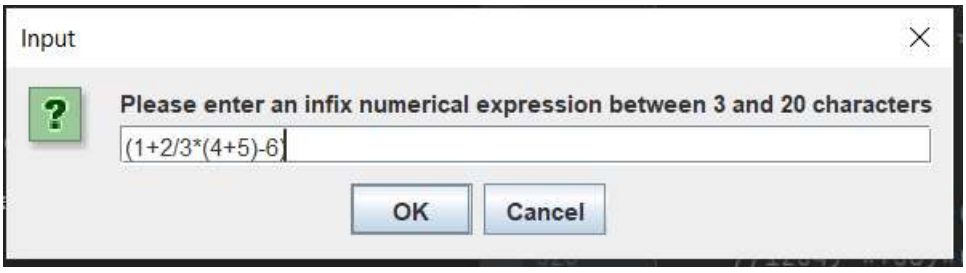
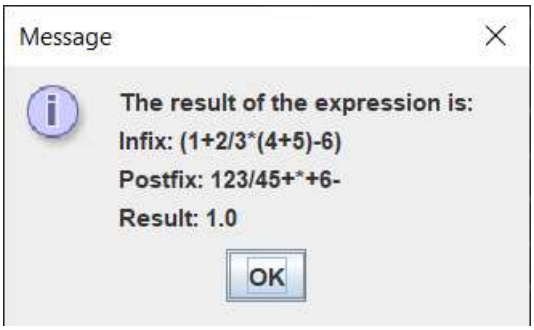
Maxwell Maia

21236277

Assignment 2

Solving Expressions in Postfix Notation using Stacks





Year 2, Semester 2

Problem Statement

A program that reads a numerical infix expression from the user, converts it to postfix using a stack and then solves the postfix expression using stacks. And Prints the final answer.

Research was done into the infix to postfix algorithm (https://youtu.be/vXPL6UavUeA).

The program only calculates single digit numerical inputs. E.g. Cannot do 21 * 400. Can do 2 * 9

And the only characters that may be input are:

+ - * / ^ ( ) 0 1 2 3 4 5 6 7 8 9

The whole infix input expression from the user should be between 3 and 20 characters long (inclusive).

The program checks whether the input is valid. If the input is invalid, the user will be prompted with an explanation as why it is invalid. The user will be able to re-enter their expression after an invalid input.

Analysis and Design Notes

The program can be divided into 3 sections

The bit that gets the infix notation from the user

The bit that turns the infix notation into postfix notation

The bit that solves the postfix notation and returns an answer

Section 1 - Get valid user input

Step 1

Get the user input

Step 2

While the user input is invalid get the user input
More detail: Checking for invalid input:
- invalid input if length of string is < 3 or length of string is > 20
- an invalid char was used not one of these: + - * / ^ ( ) 0 1 2 3 4 5 6 7 8 9
Solution for invalid input check:
Check each character for a valid char, if no valid char is found: flag it with a Boolean
If the Boolean is true OR the string is of the wrong length, the input was invalid and the user is prompted to enter a new input, after the new input, check the for invalid again. This is perfect for a while loop

Section 2 - Infix to postfix

This video was used to learn the algorithm of converting infix to postfix

https://youtu.be/vXPL6UavUeA

The video contains no code, only an on-paper step by step example of infix to postfix.

The premise is as follows
- read from an input string, one character at a time,
- detecting what the character is (operand or number), and
- place that character in either the output string (the postfix expression) or the stack, depending on precedence and brackets, furthermore I may
- move characters from the top of the stack to the output string.
This will happen because “no two operators of the same precedence can stay next to each other in the stack.”
- After the last character has been scanned and dealt with, everything remaining in the stack is placed in the output string.

inputString = infix expression

outputString = postfix expression

Flow of control

Create a new Array Stack

For every character in the inputString do:

If the character is an OPERAND (using isNumber utility method):
APPEND it to the outputString

