Project Documentation

Name: Dan Beck

Assignment: Project 1

Date: September 1, 2020

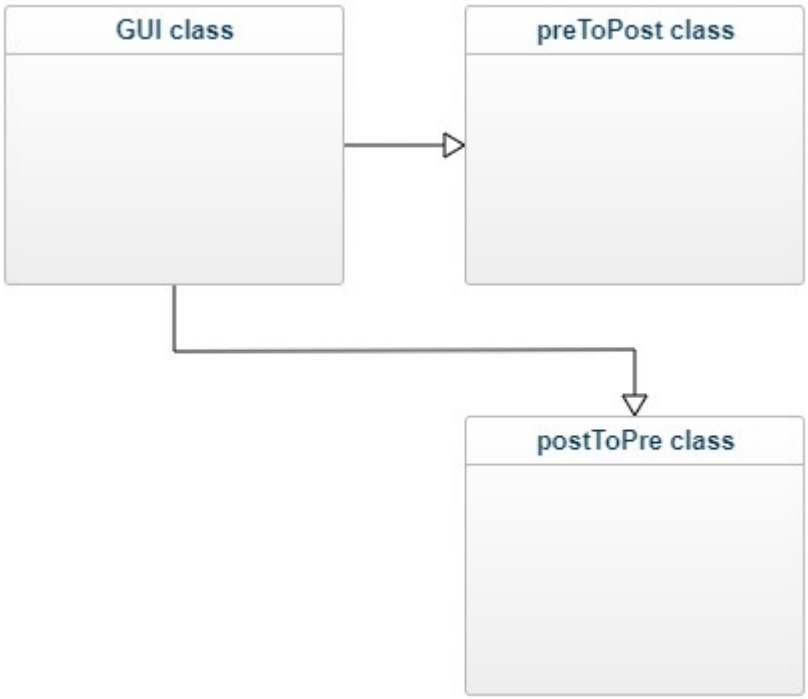
Problem Statement: The writing a program that converts prefix expressions to postfix and postfix expressions to prefix.

Analysis: Expressions used:

Posfix to prefix test – 2 2 12 9*2-+/

Prefix to postfix test - * + * 2 / 2 -+ 12 9 2

Design:



