```java
import java.util.Scanner;
import java.util.Stack;

public class MathCalculator {

    public static void main(String[] args) {
        Scanner inputScanner = new Scanner(System.in);
        System.out.print("Enter a mathematical expression : ");
        String expressionInput = inputScanner.nextLine();

        try {
            double calculationResult = computeExpression(expressionInput);
            System.out.println("result is: " + calculationResult);
        } catch (Exception ex) {
            System.out.println("Error: " + ex.getMessage());
        }

        inputScanner.close();
    }

    public static double computeExpression(String expression) throws Exception {
        // Remove spaces for easier processing
        expression = expression.replaceAll(" ", "");

        Stack<Double> valueStack = new Stack<>();
        Stack<Character> operatorStack = new Stack<>();

        int index = 0;
        while (index < expression.length()) {
            char character = expression.charAt(index);

            if (Character.isDigit(character)) {
```

```
33                  StringBuilder numberBuilder = new
   StringBuilder();
34                  while (index < expression.length
   () && (Character.isDigit(expression.charAt(index
   )) || expression.charAt(index) == '.')) {
35                      numberBuilder.append(expression.
   charAt(index++));
36                  }
37                  valueStack.push(Double.parseDouble(
   numberBuilder.toString()));
38                  index--; // Step back for the outer
   loop
39              }
40              // Handle opening parenthesis
41              else if (character == '(') {
42                  operatorStack.push(character);
43              }
44              // Handle closing parenthesis
45              else if (character == ')') {
46                  while (operatorStack.peek() != '(') {
47                      valueStack.push(performOperation(
   operatorStack.pop(), valueStack.pop(), valueStack.pop
   ()));
48                  }
49                  operatorStack.pop(); // Remove '('
   from stack
50              }
51              // Handle operators
52              else if (isOperator(character)) {
53                  while (!operatorStack.isEmpty() &&
   precedenceLevel(operatorStack.peek()) >=
   precedenceLevel(character)) {
54                      valueStack.push(performOperation(
   operatorStack.pop(), valueStack.pop(), valueStack.pop
   ()));
55                  }
56                  operatorStack.push(character);
57              }
58              index++;
59          }
60
```

```
61          // Complete remaining operations
62          while (!operatorStack.isEmpty()) {
63              valueStack.push(performOperation(
    operatorStack.pop(), valueStack.pop(), valueStack.
    pop()));
64          }
65
66          return valueStack.pop();
67      }
68
69      // Function to apply an operator to two numbers
70      public static double performOperation(char
    operator, double secondOperand, double firstOperand
    ) throws Exception {
71          switch (operator) {
72              case '+':
73                  return firstOperand + secondOperand;
74              case '-':
75                  return firstOperand - secondOperand;
76              case '*':
77                  return firstOperand * secondOperand;
78              case '/':
79                  if (secondOperand == 0) throw new
    Exception("Division by zero is not allowed.");
80                  return firstOperand / secondOperand;
81              case '%':
82                  return firstOperand % secondOperand;
83              default:
84                  throw new Exception("Unsupported
    operator: " + operator);
85          }
86      }
87
88      // Function to check operator precedence
89      public static int precedenceLevel(char operator
    ) {
90          switch (operator) {
91              case '+':
92              case '-':
93                  return 1;
94              case '*':
```

```
 95                 case '/':
 96                 case '%':
 97                     return 2;
 98                 default:
 99                     return -1;
100             }
101         }
102 
103         // Function to check if a character is an
    operator
104         public static boolean isOperator(char character
    ) {
105             return character == '+' || character == '-'
     || character == '*' || character == '/' ||
    character == '%';
106         }
107 }
108 
```