# Project 2 | Notation

**Implement a utility class that will –**

* **Convert Infix to Postfix expressions**
* **Convert Postfix to Infix expressions**
* **Evaluate Postfix expressions**

**Infix notation** – A **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** – a **notation** for writing arithmetic expressions in which the operands appear before their operators. There are no precedence rules to learn, and parentheses are not needed.

**Academic Honesty Policy** – **Do your own work.** The Dept. will valid your project submission in order to ensure that it meets the Academic Integrity Policy.

**Good Faith Attempt (GFA) & Project Expectations – Review the GFA and Project Expectations posts on Bb.** See your instructor if you are not clear regarding these expectations.

**Testing –** Have you tested your code thoroughly using the provided GUI? You are NOT done with your project until you have done so. That’s what your instructor will do.

**Expectations**

* Your project **must implements** the provided Java Interfaces in order to satisfy the GFA
  + **Do not** modify any of the provided Interfaces. Seek clarifications if needed – you do not need to modify them
* Anticipate both **valid and invalid expressions**. Here are some examples:
* **(5+4** // missing a parenthesis
* **5+\*4** // extra operator
* **5 + 4** // valid expression
* **(5+4)** // also valid
* **5 / 3** // result is 1. It’s an integer expression. Result is not 1.6666
* **5 % 3** // result is 2
* Others // for you to think about

**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)

1. 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’s sake:

* 1. operands will be single digit numbers
  2. the following arithmetic operations will be allowed in an expression:
* addition (+)
* subtraction (-)
* multiplication (\*)
* division (/)
* **modulus (%) // It has the same precedence as \* and /**

**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.

# Suggested 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 parenthesisthrow 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.

***Postfix expression to infix expression:***

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

***Evaluating a postfix expression***

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,

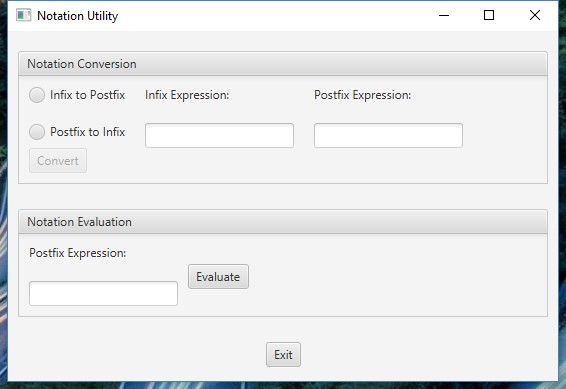
* 1. Pop the top 2 values from the stack. If there are fewer than 2 values throw an error
  2. Perform the arithmetic calculation of the operator with the first popped value as the right operand and the second popped value as the left operand 3. Push the resulting value onto the stack

When the postfix expression has been read:

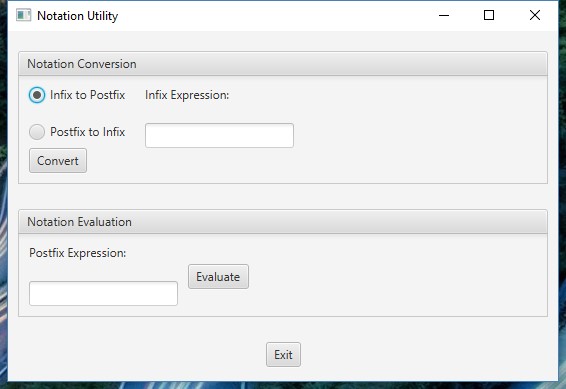
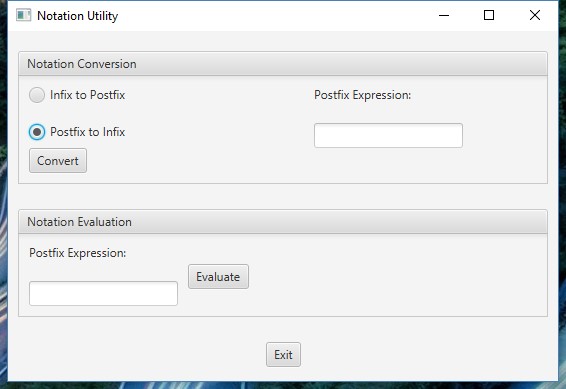
If there is only one value in the stack – it is the result of the postfix expression, if more than one value, throw an error

**These are sample runs only.** Use them as a guide as your project requirements is different.

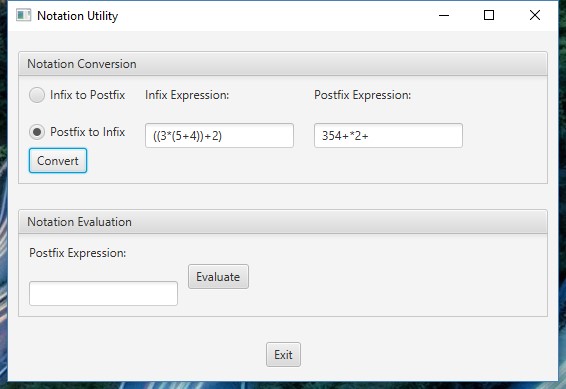
At startup



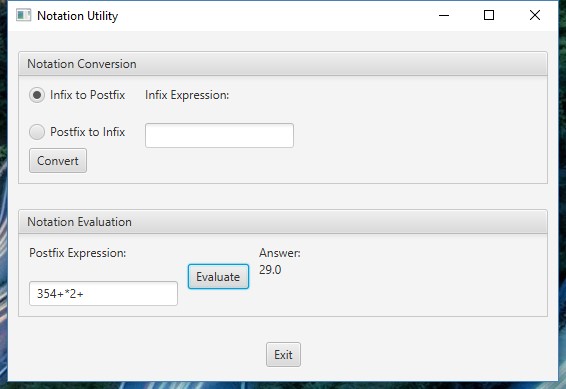
After selecting Infix to Postfix After selecting Postfix to Infix



After selecting Convert Button



After selecting Evaluate button



# Write-up & Submission Requirements

Your write-up, at a minimum, must address the following -

* **Approach, design & algorithm**
  + **DO NOT** start coding your project immediately! Come up with a high-level design first, and then write your code
    - Break the project into smallest modules where applicable
  + Each student is welcome to expand on the design. If your project includes “above and beyond” features, clearly articulate them in your write-up. Potential extra credit could be earned if that is the case
* **Test plan, Junit test cases and JavaFX runs** 
  + Screen capture numerous screenshots of the actual “runs” as you are test your project (just like those included in this write-up)
  + I need to see your Junit tests and JavaFX runs
  + What test cases did you run?
  + What were the actual outputs (in the JavaFX GUI) when you ran it?
  + What were the expected vs. actual outputs?
  + Will your project be able to pass a set of private test cases?
* Highlight your learning experience and lessons learned
* Assumptions that you made
* Anything else that I need to know?

Each student must submit one zip (compressed) file back to the Assignment (link) with the following:

* Your Eclipse project folder (preferred), or source code (java files)
* A write-up (a Word or PDF file)
* Name your compressed file as <lastname>\_project\_x
  + where x is the project number (e.g., Thai\_Project\_1.zip)
* **I can only grade what’s being submitted. Double check your submission**

**I MUST BE able to compile, run and test every submission on my computer.** Just as important, I will READ your write-up first

Not clear? That’s okay, but do ask your questions. **“I did not know” or “I did not understand” is not good enough.**

Start working on each project immediately so that we can discuss any concerns or questions you have.