2217 Java Software Dev 2
Project 7 – Postfix Calculator

Purpose

Create an abstract data type that behaves like a stack.

Goal

Write a GUI 4-function postfix calculator application.

Project Overview

According to Copilot:

> *"Postfix expressions, or Reverse Polish Notation (RPN), place operators after operands. This stack-compatible format removes the need for parentheses, simplifying expression evaluation."*

AI Tip: Research postfix expressions. Study the examples until you understand how they work.

A stack is the ideal data structure for solving postfix expressions. Given a string containing a postfix expression, here is the algorithm for solving the expression:

```
Break the string into tokens. Each token is either an operand or an
operator.
For each token:
    If the token is an operand, push it onto the stack
    If the token is an operator (+, -, *, or /),
        Pop the stack twice
        Do the calculation
        Push the result onto the stack
End loop
Pop the stack to get the final result.
```

Program Requirements

Download the starter project provided with the assignment. It includes all necessary files, including pre-written GUI code. You are required to complete the implementation tasks marked with `TODO:` comments. **You must use the starter project as-is**—projects built from scratch will not be accepted.

In this exercise, you will implement a custom stack abstract data type (ADT) in the `StackADT` class. This is purely a learning exercise, given that the Java library already contains a Stack class. Instead of building stack behavior from the ground up, you will use an instance of `java.util.LinkedList` as the underlying storage mechanism. Think of this as wrapping a stack interface around a linked list.

The stack needs the following behaviors:

`push`	- Adds data of type `Object` to the stack.
`pop`	- Removes and returns the top element from the stack.
`peek`	- Returns the top element from the stack without removing it.
`isEmpty`	- Returns true if the stack is empty, otherwise false.
`count`	- Returns the total number of elements in the stack.

Each of these methods will be short because all they must do is call one of the linked list methods. The question you must answer is, "Which element in the list is the top element of the stack? The first one? The last one?" Note that all operations must be performed in O(1) time, so that will determine which linked list element you choose to be the top of the stack.

The core logic for this project resides in the `ExpressionUtils` class. Within it, you'll find a static method named `postfixCalc`, which takes a mathematical expression string as its input parameter.

This class serves as a utility library for processing mathematical expressions. While you're currently implementing `postfixCalc`, the design anticipates future additions—such as a `prefixCalc` method—so think of `ExpressionUtils` as a growing repository of expression-related functionality.

Implement the algorithm described in the "Project Overview" section of this document. Be sure to follow the defined steps carefully to ensure the correct evaluation of the postfix expression.

AI Tip: If you're unsure how to break a string into individual tokens, research how to split a string into an array using spaces as delimiters. This is essential for parsing each element of the expression correctly.

To speed up the process, most of the GUI functionality has already been implemented in the `CalculatorFrame` class. You are welcome to enhance the visual design or layout as you see fit.

Note that the event handling logic for the button is currently incomplete. You are responsible for implementing this missing functionality to ensure a correct response to user input.

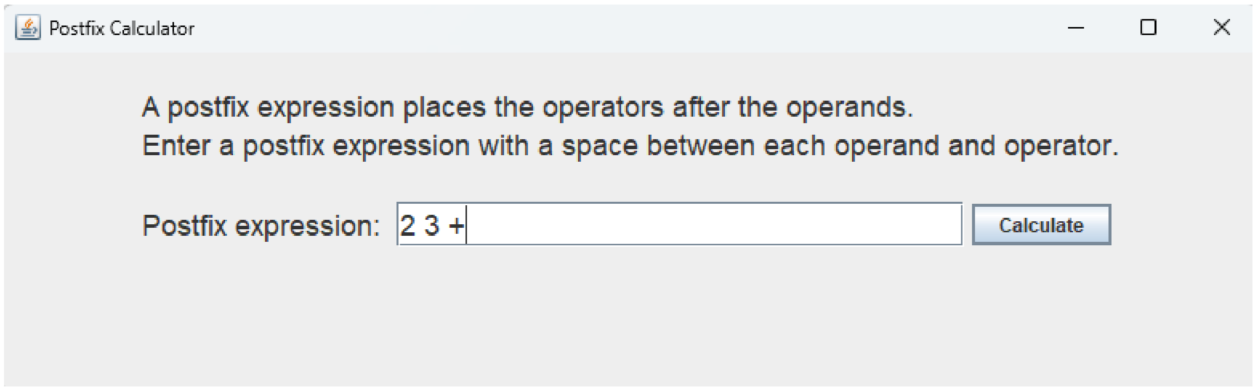
Users should be able to enter a postfix expression and press the <`Enter`> key to evaluate it without needing to click the Calculate button. This feature improves usability and workflow efficiency. *AI Tip*: Research how to designate a button on a form as the default button.