Else: (The character is an operator)
- If the character is a ‘(‘:
  PUSH input character to Stack
- Else:
-- If the stack is empty:
   PUSH input character to Stack
-- else:
   Get the precedence of the operator on top of the Stack
   end of if
- If (the precedence of the input character is > the precedence of the operator on top of Stack) OR the top operator on the Stack is a ‘(‘:
  PUSH input character to Stack
- else: (The operator in the Stack is bigger precedence or equal precedence to the input operator)
  while the precedence of the operator on top of the Stack >= the precedence of the input operator:
    If the character on top of the Stack is a ‘(‘:

POP the '(' from the Stack to discard it
break from the while loop
end of if
(back in while loop)
POP the operator from the stack and APPEND character to output
end of while
If the input is a ')':
do nothing (this discards the ')')
else:
PUSH the input character to the Stack
end of if
end of if
end of if
If the input character is a '(':
PUSH the input character to the Stack
end of if
If the input character is a ')':
while there is a character in the stack AND the character on top of the stack is not a '(':
POP the character in the stack
APPEND the character to the outputString
End of while
Pop the character in the stack to discard the bracket
end of if
End of for every character loop

while the stack is not empty:
Pop the character in the stack
APPEND the character to the outputString
End of while

The outputString now contains the converted postfix expression

Section 3 - Evaluation of postfix expressions

Initialize an empty stack (or use same stack as before to save memory)

For every scanned element of the postfix expression

If the element is a number: Add it to the stack

If the element is an operator: pop 2 operands, then (using if statements) evaluate the operator and calculate the result and push result to stack

Pop the Double in stack, this is the final result

Now display the infix, postfix and final result

Methods:

operatorValue function for returning the precedence of an operator (utility method)

there are 3 levels of precedence

these will be represented by an integer

3 is ^

2 is * or /

1 is + or –

0 returned when the operator is not a recognized operator

Utility method

public boolean isNumber(char c)

