**Problem Statement**

The main problem I anticipate in this assignment is managing the different variables on the stack. It will be difficult ensure that the variables are prioritized so that the correct answer can be obtained. I also anticipate that discarding brackets correctly while maintaining the integrity of that stack will be difficult.

Before beginning my work on this project I researched how to correctly use and implement stacks in java , my main source were the notes on blackboard but occasionally I had to consult stack overflow or other online forums.

I predict I will have to use switch statements to assign operative preference assigning a different value to a variable based on the operator. The higher the number the greater the priority. Throughout the assignment it will be important to separate any repeated code into its own function, not only for readability but also to avoid ant excess code repetition.

**Analysis and design Notes**

To plan correctly I ran some basic tests in an java Ide on a character and string object. This helped me to gauge what functions could be applied to each, and what functions couldn’t. First I learned that small apostrophes ( ‘ ) must be used when declaring and comparing characters. Using normal apostophes (“) causes java to assume that a string is being used. .equals methods must also be used differently on characters and strings. It was important to make these distinctions before I began coding so unnecessary time was not wasted solving errors which could have been foreseen with some simple experimentation and planning.

I begun coding this assignment by correctly declaring a stack. I brushed up on the different functions of the stack object and ensured that I thoroughly understood how and where they should be used. Secondly I declared a string variable which is designed to hold the variables entered by the user once they have been sorted correctly.

The first major hurdle was designing the checks for correct operator/operand popping/pushing. I planned how I was going to follow the assignment brief by writhing out pseudocode by hand. I knew I would have to check the precedence of the operators multiple time and so early on in my implementation I decided to divide this into a separate function along with the validity and operator checker. Failing to divide this code into separate functions would have meant huge amounts of repetition and would have significantly affected the readability of the code. In the precedence function I decided to use ‘if’ statements because there were only a small amount of checks in comparison to the validityChecker which had to check for a wide range of characters. Invalid characters are ignored and the function recalls main if an invalid character is entered

**infixPostFix function**

1. First step was checkinf for a bracket if the character entered was a bracket then it was pushed.
2. Second check appends the top character in the stack to a string if the user enters a bracket,if the stop of the stack is not a bracket and if the stack is not empty.
3. The third check pops the bracket if the top of the stack is a left bracket
4. The fourth check ensures that the value is not an operand and if its not then the value is appended to the end of the string if the stack is empty.
5. The final check appends the operator to the string but does it using the precedence functions, so the operators are in the correct order
6. If the value has passed all these checks then it is pushed to the stack.

**evaluatePostFix function**

1. First like the last function a stack is declared. A for loop is used and the expression is divided into characters based on its length.
2. Value is pushed to the stack if its not an operator. If it is an operator then it is pushed from the stack and stored in an int value for use in calculations. The result variable holds the answer
3. The switch statement accounts for operators and ensures correct calculation. Although it may not be 100% necessary once all calculations have taken place the answer is pushed to the stack.
4. The returned value is the top value on the stack which will always be the result of the calculations.

**Main function**

1. I believe this function to be mostly self-explanatory. As all main functions it acts a caller for other functions
2. It uses a simple scanner object to take in user input and prints the result of the functions to the terminal.

Code for Assignment

import *java*.*util*.*Scanner*;

import *java*.*util*.*Stack*;

*public* *class* InfixToPostfix {

*private* *static* boolean *operatorCheckl*(char c) { //*checks if value entered is or is not an operator*

*return* c == '+' || c == '-' || c == '\*' || c == '/' || c == '^';

  }

*private* *static* boolean *validityChecker*(char c){//*checks aginast all valid characters to ensure user has entered only valid characters*

*return* c == '+' || c == '-' || c == '\*' || c == '/' || c == '^' ||  c == '0' ||  c == '1' ||  c == '2' ||  c == '3' ||  c == '4' ||  c == '5' ||  c == '6' ||  c == '7' ||  c == '8' ||  c == '9' || c =='(' || c==')';

  }

*private* *static* int *precedence*(char c) {

    //*returns a different value based on the precdence of the operator entered.*

*if*(c == '+'){

*return* 1;