Code:

```
package BeckProject1;


import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import java.awt.Rectangle;
import java.awt.Font;

/* File: Project 1 - GUI
 * Author: Dan Beck
 * Date: August 29, 2020
 * Purpose: Class that generates the GUI and passes parameters to
 *                          other classes.
 */

public class GUI
{
        public GUI()
        {
                //***************Frame*****************

                //Generates the JFrame
                JFrame frame = new JFrame();
                frame.setBounds(new Rectangle(600, 400, 450, 175));
```

```
frame.setTitle("Expression Converter");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.getContentPane().setLayout(null);

//***************Text Fields*****************

//Generates the text field to see the results
JTextField resultText = new JTextField();
resultText.setEditable(false);
resultText.setColumns(10);
resultText.setBounds(125, 95, 285, 20);
frame.getContentPane().add(resultText);
frame.setVisible(true);

//Generates the text field to enter expression
JTextField expressionText = new JTextField();
expressionText.setBounds(125, 10, 285, 20);
frame.getContentPane().add(expressionText);
expressionText.setColumns(10);


//***************Buttons*****************

//Generates the prefix to postfix button
JButton prefixToPostfixButton = new JButton("Prefix to Postfix");
```

```
prefixToPostfixButton.setFont(new Font("Tahoma", Font.BOLD, 11));
prefixToPostfixButton.setBounds(70, 45, 130, 40);
frame.getContentPane().add(prefixToPostfixButton);
prefixToPostfixButton.addActionListener(new ActionListener()
{
	public void actionPerformed(ActionEvent e)
	{
		try
		{
			String expression = expressionText.getText();
			preToPost preTp = new preToPost();
			resultText.setText(preTp.preToPost(expression));
		}//end try
		catch(Exception e1)
		{
			JOptionPane.showMessageDialog(null, "Please enter a
valid Prefix expression!");
		}//end catch
	}//end Action Performed
});//addActionListener

//Generates the postfix to prefix button
JButton postfixToPrefixButton = new JButton("Postfix to Prefix");
postfixToPrefixButton.setFont(new Font("Tahoma", Font.BOLD, 11));
postfixToPrefixButton.setBounds(230, 45, 130, 40);
```

```
frame.getContentPane().add(postfixToPrefixButton);
postfixToPrefixButton.addActionListener(new ActionListener()
{
	public void actionPerformed(ActionEvent e)
	{
		try
		{
			String expression = expressionText.getText();
			postToPre postTp = new postToPre();
			resultText.setText(postTp.postToPre(expression));
		}//end try
		catch(Exception e2)
		{
			JOptionPane.showMessageDialog(null, "Please enter a
valid Postfix expression!");
		}//end catch
	}//end Action Performed
});//addActionListener

//***************Labels*****************
//Generates the results label
JLabel resultLabel = new JLabel("Result:");
resultLabel.setFont(new Font("Tahoma", Font.BOLD, 12));
resultLabel.setBounds(70, 95, 55, 20);
frame.getContentPane().add(resultLabel);
```

```
		//Generates the Enter Expression label
		JLabel enterExpressionLabel = new JLabel("Enter Expression:");
		enterExpressionLabel.setFont(new Font("Tahoma", Font.BOLD, 12));
		enterExpressionLabel.setBounds(10, 10, 120, 20);
		frame.getContentPane().add(enterExpressionLabel);
	}

	public static void main(String[] args)
	{
		// TODO Auto-generated method stub
		new GUI();
	}//End Main
}//End GUI class

package BeckProject1;

import java.util.Stack;
import java.util.StringTokenizer;

/* File: Project 1
 * Author: Dan Beck
 * Date: August 29, 2020
 * Purpose: Class that receives Prefix expression and converts it
 *                          to a Postfix expression.
```

```
 */

class preToPost
{
	//function that checks if character is an operator
	static boolean isOperator(String op)
	{
		switch (op)
		{
		case "+":
		case "-":
		case "/":
		case "*":
			return true;
		}//end switch (check)
		return false;
	}//end static boolean isOperator(String check)

	//Converts Prefix to Postfix
	String preToPost(String convert)
	{
		//New string to be generated
		StringBuffer newString = new StringBuffer();

		//Sets the first character of the new string
```

```
newString.append(convert.charAt(0));

for (int i = 1, n = convert.length(); i < n; i++)
{
        //Checks if character is a space
        if(Character.isSpaceChar(convert.charAt(i)))
        {
                newString.append(convert.charAt(i));
        }
        //checks if character is a digit with an operator before it
        else if(isOperator(String.valueOf(convert.charAt(i-1))) == true &&
                        Character.isDigit(convert.charAt(i)))
        {
                newString.append(" " + convert.charAt(i));
        }
        //checks if character is a operator with a digit before it
        else if(Character.isDigit(convert.charAt(i-1)) &&
                        isOperator(String.valueOf(convert.charAt(i))) == true)
        {
                newString.append(" " + convert.charAt(i));
        }
        //checks if character is an operator with an operator before it
        else if(isOperator(String.valueOf(convert.charAt(i))) == true &&
                        isOperator(String.valueOf(convert.charAt(i-1))) == true)