if the character is a number (operand) then return true, else return false

Code

```
import javax.swing.JOptionPane;
public class CalculatorTest
{
   public static void main(String[] args)
   {
      CalculatorTest t = new CalculatorTest();



   }
//  precedence: ^ then */ then +-
// returns the precedence of an operator
   // 3 is the highest precedence ^
   // 2 is middle precedence * /
   // 1 is lowest precedence + -
```

```
//returns 0 if the operator is not a recognized operator
public int operatorPrecedence(char operator)
{
   int precedence = 0;
   if(operator == '^')
      precedence = 3;
   if(operator == '*' || operator == '/')
      precedence = 2;
   if(operator == '+' || operator == '-')
      precedence = 1;

   return precedence;
}

//is a number utility function
//if the character is a number (operand) then return true, else return false
public boolean isNumber(char c)
{
   if(c == '0')
      return true;
   if(c == '1')
      return true;
   if(c == '2')
      return true;
   if(c == '3')
      return true;
   if(c == '4')
      return true;
   if(c == '5')
      return true;
   if(c == '6')
```

```
        return true;
    if(c == '7')
        return true;
    if(c == '8')
        return true;
    if(c == '9')
        return true;
    //else is not a number
    return false;
}

public CalculatorTest()
{
    //This section will get a character array that only contains the legal characters
    //legal characters include + - * / ^ ( ) 0 1 2 3 4 5 6 7 8 9

    //String for input
    String userInput = "";

    //boolean will make the user re-enter input if found to be invalid
    boolean invalidCharUsed = false;

    //Get the user input as a string
    userInput = JOptionPane.showInputDialog("Please enter an infix numerical expression between
3 and 20 characters");

    //check all the characters of the string for an invalid character
    char inputChar;

    for(int i = 0; i < userInput.length(); i++)
```

```
{
    inputChar = userInput.charAt(i);
    if(inputChar == '+')
        continue;
    if(inputChar == '-')
        continue;
    if(inputChar == '*')
        continue;
    if(inputChar == '/')
        continue;
    if(inputChar == '^')
        continue;
    if(inputChar == '(')
        continue;
    if(inputChar == ')')
        continue;
    if(inputChar == '0')
        continue;
    if(inputChar == '1')
        continue;
    if(inputChar == '2')
        continue;
    if(inputChar == '3')
        continue;
    if(inputChar == '4')
        continue;
    if(inputChar == '5')
        continue;
    if(inputChar == '6')
        continue;
    if(inputChar == '7')
```

```
            continue;
        if(inputChar == '8')
            continue;
        if(inputChar == '9')
            continue;

        invalidCharUsed = true;
        break;
    }

    //System.out.println("invalidCharUsed = " + invalidCharUsed);

    //If the input is valid, the user will have to try again
    //Will exit loop when input is valid
    while (userInput.length() < 3 || userInput.length() > 20 || invalidCharUsed)
    {
        //Display the reason why the user has to try again
        if(invalidCharUsed)
        {
            //Display message if the issue was in invalid character
            JOptionPane.showMessageDialog(null,"The only valid characters are: +, -, *, /, ^, (, ), , and
numbers 0-9");
        }
        else
        {
            //Display message if the issue was invalid length
            JOptionPane.showMessageDialog(null,"Has to be between 3 and 20 characters long");

        }

        //need to be reset for next loop
```

```
        invalidCharUsed = false;


        System.out.println("Wrong input, try again.");
        System.out.println("You entered: " + userInput);


        userInput = JOptionPane.showInputDialog("Try again\nPlease enter an infix numerical
expression between 3 and 20 characters");


        //Check all the characters for an invalid character
        //if the character in the string is valid, continue to the next iteration (check next character)
        //if the character is not valid, set the invalidChar used variable to true
        for(int i = 0; i < userInput.length(); i++)
        {
           inputChar = userInput.charAt(i);
           if(inputChar == '+')
              continue;
           if(inputChar == '-')
              continue;
           if(inputChar == '*')
              continue;
           if(inputChar == '/')
              continue;
           if(inputChar == '^')
              continue;
           if(inputChar == '(')
              continue;
           if(inputChar == ')')
              continue;
           if(inputChar == '0')
              continue;
           if(inputChar == '1')
```

```
            continue;
        if(inputChar == '2')
            continue;
        if(inputChar == '3')
            continue;
        if(inputChar == '4')
            continue;
        if(inputChar == '5')
            continue;
        if(inputChar == '6')
            continue;
        if(inputChar == '7')
            continue;
        if(inputChar == '8')
            continue;
        if(inputChar == '9')
            continue;

        invalidCharUsed = true;
        break;
    }
}

//Out of loop invalid loop
//userInput string contains a valid input

System.out.println("Correct!");
System.out.println("You entered: " + userInput);

//Prefix to Postfix
```

```
//Create a stack
Stack s = new ArrayStack(20);
char tempChar = '\0';
//String tempString = "a";
//s.push(testChar);

char[] input = new char[userInput.length()];
String outputString = "";
for(int i = 0; i < userInput.length(); i++)
{
   System.out.println("====================");
   System.out.println("ITERATION " + i);

   //convert string into character array
   input[i] = userInput.charAt(i);
   System.out.println("input array character at "+i+ " is:" + input[i]);

   //Infix to postfix algorithm
   //Step 2
   //If character is an operand, add it to the output string
   System.out.println("is " + input[i] + " an operand?");
   if(isNumber(input[i])) {