The postfix calculator needs to be reasonably robust. We won't trap every possible error, but we should be able to handle the following common errors:

Input	**Error**
A 2 +	'A' is not a number.
1 + -	There are too many operators.
1 2 3 +	There are too many operands.

The first two examples will automatically generate an exception. The third example will not. This is a bit tricky. How can we tell if there are too many operands? In that case, what will be the state of our stack at the end? Once your code determines that there are not enough operands, it should throw an `ArithmeticException`.

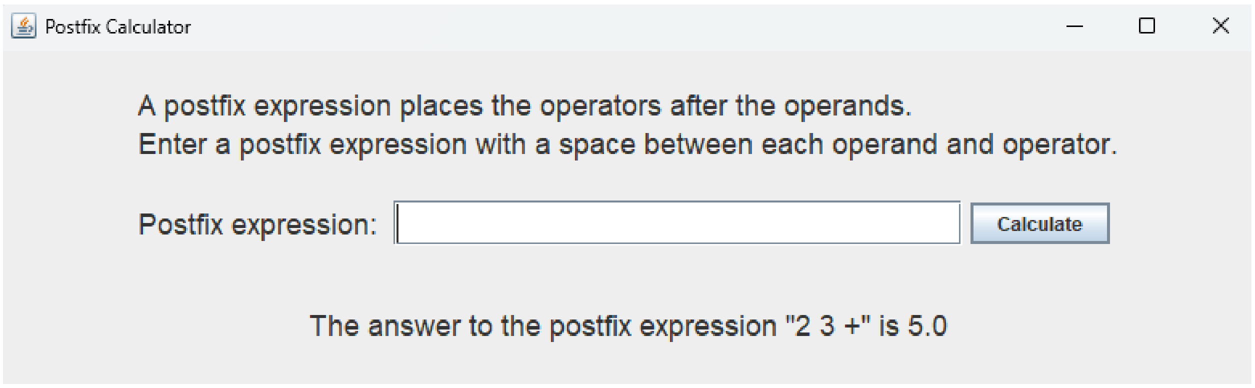
Here are sample runs of the program.



A postfix expression places the operators after the operands.
Enter a postfix expression with a space between each operand and operator.

Postfix expression: 2 3 +

The expression has been entered. Then the user presses the <`Enter`> key to get the result.

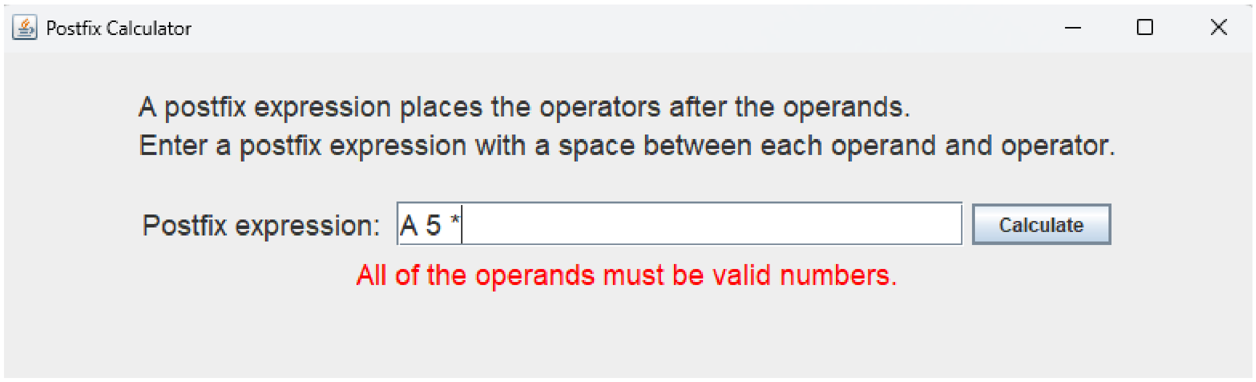

Postfix Calculator

A postfix expression places the operators after the operands.
Enter a postfix expression with a space between each operand and operator.

