# CMSC 204
# Assignment #2
# Notation

**Infix notation** is the **notation** commonly used in arithmetical and logical formulae and statements. It is characterized by the placement of operators between operands – "infixed operators" – such as the plus sign in "2 + 2".

**Postfix notation** is a **notation** for writing arithmetic expressions in which the operands appear before their operators. There are no precedence rules to learn, and parentheses are never needed.

**You will be creating a utility class that converts an infix expression to a postfix expression, a postfix expression to an infix expression and evaluates a postfix expression.**

The GUI is provided.

**Concepts tested:**

Generic Queue

Generic Stack

Exception handling

**Data Element**

String

**Data Structures**

1. Create a generic queue class called *MyQueue*. *MyQueue* will implement the *QueueInterface* given you. You will be creating MyQueue from scratch (do not use an internal object of the Queue class from java.util)
2. Create a generic stack class called *MyStack*. *MyStack* will implement the *Stack Interface* given you. You will be creating MyStack from scratch (do not use an internal object of the Stack class from java.util)

**Utility Class**

The *Notation* class will have a method infixToPostfix to convert infix notation to postfix notation that will take in a string and return a string, a method postfixToInfix to convert postfix notation to infix notation that will take in a string and return a string, and a method to evaluatePostfix to evaluate the postfix expression. It will take in a string and return a double.

**In the infixToPostfix method, you MUST use a queue for the internal structure that holds the postfix solution. Then use the toString method of the Queue to return the solution as a string.**

For simplicity sake:

a. operands will be single digit numbers
b. the following arithmetic operations will be allowed in an expression:

| | |
|---|---|
| + | addition |
| - | subtraction |
| * | multiplication |
| / | division |

**Exception Classes**

Provide the following exception classes:

1. InvalidNotationFormatException – occurs when a Notation format is incorrect
2. StackUnderflowException – occurs when a top or pop method is called on an empty stack.
3. StackOverflowException– occurs when a push method is called on a full stack.
4. QueueUnderflowException  – occurs when a  dequeue method is called on an empty queue.
5. QueueOverflowException – occurs when a enqueue method is called on a full queue.

**GUI Driver (provided)**

1. Initially neither radio button for notation is selected. When a radio button is selected, the Convert button is enabled and the appropriate label and field are visible for the user input.
2. When the user selects the Convert button, the appropriate label and field with the conversion will be displayed.
3. When the user selects the Evaluate button, the "answer" to the expression will be displayed.
4. When the Exit button is selected, the application will close.

## ALGORITHMS

### ***Infix expression to postfix expression:***

Read the infix expression from left to right and do the following:

If the current character in the infix is a space, ignore it.
If the current character in the infix is a digit, copy it to the postfix solution queue

If the current character in the infix is a left parenthesis, push it onto the stack
If the current character in the infix is an operator,

1. Pop operators (if there are any) at the top of the stack while they have equal or higher precedence than the current operator, and insert the popped operators in postfix solution queue
2. Push the current character in the infix onto the stack

If the current character in the infix is a right parenthesis

1. Pop operators from the top of the stack and insert them in postfix solution queue until a left parenthesis is at the top of the stack, if no left parenthesis- throw an error
2. Pop (and discard) the left parenthesis from the stack

When the infix expression has been read, Pop any remaining operators and insert them in postfix solution queue.

***<u>Postfix expression to infix expression</u>:***

Read the postfix expression from left to right and to the following:

If the current character in the postfix is a space, ignore it.
If the current character is an operand, push it on the stack
If the current character is an operator,

1. Pop the top 2 values from the stack. If there are fewer than 2 values throw an error
2. Create a string with 1st value and then the operator and then the 2nd value.
3. Encapsulate the resulting string within parenthesis
4. Push the resulting string back to the stack

When the postfix expression has been read:

If there is only one value in the stack – it is the infix string, if more than one value, throw an error

***<u>Evaluating a postfix expression</u>***

Read the postfix expression from left to right and to the following:

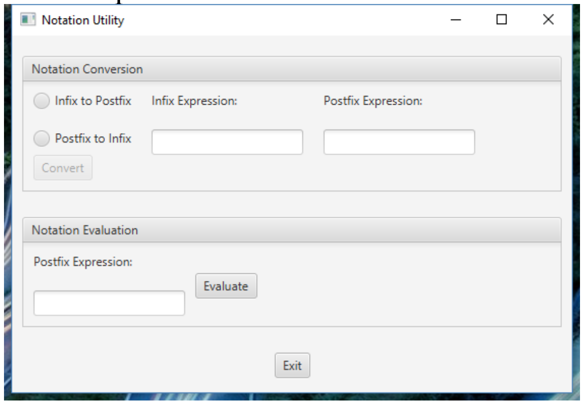
If the current character in the postfix expression is a space, ignore it.
If the current character is an operand ~~or left parenthesis~~, push on the stack
If the current character is an operator,

1. Pop the top 2 values from the stack. If there are fewer than 2 values throw an error
2. Perform the arithmetic calculation of the operator with the first popped value as the right operand and the second popped value as the left operand
3. Push the resulting value onto the stack