      //The character is a number
      //Add it to the output string
      outputString += input[i];
      System.out.println(input[i] + " is a number");
      System.out.println("added to outputString now = " + outputString);
   }
   //Step 3.1
   //else the character must be an operator
   //If this operator is bigger than the precedence of the operator in the stack
```

```
        //OR
        //If the stack is empty.
        //OR
        //The top of the stack contains an opening parenthesis
        //Then Push the scanned character to the stack
        else
        {
           System.out.println(input[i] + " is not number");

           //ADDED off the guidelines
           //If the scanned character is a ( push it to the stack
           //else continue with handling the scanned character
           if(input[i] == '(')
           {
              s.push(input[i]);
              System.out.println("input[i] is a ( input["+i+"] : " + input[i] + " pushed to stack ");
           }
           else
           {
              System.out.println("Input is not a ( : input["+i+"] : " + input[i] + " carry on with handling
the scanned character");


              //Preparing to compare precedence
              //If the stack is empty set the precedenceStack variable to a number that will trigger the
if statement
              // that follows this if statement. precedenceStack = -10; will always be smaller than the
precedence of
              // the scanner operator // so if the stack is empty, the scanned operator will be pushed
              int precedenceStack = 0;
              if(s.isEmpty())
              {
```

```
            //If the stack is empty always push
            //This value will always make the next if condition evaluate as true
            precedenceStack = -10;
            System.out.println(input[i] + " Empty stack. precedenceStack = -10;");
        }
        else
        {
            //The stack is not empty so get the precedence
            precedenceStack = operatorPrecedence((char)s.top());
            System.out.println(input[i] + " Non empty stack. precedenceStack = " +
precedenceStack + " top stack: " + (char)s.top());
        }

        if((operatorPrecedence(input[i]) > precedenceStack) || ((char)s.top() == '(')) //are we
talking contains or on the top? //s.containsChar('(') //
        {
            //The character is an operator
            //push input[i]
            s.push(input[i]);
            System.out.println("pushed input["+i+"]: " + input[i] + " is an OPERATOR and the stack
is EMPTY OR the precedence of the scanned character is bigger than the precedence of the operator
in the stack OR the top of the stack is an (");
        }
        else //Step 3.2
        {
            //The scanned operator is the same or smaller precedence than the operator in the
stack
            System.out.println(" current " + input[i] + "is the same or smaller precedence than " + "
top stack: " + (char)s.top());
            System.out.println("Or worded differently...");
            System.out.println(" The precedence of the operator on top of the stack: " +
(char)s.top() + " is bigger or equal to the precedence of the scanned operator: " + input[i] );
```

```
            //The operator in the stack is bigger precedence or equal precedence to the scanned
operator
            //therefore we need to do something... remove all the operators that are bigger or
equal and
            // pop them. then at the end push the scanned operator to the stack
            // stop when you get to a (

            //Solution:
            //while the precedence of the operator in the stack is the same or bigger precedence
than
            // the precedence of the scanned operator
            // pop the stack and add to outputString
            // Note: the operatorPrecedence function returns a value of 0 for (
            // therefore, when a ( is encountered, the operator in the stack is smaller precedence
            // than the scanned character, the while loop will stop there

            //After while loop
            //Then push the scanned operator to the stack

            while(operatorPrecedence((char)s.top()) >= operatorPrecedence(input[i]))
            {

              //Stop this while when we reach a (
              if((char)s.top() == '(')
              {
                 //Remove the ( from the stack
                 tempChar = (char)s.pop();

                 //And leave the while loop
                 break;
              }
```

```
                System.out.println("Precedence of the operator in stack " + (char)s.top() + "
operatorPrecedence: " + operatorPrecedence((char)s.top()) );

                System.out.println("Is the same or bigger than");

                System.out.println("Precedence of the scanned operator input[i]: " + input[i] + "
operatorPrecedence: " + operatorPrecedence(input[i]));

                System.out.println("Top of stack : " + (char)s.top());

                tempChar = (char)s.pop();

                System.out.println("tempChar: " + tempChar);

                outputString += tempChar;

                System.out.println("Popped stack and added to outputString = outputString is now: "
+ outputString);

            }


            //Do not push a )

            if(input[i] == ')')

            {

                System.out.println("(char)s.top()= " + (char)s.top() );

                System.out.println("input["+i+"] = " + input[i] + " is a )");

                System.out.println("Did not push the )");

                System.out.println("End of precedence conflict: Did not push input["+i+"] = " +
input[i] + " to stack");

                System.out.println("(char)s.top()= " + (char)s.top() );

            }

            else

            {

                //Push the scanned character to the stack

                System.out.println("Want to push. input["+i+"] = " + input[i] + " is not a ), we can
push to stack");

                System.out.println("(char)s.top()= " + (char)s.top() );