    }

*if*(c == '-'){

*return* 1;

      }

*if*(c == '\*'){

*return* 2;

      }

*if*(c == '/'){

*return* 2;

        }

*if*(c == '^'){

*return* 3;

            }

*return* -1;

  }

*public* *static* String *infixPostfix*(String expression) {

    Stack<Character> stackOne = *new* Stack<>();//*declares stack*

    String postFixString =""; //*creates string which will hold postfix value*

*for* (int i = 0; i < expression.*length*(); i++) {

      //*the operator is appended to a post fix string if the necessarry checks are passed.*

      char c = expression.*charAt*(i);

      //*these if statments are explained in greater detail in the pdf*

*if* (c == '(') { //*checks if bracket*

        stackOne.*push*(c);

      } *else* *if* (c == ')') {

*while* (!stackOne.*isEmpty*() && stackOne.*peek*() != '(') {

          postFixString = postFixString + stackOne.*pop*();

        }

*if* (!stackOne.*isEmpty*() && stackOne.*peek*() == '(') {

          stackOne.*pop*();

        }

      } *else* *if* (!*operatorCheckl*(c)) {

        postFixString = postFixString + c;

      } *else* {

*while* (!stackOne.*isEmpty*() && *precedence*(stackOne.*peek*()) >= *precedence*(c)) {//*this if statment also checks that the operators preference is correct.*

          postFixString = postFixString + stackOne.*pop*();

        }

        stackOne.*push*(c);//*pushes character to stack if all checks are passed*

      }

*if*(!*validityChecker*(c)){

        System.*out*.*print*("Your input contains an invalid character. Please make sure all of your input is a number or operator\n");

        String temp[] = {""};

*main*(temp);//*function repeats itself if invalid character is entered*

      }

    }

*while* (!stackOne.*isEmpty*()) {

      //*postfix.append(stack.pop());*

      postFixString = postFixString +stackOne.*pop*();

    }

*return* postFixString;

  }

*public* *static* int *evaluatePostfix*(String postfix) {

    Stack<Integer> stackTwo = *new* Stack<>();//*stack declaration*

*for* (int i = 0; i < postfix.*length*(); i++) {

      char c = postfix.*charAt*(i);

*if* (!*operatorCheckl*(c)) {//*if not an operator than push to stack*

        stackTwo.*push*(c - '0');

      } *else* {

        int number2 = stackTwo.*pop*();//*gets correctly ordered numbers from stack*

        int number1 = stackTwo.*pop*();

        int resultHolder = 0;

    //*following switch statment puts the numbers through the various diffferent calculations.*

*switch* (c) {

*case* '+'*:*

            resultHolder = number1 + number2;

*break*;

*case* '-'*:*

            resultHolder = number1 - number2;

*break*;

*case* '\*'*:*

            resultHolder = number1 \* number2;

*break*;

*case* '/'*:*

            resultHolder = number1 / number2;

*break*;

*case* '^'*:*

            resultHolder = (int) Math.*pow*(number1, number2);

*break*;

        }

        stackTwo.*push*(resultHolder);//*psushes result to the stack*

      }

    }

*return* stackTwo.*pop*();//*returns answer*

  }

*public* *static* void *main*(String args[]) {

    Scanner sc = *new* *Scanner*(System.*in*);

    System.*out*.*println*("Please enter you infix expression. The valid characters are +,-,\*,/,^");

    String expression = sc.*nextLine*();

    String postfix = *infixPostfix*(expression);

    System.*out*.*println*("Infix expression: " + expression);

    System.*out*.*println*("Postfix expression: " + postfix);

    int result = *evaluatePostfix*(postfix);

    System.*out*.*println*("Result: " + result);

  }

}

Testing for assignment

Simpleinfix expression and postfix output

Text

Description automatically generated

Longer more advanced infix input and postfix output

Text

Description automatically generated

Long infix expression with brackets and multiple operands

Text

Description automatically generated

long infix input with bracket and division

Text

Description automatically generated

Invalid character tests

Text

Description automatically generated

Text

Description automatically generated