# Overview

In this lab, you will create a class called Postfix.java. A postfix equation requires that its operators come after the corresponding operands (ex. 5 4 + → 9). A valid operator includes +, -, /, \* and a valid operand is any integer.

# Objectives

* Practice creating a class
* Practice with Java Stacks
* Practice with Java Postfix equations
* Apply test cases to your program

# Steps

1. Create a folder on your local machine for your Java program, you can name it whatever you like
2. Start Visual Studio Code (VS Code)
3. In VS Code, Open that newly created folder.
4. Now it’s time to start coding:
   1. Create a class called Postfix
   2. Create the following in Postfix.java:
      1. A **static** method called **checkValidOperator** (returns a char) that has a string as a parameter. Use a switch statement ([see example](#tf1sxis3ai3r)) to see if the given string is a valid operator. If it is, return the string as a character. If it’s not return a ? as a character

Example (results in red)

| System.out.println( checkValidOperator(“+”));  System.out.println( checkValidOperator(“!”));  System.out.println( checkValidOperator(“--”)); |
| --- |

+

?

?

* + 1. A **static** method **execute** which gets the Stack of Integers and a char representing a valid operator as parameters. This method assumes that there are at least2 elements on the stack and that the char is a valid operator char.
       1. pop 2 elements off stack: first is right operand, second is left operand
       2. using a switch statement, do the operation, assign the result to a variable
       3. push the result onto the stack
       4. return the result

Example (result in red)

| Stack<Integer> s = new Stack<>();  s.push(2);  s.push(2);  char operator = ‘+’;  execute(s, operator);  System.out.println(s); |
| --- |

[4]

* + 1. A **static** method called **evaluateExpression** that takes a String **expr** as a parameter and returns a **Number**.

1. Put these variables into the method (to use as error messages)
   1. String tooFewOperands = "Too few operands";
   2. String unknownOperator = "Unknown operator: ";
   3. String tooManyOperands = "Too many operands.";
2. Create an empty stack of Integers called **stack**. Create a String variable named **errMsg** and set to null. Initialize a scanner using **expr** as a parameter to its constructor, which will allow you to read tokens from the expression.
3. Write a loop that will run as long as the scanner has input (hasNext) and error message is null.
   1. If the scanner contains an integer next (hasNextInt), read the integer from the scanner into a variable. Print output to indicate that an operand was read, such as "Operand read: 3". Then push that integer value onto the stack.
   2. If the scanner does not contain an integer next, read the next token as a String. If a valid operator was read, print output to indicate that an operator was read, such as "Operator read: -”.
      1. If there are at least 2 operands on the stack, call **execute(stack, operator)**
      2. If there are fewer than 2 operands on the stack, set **errMsg** equal to **tooFewOperands**
   3. If an invalid/unknown operator was read set **errMsg** equal to **unknownOperator** plus the invalid operator that was given
   4. Print "------ Stack state -----"
   5. Print the stack
4. If **errMsg** was set above it will no longer be null. Print out "Failed evaluation of |" + **expr** + "|\n" + **errMsg**). Return null.
5. If there are too many operands (more than one value remains on the stack) print out "Failed evaluation of |" + **expr** + "|\n" + **tooManyOperands** + **stack to string**). Return null.
6. If both of those are not true, return the value at the top of the stack.
   * 1. A main method (also static.)
7. Ask the user to enter a postfix expression, declare an input string, and initialize a scanner.
8. If the scanner has a next line, save the value to the input string and call **evaluateExpression** on that input string. Save the result from **evaluateExpression** into a variable.
   1. If the result of **evaluateExpression** was not null, print "Expression: " + input string + " --> " + result of evaluateExpression(input string)

# 

# Example Output

Red text is user input

| Enter a postfix expression  1 2 +  Operand read: 1  ------ Stack state -----  [1]  Operand read: 2  ------ Stack state -----  [1, 2]  Operator read: +  ------ Stack state -----  [3]  Expression: 1 2 + --> 3 |
| --- |

| Enter a postfix expression  1 2 + 3 \* 1 + 2 /  Operand read: 1  ------ Stack state -----  [1]  Operand read: 2  ------ Stack state -----  [1, 2]  Operator read: +  ------ Stack state -----  [3]  Operand read: 3  ------ Stack state -----  [3, 3]  Operator read: \*  ------ Stack state -----  [9]  Operand read: 1  ------ Stack state -----  [9, 1]  Operator read: +  ------ Stack state -----  [10]  Operand read: 2  ------ Stack state -----  [10, 2]  Operator read: /  ------ Stack state -----  [5]  Expression: 1 2 + 3 \* 1 + 2 / --> 5 |
| --- |

### Example switch statement:

int user = scan.nextInt();

switch(user) {

case 10:

System.out.println("You are user 10.");

break;

case 20:

System.out.println("You are user 30.");

break;

case 30:

System.out.println("You are user 20.");

break;

default:

System.out.println("You are an unknown user with id " + user + ".");

break;

}