                s.push(input[i]);

                System.out.println("End of precedence conflict: Pushed input["+i+"] = " + input[i] + "
to stack");

                System.out.println("(char)s.top()= " + (char)s.top() );
```

```
                }


//                  System.out.println("(char)s.top()= " + (char)s.top() );
//                  s.push(input[i]);
//                  System.out.println("End of precedence conflict: Pushed input["+i+"] = " + input[i] + " to stack");
//                  System.out.println("(char)s.top()= " + (char)s.top() );
              }
            }
          }



          //Step 4
          if(input[i] == '(')
          {
            s.push(input[i]);
          }


          //Step 5
          if(input[i] == ')')
          {
            //While there is something in the stack AND the character on top of the stack is not a '(', continue with the while loop
            //If there stack is empty returns false ending the while loop
            //else, (the stack is not empty) if the element on top of the stack is not a '(', then continue with the while loop.
            while(  (s.isEmpty())?(false):( ((char)s.top() != '(')?(true):(false) )  )
            //while(  (s.top() == null)?(false):( ((char)s.top() != '(')?(true):(false) )  )
            {
              tempChar = (char)s.pop();
              outputString += tempChar;
            }
```

```
        //and discard both parenthesis
        //discord the ')' by not storing it anywhere
        //discord the '(' by popping it from the stack and not doing anything with it
        tempChar = (char)s.pop();
    }

    if(!s.isEmpty())
    {
        System.out.println("Top of stack : " + (char)s.top());
    }
    else
    {
        System.out.println("Top of stack : stack is empty");
    }
    System.out.println("Postfix outputString : " + outputString);
    System.out.println("=====================");

}
//All of the characters of the input string has been scanned
//Step 7
//Pop and append any remaining content from the stack
while( !s.isEmpty() )
{
    outputString += (char)s.pop();
}

//Infix to postfix complete
System.out.println("Infix to postfix conversion complete");
System.out.println("Infix: " + userInput);
System.out.println("Postfix: " + outputString);
```

```
//Evaluation of postfix expressions
double op1, op2 = 0.0;
double result = 0.0;
//Create a stack to evaluate postfix, this will store operands
//We could use the previous stack since it is empty. This saves memory
//Using the same ArrayStack as before
//Stack s = new ArrayStack(20);

System.out.println("Evaluation of postfix expression");

//For every scanned element of the postfix expression
for(int i = 0; i < outputString.length(); i++)
{
   System.out.println("outputString.charAt(" + i + ") : " + outputString.charAt(i));

   //If the element is a number
   if(isNumber(outputString.charAt(i)))
   {
      //Add it to the stack
      s.push(outputString.charAt(i));
      System.out.println("Is a number, " + outputString.charAt(i) + " pushed to stack");
   }

   //If the element is an operator, pop, calculate and push result
   //Operator precedence is non-zero for all operators
   if(operatorPrecedence(outputString.charAt(i)) != 0)
   {
      System.out.println(outputString.charAt(i) + " Is an operator");
      //Pop two operands
      //Operands in stack may be a Double because they are a result from an earlier calculation
      //If the top of the stack contains a double, convert it in a different way when popping
```

```
//(double) Character.digit((char)s.pop(), 10) converts the popped object to a double
if(s.top() instanceof Double)
{
   //Double in stack
   op1 = (double) s.pop();
}
else
{
   //Character or object in stack
   op1 = (double) Character.digit((char)s.pop(), 10); //(double) s.pop();
}

if(s.top() instanceof Double)
{
   //Double in stack
   op2 = (double) s.pop();
}
else
{
   //Character or object in stack
   op2 = (double) Character.digit((char) s.pop(), 10); //(double) s.pop();
}

//Evaluate operator
//Calculate the result based on the operator
if(outputString.charAt(i) == '+')
{
   System.out.println(outputString.charAt(i) + " Is a +");
   result = op2 + op1;
}
else if (outputString.charAt(i) == '-')
```

```
        {
            System.out.println(outputString.charAt(i) + " Is a -");
            result = op2 - op1;
        }
        else if (outputString.charAt(i) == '*')
        {
            System.out.println(outputString.charAt(i) + " Is a *");
            result = op2 * op1;
        }
        else if (outputString.charAt(i) == '/')
        {
            System.out.println(outputString.charAt(i) + " Is a /");
            result = op2 / op1;
        }
        else if (outputString.charAt(i) == '^')
        {
            System.out.println(outputString.charAt(i) + " Is a ^");
            result = Math.pow(op2, op1);
        }
        else
        {
            System.out.println(outputString.charAt(i) + " Is a else");
            result = 999;
        }

        //Push the result back to the stack
        System.out.println("Pushed " + result + " to the stack, result");
        s.push(result);
    }
}
```

```
    double answer = (double) s.pop();
    System.out.println("The answer is: " + answer);

    String messageOutput = "The result of the expression is:";
    messageOutput += "\nInfix: " + userInput;
    messageOutput += "\nPostfix: " + outputString;
    messageOutput += "\nResult: " + answer;

    JOptionPane.showMessageDialog(null, messageOutput);
}

//Try
//(2+3)*(4+5)
//Will be
//23+45+*
//I got
//23)+45)+(*(
//23)+45)+(*
//23)+45)+(*(
//Finally 23+45+*

//Try
//(1+2*3-4)/(5*6)
//Will be
//123*+4-56*/
//I got
//1234)-*+56)*(/
//1234)-*+56)*(/
//Finally 123*+4-56*/

//Try
```

```
    //(1+2/3*(4+5)-6)
    //Will be
    //123/45+*+6-
    //I got
    //123/45+*+6-
    //through lots of system.outs and stepping through line by line with a technical example found in a youtube video
    //Youtube video had no code. it just explained the algorithm
    //https://youtu.be/vXPL6UavUeA



}
```