Postfix expression:

The answer to the postfix expression "2 3 +" is 5.0

Note that the expression and the answer are displayed in a label and that the result is of type `double`. The expression field has been cleared, and the application is waiting for another expression to be entered.

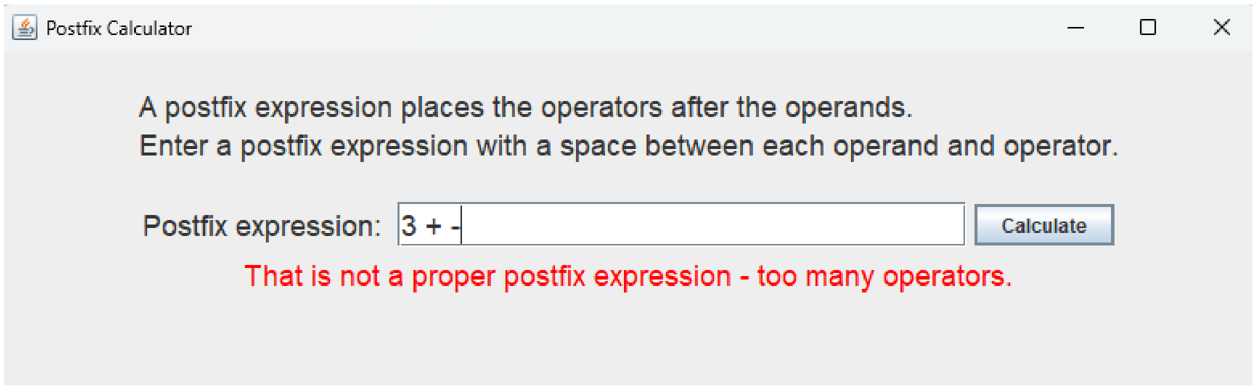

Postfix Calculator

A postfix expression places the operators after the operands.
Enter a postfix expression with a space between each operand and operator.

Postfix expression: A 5 *

All of the operands must be valid numbers.

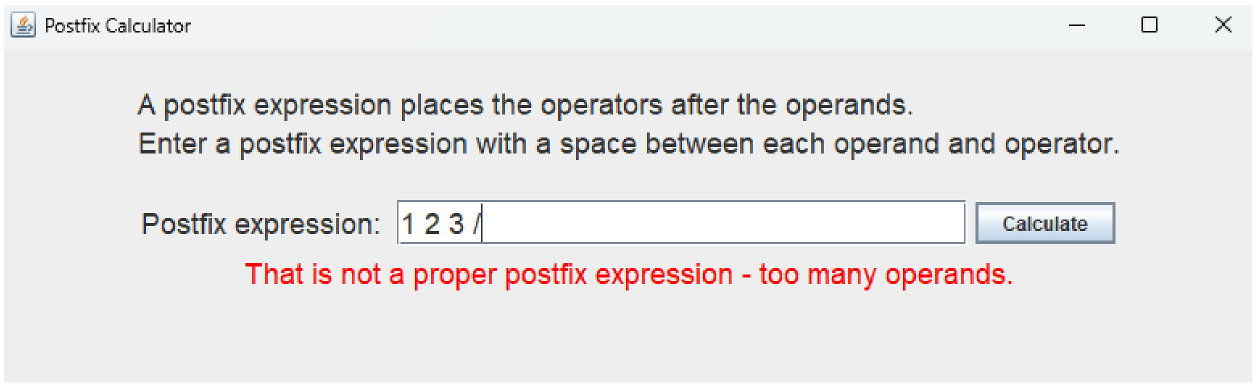
“A” is not a number.


Postfix Calculator

A postfix expression places the operators after the operands.
Enter a postfix expression with a space between each operand and operator.

Postfix expression: 3 + -

That is not a proper postfix expression - too many operators.



A postfix expression places the operators after the operands.
Enter a postfix expression with a space between each operand and operator.

Postfix expression: 1 2 3 /

That is not a proper postfix expression - too many operands.

Expectations

The program is expected to compile successfully, include identification comments at the top, and produce output that exemplifies good grammar and spelling. Failure to meet these standards may result in point deductions.