

```

1 package com.example.calculator;
2
3 /*
4     A Java program to evaluate a
5     given expression where tokens
6     are separated by space.
7     @Credits - GeeksOfGeeks
8     Used by Rajkumar B L
9 */
10 import java.util.Stack;
11
12 public class EvaluateString
13 {
14     public static int evaluate(String expression)
15     {
16         char[] tokens = expression.toCharArray();
17
18         // Stack for numbers: 'values'
19         Stack<Integer> values = new
20             Stack<Integer>();
21
22         // Stack for Operators: 'ops'
23         Stack<Character> ops = new
24             Stack<Character>();
25
26         for (int i = 0; i < tokens.length; i++)
27         {
28
29             // Current token is a
30             // whitespace, skip it
31             if (tokens[i] == ' ')
32                 continue;
33
34             // Current token is a number,
35             // push it to stack for numbers
36             if (tokens[i] >= '0' &&
37                 tokens[i] <= '9')
38             {
39                 StringBuffer sbuf = new
40                     StringBuffer();
41
42                 // There may be more than one
43                 // digits in number
44                 while (i < tokens.length &&
45                     tokens[i] >= '0' &&
46                     tokens[i] <= '9')

```

```

47         sbuf.append(tokens[i++]);
48         values.push(Integer.parseInt(sbuf.
49             toString()));
50
51         // right now the i points to
52         // the character next to the digit,
53         // since the for loop also increases
54         // the i, we would skip one
55         // token position; we need to
56         // decrease the value of i by 1 to
57         // correct the offset.
58         i--;
59     }
60
61     // Current token is an opening brace,
62     // push it to 'ops'
63     else if (tokens[i] == '(')
64         ops.push(tokens[i]);
65
66         // Closing brace encountered,
67         // solve entire brace
68     else if (tokens[i] == ')')
69     {
70         while (ops.peek() != '(')
71             values.push(applyOp(ops.pop(),
72                 values.pop(),
73                 values.pop()));
74         ops.pop();
75     }
76
77     // Current token is an operator.
78     else if (tokens[i] == '+' ||
79         tokens[i] == '-' ||
80         tokens[i] == '*' ||
81         tokens[i] == '/')
82     {
83         // While top of 'ops' has same
84         // or greater precedence to current
85         // token, which is an operator.
86         // Apply operator on top of 'ops'
87         // to top two elements in values stack
88         while (!ops.empty() &&
89             hasPrecedence(tokens[i],
90                 ops.peek()))
91             values.push(applyOp(ops.pop(),
92                 values.pop(),

```

```

93             values.pop()));
94
95             // Push current token to 'ops'.
96             ops.push(tokens[i]);
97         }
98     }
99
100    // Entire expression has been
101    // parsed at this point, apply remaining
102    // ops to remaining values
103    while (!ops.empty())
104        values.push(applyOp(ops.pop(),
105                            values.pop(),
106                            values.pop()));
107
108    // Top of 'values' contains
109    // result, return it
110    return values.pop();
111 }
112
113 // Returns true if 'op2' has higher
114 // or same precedence as 'op1',
115 // otherwise returns false.
116 public static boolean hasPrecedence(
117     char op1, char op2)
118 {
119     if (op2 == '(' || op2 == ')')
120         return false;
121     if ((op1 == '*' || op1 == '/') &&
122         (op2 == '+' || op2 == '-'))
123         return false;
124     else
125         return true;
126 }
127
128 // A utility method to apply an
129 // operator 'op' on operands 'a'
130 // and 'b'. Return the result.
131 public static int applyOp(char op,
132                           int b, int a)
133 {
134     switch (op)
135     {
136         case '+':
137             return a + b;
138         case '-':

```

```
139         return a - b;
140     case '*':
141         return a * b;
142     case '/':
143         if (b == 0)
144             throw new
145                 UnsupportedOperationException(
146                     "Cannot divide by zero");
147         return a / b;
148     }
149     return 0;
150 }
151 }
```