Testing

1. Valid input

Input: (1+2/3*(4+5)-6)

Expected result:

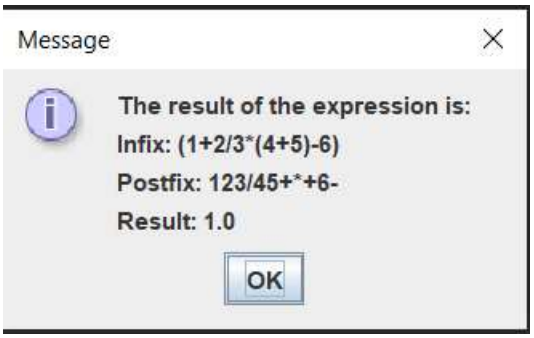
Infix: (1+2/3*(4+5)-6)

Postfix: 123/45+*+6-

Result: 1.0

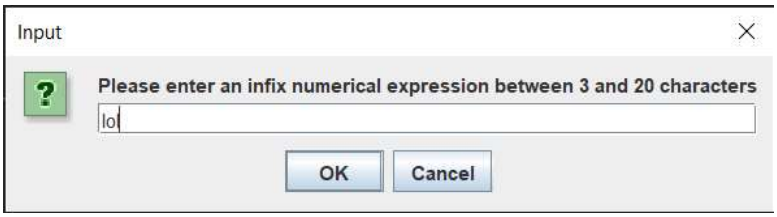
Actual result:

Test success



2. Invalid input – characters included

Input: lol



Expected result:

Message showing the only valid characters should appear

Another input dialog should appear afterward to allow a retry

Actual result:

Followed by

Test success

3. Invalid input – not enough digits/operators

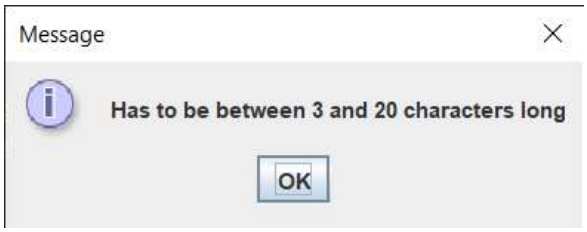
Input: 1

Expected result:

Message explaining what the length of the input should be

Another input dialog should appear afterward to allow a retry

Actual result:

