God please help me, what is wrong with my life
Pseudocode:

MakeVariablesList (Expression, Variables Array, Arrays Array){
Break expression apart according to denims and spaces —> Make sure each item/TOKEN is something that can be added to the arrays and ignores spaces
Create an empty string variable (est) to record current items/TOKENS
Iterate through each item/token in the expression
Find the token, set it to est
Find the token of the index and the length of the token (since it could be something like ‘a’ or ‘varA’ or ‘x’ or ‘A[3]’ etc.)
Check if token is the last character in expression —> add to variables (since last token can never be an opening array sign)
Check if token contains an opening bracket —> add to array
Otherwise —> add to variables (If the token’s not an array, it has to be a variable)
}

Evaluate (Expression, Variables Array, Arrays Array){
Create a stack for the numbers
Create a stack for the operators
Iterate through the Expression (let’s say VARIABLE A here){
Check if the character at index VAR A in Expression is a digit{ **(<— THIS IS TO ADD NUMBERS TO THE STACK FOR NUMBERS)**
Make a dummy variable (VAR P) and set it equals to VAR A.
//^^^You’re gonna have to substring the Expression to add onto the stack for numbers. The number is from A —> P
Iterate through the Expression using VAR P (as VAR P increases){
Break this iteration when VAR P is equals to the Expression’s length
Substring Expression using VAR A and VAR P (<— This should be a #)
Add the substring to the stack for numbers
Manipulate VAR A so that it doesn’t iterate through the same characters of Expression
}
Check if character at index VAR A in Expression is an array{
Use a variable (VAR K) to keep track of what comes before the opening bracket in an array. You need this to know which array you are going to access
Iterate backwards and find the array list’s name;
Iterate through Arrays array (say we use VAR D here){
Check whether or not the specific iteration is equal to the Array substring, checking if index D in ArrayList arrays is the same as the one we just found
Create dummy array to find the array values using VAR D

Use recursion to EVALUATE what's after the opening bracket; MAKE SURE YOU FIND THE VALUE FROM THE DUMMY ARRAY
If you're going to find something from an array:

StackForNumbers.push((float)dummyArray[(int)evaluate(expr.substring(AFTER OPENING BRACKET), vars, arrays)]);
Push/Add the newfound variable as an array onto numStack Stacks
Break out of this loop because you DO NOT want to repeat this as it would add Array indexes to the same position
}
Check for balance of brackets using integer balance around 0 (meaning set an int to 0; add or subtract it accordingly and do an equality check for the int at 0)
}
Check if character at index VAR A in Expression is a letter{
Make a replica of VAR A, yet again. This time we'll use VAR H.
Iterate through Expression using VAR H as long as the Expression at index VAR H is a letter{
Basically find out how long the letters are
Check if VAR H is the same size as Expression{
Iterate through ArrayList Variables to check whether or not there are variables
If there are variables, add the value of that variable onto the numStack Stack
Reduce the size of VAR A since you already checked for VAR H amount of variables
}
Check if Expression at VAR H is NOT an opening Array bracket{
Iterate through Variables array (use VAR C in this case)
Check if Variables at index VAR C is equal to the substring of Expression between VAR A and VAR H (<— Between VAR A and VAR H should be a letter variable)
Push the item in Variables at index VAR C
Reduce the size of VAR A since you are removing that amount of variables
Time to check for parentheses; Check for opening parentheses at index VAR A in Expression{
Since it's a parentheses, use recursion to send back the expression after the parentheses to evaluate (INCLUDE VARS + ARRAYS)
^^^ Should look like --> StackOfNumbers.push((float)evaluate(expr.substring(EVERYTHING AFTER PARENTHESES), vars, arrays));
Check for balance of brackets using integer balance around 0 (meaning set an int to 0; add or subtract it accordingly and do an equality check for the int at 0)
^^^ You run almost the exact thing to find the balance of opening and closing parentheses

Check if the character at index VAR A is addition, subtraction, multiplication, and division --> transfer it to the stack for operators

Check for ending parentheses/brackets or if nearing the end of expression{

Check if there's operator stacks, otherwise there's a big issue{

Create new stacks for operators

Create new stacks for numbers

Iterate through numStack and basically reverse the order into the new stack for numbers

Iterate through operatorStack and reverse the order into the new stack for operators

Iterate through the new stack of operators

Look forward to the operator and check each individual operator{

ADDITION

Pop the stack off the new opStack and push it back on to the old opStack

Do the same exact thing for the new numStack and the old numStack

check if there are no more new opStacks and if there's still more new numStack items

If there are more new numStack items, just keep adding them to the old numStack;

SUBTRACTION

Same process as add

MULTIPLICATION/DIVISION

Create a float (say value) and set it equal to the first pop of the newNumStack; This is to find the dividend and the multiplicand (as you need the ones first, then tens, and so on)

To make life easier, make a while loop for multiply and divide here. For every iteration of the OG while loop, this will check for >1 multiples or divides.

If it's a multiply operator, do value *= the float you made before

If it's a divide operator, do value /= the float

If the new operatorStack is empty, break

REMEMBER --> Push the value variable onto the new numStack

}

If old numStack isn't empty, push the rest of the items onto the new numStack

Same thing for old operatorStack

Create another float variable equals to the first item of the new numStack

Iterate through the new operatorStack until it's empty.{ (<— Check if the operator is + or -)

Make a string variable and set it equal to the pop off the new operatorStack

If the str variable is add, do value = value + the pop off the

```
new numStack
                                If the str variable is subtract, do the opposite thing of add.
                            }
                            Return the value variable once this loop is done
                        }
                        Else —> return the top of numStack;
                    }
                }
}
```