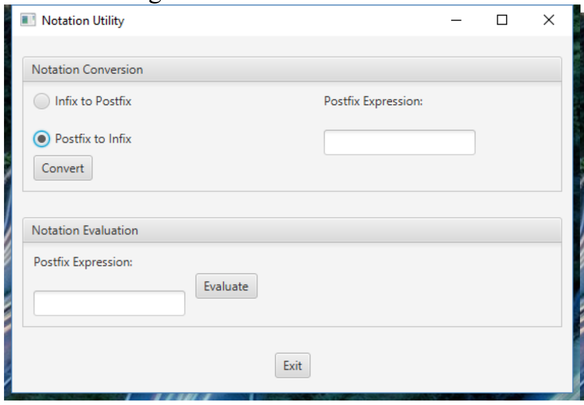
```
When the postfix expression has been read:
    If there is only one value in the stack – it is the result of the postfix expression, if
    more than one value, throw an error
```

**Examples:**

At startup

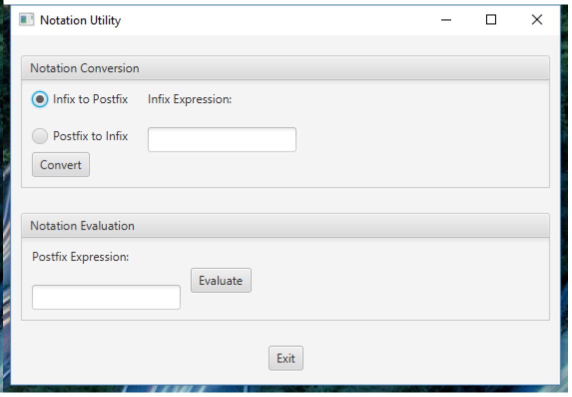


After selecting Infix to Postfix



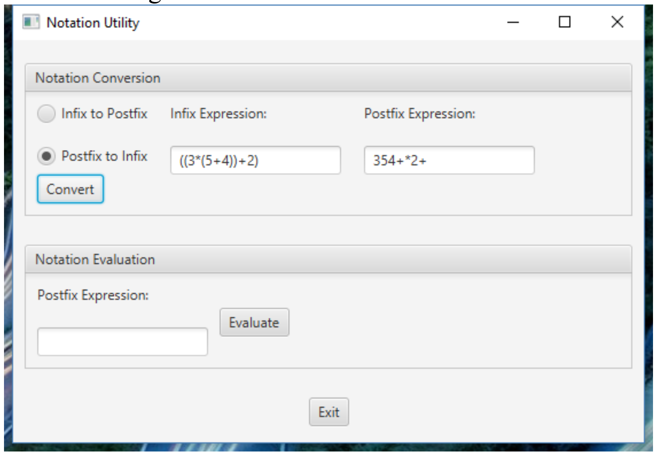
Infix to Postfix [✓ Postfix to Infix]

After selecting Postfix to Infix



[✓ Infix to Postfix] Postfix to Infix

After selecting Convert Button

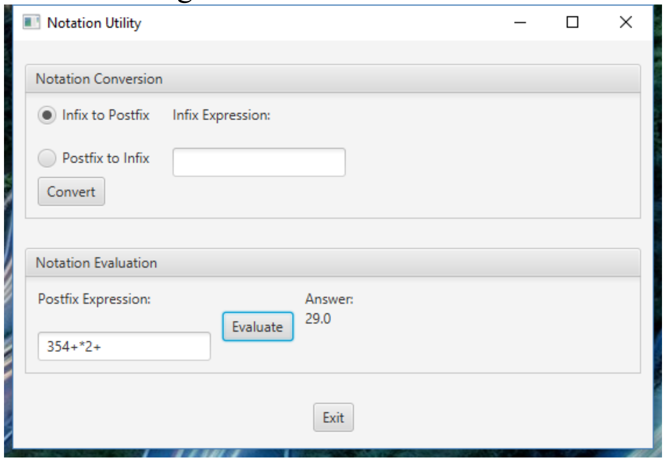


Infix to Postfix [✓ Postfix to Infix]

Infix Expression: ((3*(5+4))+2)

Postfix Expression: 354+*2+

After selecting Evaluate button



(●) Infix to Postfix

Postfix Expression: 354+*2+

Answer: 29.0

# Grading Rubric - CMSC 204 Project #2

Name ______________________________ .

| | |
|---|---|
| **TESTING** (100 pts) | |
| Passes public JUnit tests | 20 pts _____ |
| Passes STUDENT JUnit tests | 5 pts _____ |
| Passes private instructor tests | 75 pts _____ |
| Possible Sub-total | 100 pts _____ |
| **REQUIREMENTS** (Subtracts from Testing total) | |
| **Documentation:** | |
| Javadoc was not provided for all student generated classes | - 5 pts _____ |
| Documentation within source code was missing or incorrect<br>Description of what class does was missing<br>Author's Name, @author, was missing<br>Methods not commented properly using Javadoc @param, @return<br>Javadoc matches submitted java files | - 5 pts _____ |
| JUnit STUDENT methods were not implemented | -5 pts _____ |
| Learning Experience (text document)<br>In 3+ paragraphs, highlight your lessons learned and learning experience from working on this project. What have you learned? What did you struggle with? What would you do differently on your next project? | -4 pts _____ |
| **Programming Style:** | |
| Incorrect use of indentation, statements, structures | -10 pts _____ |
| **Design:** Classes do not have the functionality specified, i.e., | |
| 1. Data Structures classes<br>• Generic Stack class<br>• Generic Queue class | - 16 pts_____ |
| 2. Utility class - Notation<br>• Does not follow provided Javadoc<br>• Does not correctly handle exceptions | -15 pts_____ |
| Possible decrements: | -60 pts _____ |
| Possible total grade: | 100 pts _____ |