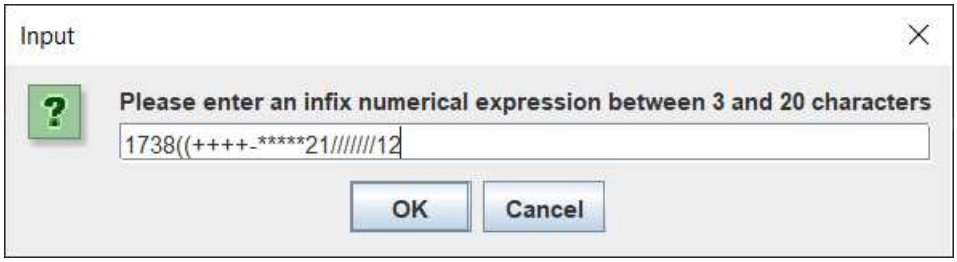


Followed by

Test success

4. Invalid input – too many digits/operators

Input: 1738((++++-*****21///////12

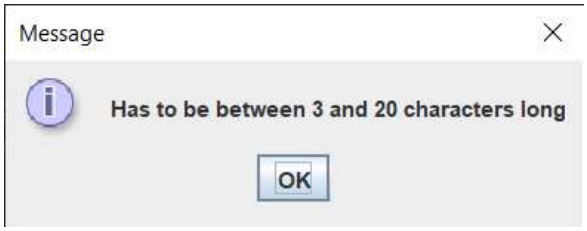


Expected result:

Message explaining what the length of the input should be

Another input dialog should appear afterward to allow a retry

Actual result:

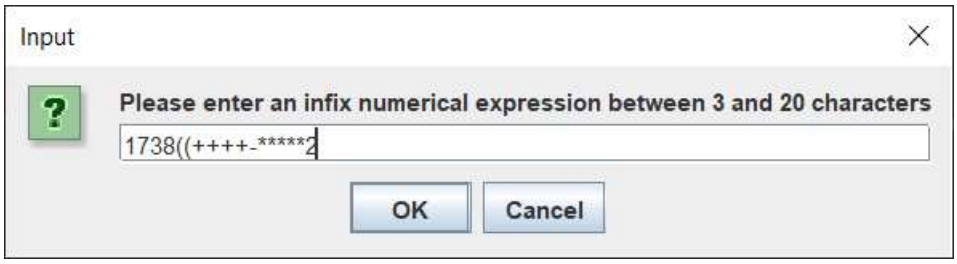


Followed by

Test success

5. Invalid input – valid inputs but mathematically invalid

Input: 1738((++++-*****2

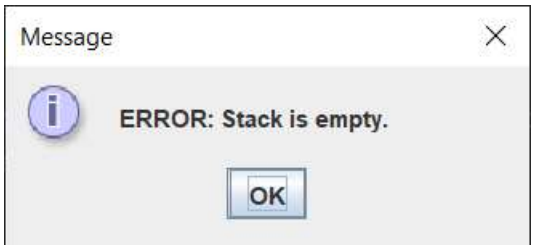


Expected result:

Message explaining that the input was mathematically invalid

Another input dialog should appear afterward to allow a retry

Actual result: program crashes due to a runtime error



```
Exception in thread "main" java.lang.NullPointerException Create breakpoint : Cannot invoke "java.lang.
    at CalculatorTest.<init>(CalculatorTest.java:449)
    at CalculatorTest.main(CalculatorTest.java:6)
```

Exceptions could be handled to show a message saying that there was a problem

And the another input dialog could appear afterward to allow a retry

6. Valid input – works if the first input was invalid

Input: (1+2*3-4)/(5*6)

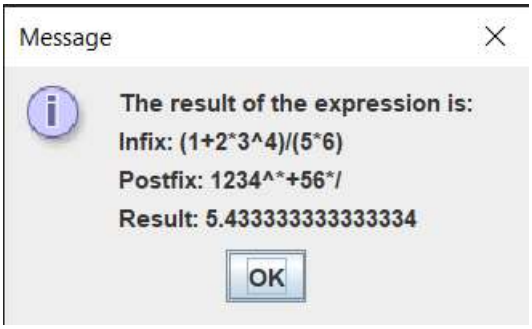
Expected result:

Infix: (1+2*3^4)/(5*6)

Postfix: 1234^*+56*/

Result: 5.43 recurring

Actual result:



Test success