


Message

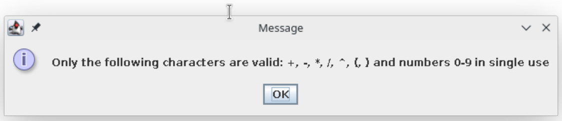
Only the following characters are valid: +, -, *, /, ^, (,) and numbers 0-9 in single use

Testing where input includes character not allowed


Input

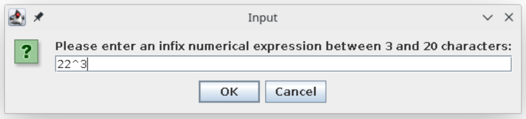
Please enter an infix numerical expression between 3 and 20 characters:

2%3


Message

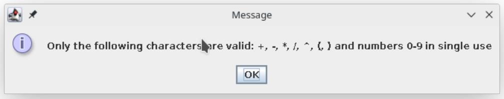
Only the following characters are valid: +, -, *, /, ^, (,) and numbers 0-9 in single use

Testing where two numbers are given in-between operators


Input

Please enter an infix numerical expression between 3 and 20 characters:

22^3


Message

Only the following characters are valid: +, -, *, /, ^, (,) and numbers 0-9 in single use