        {
```

```
                newString.append(" " + convert.charAt(i));
        }
        //checks if character is a digit
        else if (Character.isDigit(convert.charAt(i)))
        {
                newString.append(convert.charAt(i));
        }
        //Passes the character through if none others are met
        else
        {
                newString.append(convert.charAt(i));
        }
}//end for (int i = 0, n = convert.length(); i < n; i++)

//tokenize the string containing the prefix expression
StringTokenizer st = new StringTokenizer(convert);

//two stacks to perform the conversions
Stack<String> rs = new Stack<String>();
Stack<String> s = new Stack<String>();

//read the tokens
while (st.hasMoreTokens() == true)
{
        rs.push(st.nextToken());
```

```
}//end while (st.hasMoreTokens() == true)

while (rs.empty() == false)
{
        String check = rs.pop();

        // check if symbol is operator
        if (isOperator(check) == true)
        {
                // pop two operands from stack
                String n1 = s.peek(); s.pop();
                String n2 = s.peek(); s.pop();

                // concats the operands and operator
                String makeNew = n1 + n2 + check;

                // Push makeNew back to stack
                s.push(makeNew + " ");
        }//end if (isOperator(check) == true)

        // if symbol is an operand
        else
        {
                //push the operand to the stack
                s.push(check + " ");
```

```
                }//end else
            }//end while (rs.empty() == false)

            //shows the stack containing only the Postfix expression
            return s.peek();
        }//end String preToPost(String pre_exp)
}// end class preToPost

package BeckProject1;

import java.util.Stack;
import java.util.StringTokenizer;

/* File: Project 1 - postToPre
 * Author: Dan Beck
 * Date: August 29, 2020
 * Purpose: Class that receives Postfix expression and converts it
 *                to a Prefix expression.
 */

public class postToPre
{
        //function that checks if character is an operator
        static boolean isOperator(String op)
        {
```

```
        switch (op)
        {
        case "+":
        case "-":
        case "/":
        case "*":
                return true;
        }//end switch (check)
        return false;
}//end static boolean isOperator(String check)

//Converts Postfix to Prefix
String postToPre(String convert)
{
        //New string to be generated
        StringBuffer newString = new StringBuffer();

        //Sets the first character of the new string
        newString.append(convert.charAt(0));

        for (int i = 1, n = convert.length(); i < n; i++)
        {
                //Checks if character is a space
                if(Character.isSpaceChar(convert.charAt(i)))
                {
```

```
        newString.append(convert.charAt(i));
}
//checks if character is a digit with an operator before it
else if(isOperator(String.valueOf(convert.charAt(i-1))) == true &&
                Character.isDigit(convert.charAt(i)))
{
        newString.append(" " + convert.charAt(i));
}
//checks if character is a operator with a digit before it
else if(Character.isDigit(convert.charAt(i-1)) &&
                isOperator(String.valueOf(convert.charAt(i))) == true)
{
        newString.append(" " + convert.charAt(i));
}
//checks if character is an operator with an operator before it
else if(isOperator(String.valueOf(convert.charAt(i))) == true &&
                isOperator(String.valueOf(convert.charAt(i-1))) == true)
{
        newString.append(" " + convert.charAt(i));
}
//checks if character is a digit
else if (Character.isDigit(convert.charAt(i)))
{
        newString.append(convert.charAt(i));
}
```

```
        //Passes the character through if none others are met
        else
        {
                newString.append(convert.charAt(i));
        }
}//end for (int i = 0, n = convert.length(); i < n; i++)

//tokenize the string containing the postfix expression
StringTokenizer st = new StringTokenizer(newString.toString());

//one stack to perform the conversions
Stack<String> s = new Stack<String>();



//read tokens
while (st.hasMoreTokens() == true)
{
        String check = st.nextToken();

        //check if symbol is operator
        if (isOperator(check) == true)
        {
                //pop two operands from stack
                String n1 = s.peek(); s.pop();
                String n2 = s.peek(); s.pop();
```

```
                //concats the operands and operator
                String makeNew = check + " " + n2 + n1;

                //add makeNew to stack
                s.push(makeNew);
            }//end if (isOperator(check) == true)

            //if symbol is an operand
            else
            {
                //push the operand to the stack
                s.push(check + " ");
            }//end else
        }//end while (st.hasMoreTokens() == true)

        //shows the stack containing only the Prefix expression
        return s.peek();
    }//end String preToPost(String pre_exp)
}// end class postToPre
```

