Arithmetic Expression Evaluation

Arithmetic Expression Evaluation
Algorithm for Arithmetic Expression evaluation

1. Convert the expression in Reverse Polish notation(post-fix notation).
2. Push the operands into the stack in the order they appear.
3. When any operator encounters then pop two topmost operands for executing the operation.
4. After execution push the result obtained into the stack.
5. After the complete execution of expression, the final result remains on the top of the stack.

infix expression → (9 −2) * (4 +2) → input → output → 12

Step 1 → convert the infix expression in postfix expression.
Step 2 → if the incoming character is operand then push into the stack.
Step 3 → if the incoming character is operator then the pop two element from the stack. and execute the operation.
Step 4 → After execution of the operator push the result into the stack
Step 5 → After the complete execution of the expression. the final result available in the stack.
Step 6 → then after print the stack.

infix → (4 -2) * (4 +2)
postfix → 42 - * 42 + ⇒ 92 - 92 + *

① create Stack, datatype is Integer

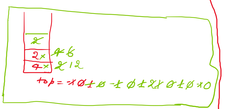
```
Stack<Integer> stack = new Stack<>();
```

② create for loop and traverse the postfix expression.

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

③ check if incoming character is digit or not if digit then push into the stack. (operand) Postfix exp → 42 - 42 + *

```
for( int i=0; i<postfix.length(); i++){
    char ch = postfix.charAt(i);
  if( Character.isDigit(ch)){
      stack.push(ch);
  }
}
```



④ if incoming character is operator then pop two element from the stack and evaluate the operator.

```
for( int i=0; i< postfix.length(); i++){
    char ch = postfix.charAt(i);
  if(Character.isDigit(ch)){
      stack.push(ch);
  }
  else if(!stack.isEmpty()){
  Integer v1 = stack.pop();
  Integer v2 = stack.pop();
  int result = 0;
  switch(ch) {
   Case '+':
        result = v2+v1;
      stack.push(result);
      break;
   Case '-':
        result = v2 -v1;
      stack.push(result);
      break;
   Case '*':
        result = v2 *v1;
      stack.push(result);
      break;
   Case '/':
        result = v2/v1;
      stack.push(result);
      break;
  }// closing switch
 }// closing else block
}// closing for loop
// finally pop the stack then return the popped element.
return stack.pop();
} // closing method
```

checking if incoming character is operand or not

if not operand then operator will come

Popping two element from the stack.

doing operator evaluation → after evaluation result is pushing into the stack

Complete program for Arithmetic expression evaluation.

```
public class ArithmeticExpressionEvaluation {

    public static int getPeriority(char ch) {
        switch (ch) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        }
        return -1;
    }

    public static String infixToPostfixExpression(String infixExp) {
        String postfixExp = "";
        Stack<Character> stack = new Stack<>();
        int length = infixExp.length();
        for (int i = 0; i < length; i++) {
            char ch = infixExp.charAt(i);
            if (Character.isLetterOrDigit(ch)) {
                postfixExp += ch;
                continue;
            } else if (ch == '(') {
                stack.push(ch);
            } else if (ch == ')') {
                while (!stack.isEmpty() && stack.peek() != '(') {
                    postfixExp += stack.peek();
                    stack.pop();
                }
                stack.pop();
            } else {
                while (!stack.isEmpty() && getPeriority(ch) <= getPeriority(stack.peek())) {
                    postfixExp += stack.peek();
                    stack.pop();
                }
                stack.push(ch);
            }
        }
        while (!stack.isEmpty()) {
            postfixExp += stack.peek();
            stack.pop();
        }
        return postfixExp;
    }

    public static int calculateArithmeticOperationFromInfixExpression(String postfixExpression) {
        int length = postfixExpression.length();
        Stack<Integer> stack = new Stack<>();
        for (int i = 0; i < length; i++) {
            char ch = postfixExpression.charAt(i);
            if (Character.isDigit(ch)) {
                stack.push(Character.getNumericValue(ch));
                continue;
            } else if(!stack.isEmpty()) {
                Integer value1 = stack.pop();
                Integer value2 = stack.pop();
                int result = 0;
                switch (ch) {
                case '+':
                    result = value2 + value1;
                    stack.push(result);
                    break;
                case '-':
                    result = value2 - value1;
                    stack.push(result);
                    break;
                case '*':
                    result = value2 * value1;
                    stack.push(result);
                    break;
                case '/':
                    result = value2 / value1;
                    stack.push(result);
                    break;
                }
            }
        }
        return stack.pop();
    }

    public static void main(String[] args) {
        String infixExpression = "(4-2)*(8-6)";
        System.out.println("Infix Expression    : " + infixExpression);
        String postfixExpression = infixToPostfixExpression(infixExpression);
        System.out.println("Postfix Expression  : " + infixToPostfixExpression(postfixExpression));
        int finalResult = calculateArithmeticOperationFromInfixExpression(postfixExpression);
        System.out.println("Arithmetic Operation result : " + finalResult);
    }

}
```

checking priority for postfix expression

converting from infix expression to postfix expression

this method will take the input as a postfix expression.

this method will evaluate postfix expression then it will return the output.

this is main method which is responsible to call the another method.