```java
import java.util.Stack;
import java.lang.*;
public class dz
{
//do simple caculation with single operator in two stacks
public static int doOp(Stack<Integer> valStk, Stack<Character> opStk)
    {
//pop elements in both stacks
    int m = valStk.pop();
    int n = valStk.pop();
    char oper = opStk.pop();
//determine which operator
    switch (oper)
    {
    case '+':
        return m+n;
    case '-':
        return m-n;
    case '*':
        return m*n;
    case '/':
//denominator couldn't be 0
        if (n == 0){System.out.println(x: "not able to be divided by 0");return 0;
        }
        return m/n;
    }
    //no found exist operator
    return 0;
}
public static int repeatOps(){
//opStk is not empty
    while(!opStk.isEmpty() && prec(c)<=prec(opStk.peek()))
    {
//output is done by doOp function
        int output = doOp(valStk, opStk);
        valStk.push(output);
    }
    opStk.push(c);
}
public static int EvalExp(String expression)
{
```

```java
    //Operand stack value
    Stack<Integer> valStk = new Stack<>();
    //Operator stack character
    Stack<Character> opStk = new Stack<>();
    for(int i=0; i<expression.length();i++)
    {
        char c = expression.charAt(i);
        if(Character.isDigit(c))    //check if it is a integer
        {
            int num = 0;
            while (Character.isDigit(c))
            {
                num = num*10 + (c-'0');
                i++;
                if(i < expression.length())
                {
                c = expression.charAt(i);
                }
                else
                {
                break;
                }
                }
            i--;
            valStk.push(num);
        }
        //( means beginning
        else if(c=='(')
        {
            opStk.push(c);
        }
        // ) means end
        else if(c==')')
        {
            while(opStk.peek()!='(')
            {
                int output = doOp(valStk, opStk);
                valStk.push(output);   //push result back to stack
            }
            opStk.pop();
        }
```

```
            }

            // if c is an operator
            else if(IsOper(c))
            {
                repeatOps();
            }
        }
    //while there's another token
        while(!opStk.isEmpty())
        {
            int output = doOp(valStk, opStk);
            valStk.push(output);   //get final result to stack
        }
        return valStk.pop();
    }
    //determine character c's predence high or low
    static int prec(char c)
    {
        switch (c)
        {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        }
            return -1;
    }
    //determine if character c is operator or not
    public static boolean IsOper(char c)
    {
        if(c=='+'||c=='-'||c=='/'||c=='*'||c=='^'){
            return true;
        }
        return false;

    }
```