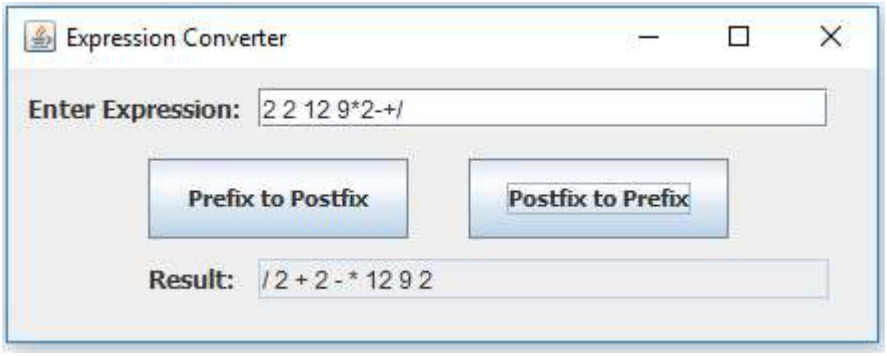
Testing:

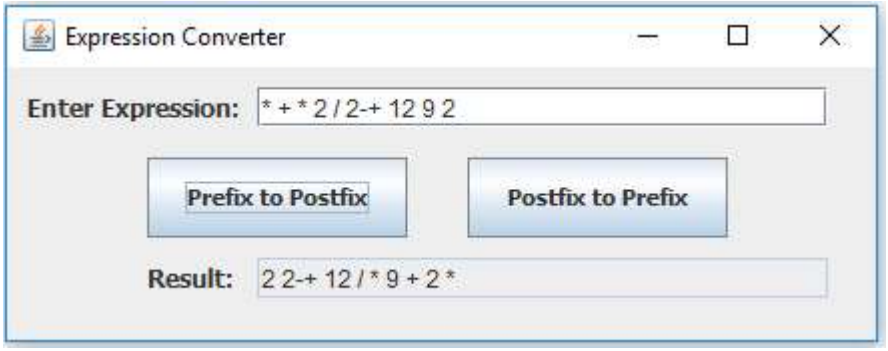
Expressions used:

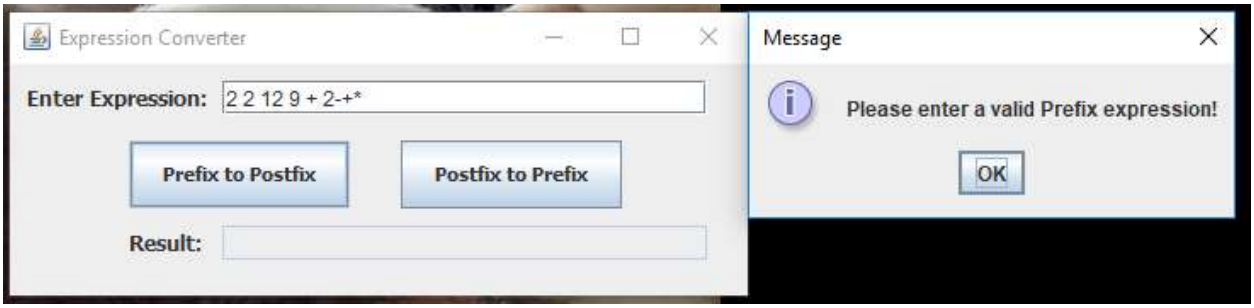
Posfix to prefix test – 2 2 12 9*2-+/

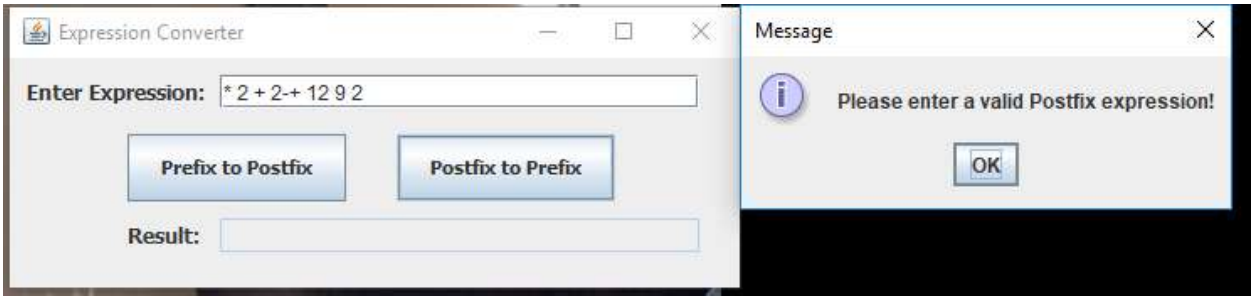
Prefix to postfix test - * + * 2 / 2 -+ 12 9 2

Output:


Expression Converter
Enter Expression:
2 2 12 9*2-+/
Prefix to Postfix
Postfix to Prefix
Result:
/ 2 + 2 - * 12 9 2


Expression Converter
Enter Expression:
* + * 2 / 2-+ 12 9 2
Prefix to Postfix
Postfix to Prefix
Result:
2 2-+ 12 / * 9 + 2 *


Expression Converter
Enter Expression:
2 2 12 9 + 2-+*
Prefix to Postfix
Postfix to Prefix
Result:
Message
Please enter a valid Prefix expression!
OK


Expression Converter
Enter Expression:
* 2 + 2-+ 12 9 2
Prefix to Postfix
Postfix to Prefix
Result:
Message
Please enter a valid Postfix expression!
OK

Reflection: For this project, I learned a few new concepts of programming. The first being the algorithm of converting prefix and postfix expressions. I have never heard of these before this project, and while I still do not feel that I have fully grasped how they work, it is very interesting. The other key concept that I learned was the StringTokenizer. I never used this class before, but this project allowed me to see how useful they are.