**Dijkstra Two-Stack Calculator Design**

Program structure

The program will be coded so that it follows a clean, readable structure, with modular design. It will consist of one main method that will handle the input and processes the expression using two stacks. One stack will be for the operands and one for the operators. The method will use helper methods: one to split the string into parts (called tokenisation), one to handle the logic of what to do which each of these parts and then one to handle the calculations. This is just to keep everything neat. The modular design will mean that everything is easy to understand, test and update

The Stack:

The stack will have the following operations

push(item): adds an item to the top of the stack (the operators or operands)

pop(): removes and returns the top item

peek(): returns the top item without removing it (at the end for the final result)

isEmpty(): checks if the stack is empty

isFull(): (if using an array) checks if the stack is full

Checking and Handling Tokens

Each character or number in the expression is a token. Tokens are brackets, numbers and operators (+ or \*). As the program loops through the expression, it must determine what the token is. If it is a number, it will be placed on the operand stack. If the token is an operator, the operator will be placed in the operator stack. If the token is a ), an operation will need to happen using the operator on top and the last two operands.

Flow of the Algorithm:

Once the string is split, the algorithm goes through it from left to right and processes each token:

Pseudocode:

for each token in the expression:

if token is "(":

do nothing

else if token is an operator:

push onto operator stack

else if token is a number:

push onto operand stack

else if token is ")":

pop operator from operator stack

pop two values from operand stack

apply operator to values

push result back onto operand stack

Splitting the String Into Tokens

To evaluate the expression correctly, I’ll need to split it up based on the spaces. For example, in this expression ( 1 + ( ( 2 + 3 ) \* ( 4 \* 5 ) ) ) each number, operator and bracket is separated by a space. I could use expression.split(" ") in Java, which will create an array of strings that I can iterate through, one at a time.

Edge Cases and Considerations

There are a few special cases that I need to take into consideration:

Mismatched parentheses: If they input extra ) or (, the program should ideally catch that.

Division by zero: The program should execute a check for division, and if dividing, it should check that the divisor isn't zero prior to executing.

Invalid tokens: The program should reject anything that is not a number, operator, or bracket, or ideally, still process it and display an error message.

Decimal management: The splitting and converting should be robust enough to handle a number like 12.5 without crashing.

Empty input: If the user enters nothing, the program should not try to evaluate it.

Scalability and future opportunities

In a future version, I could make the calculator support variables (ie x = 5), or even add more operators like ^ for powers, or % for modulus, or extend the functions to include things like sin() or sqrt(). I could also create a basic user interface so people don’t have to type expressions with brackets manually.