```java
package com.example.calculator;

/*
   A Java program to evaluate a
   given expression where tokens
   are separated by space.
   @Credits - GeeksOfGeeks
   Used by Rajkumar B L
*/
import java.util.Stack;

public class EvaluateString
{
    public static int evaluate(String expression)
    {
        char[] tokens = expression.toCharArray();

        // Stack for numbers: 'values'
        Stack<Integer> values = new
                Stack<Integer>();

        // Stack for Operators: 'ops'
        Stack<Character> ops = new
                Stack<Character>();

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

            // Current token is a
            // whitespace, skip it
            if (tokens[i] == ' ')
                continue;

            // Current token is a number,
            // push it to stack for numbers
            if (tokens[i] >= '0' &&
                    tokens[i] <= '9')
            {
                StringBuffer sbuf = new
                        StringBuffer();

                // There may be more than one
                // digits in number
                while (i < tokens.length &&
                        tokens[i] >= '0' &&
                        tokens[i] <= '9')
                    sbuf.append(tokens[i++]);
                values.push(Integer.parseInt(sbuf.
                        toString()));

                // right now the i points to
                // the character next to the digit,
                // since the for loop also increases
                // the i, we would skip one
                //  token position; we need to
                // decrease the value of i by 1 to
                // correct the offset.
                i--;
            }

            // Current token is an opening brace,
            // push it to 'ops'
            else if (tokens[i] == '(')
                ops.push(tokens[i]);

                // Closing brace encountered,
                // solve entire brace
            else if (tokens[i] == ')')
            {
                while (ops.peek() != '(')
                    values.push(applyOp(ops.pop(),
                            values.pop(),
                            values.pop()));
                ops.pop();
            }

            // Current token is an operator.
            else if (tokens[i] == '+' ||
                    tokens[i] == '-' ||
                    tokens[i] == '*' ||
                    tokens[i] == '/')
            {
                // While top of 'ops' has same
                // or greater precedence to current
                // token, which is an operator.
                // Apply operator on top of 'ops'
                // to top two elements in values stack
                while (!ops.empty() &&
                        hasPrecedence(tokens[i],
                                ops.peek()))
                    values.push(applyOp(ops.pop(),
                            values.pop(),
                            values.pop()));

                // Push current token to 'ops'.
                ops.push(tokens[i]);
            }
        }

        // Entire expression has been
        // parsed at this point, apply remaining
        // ops to remaining values
        while (!ops.empty())
            values.push(applyOp(ops.pop(),
                    values.pop(),
                    values.pop()));

        // Top of 'values' contains
        // result, return it
        return values.pop();
    }

    // Returns true if 'op2' has higher
    // or same precedence as 'op1',
    // otherwise returns false.
    public static boolean hasPrecedence(
            char op1, char op2)
    {
        if (op2 == '(' || op2 == ')')
            return false;
        if ((op1 == '*' || op1 == '/') &&
                (op2 == '+' || op2 == '-'))
            return false;
        else
            return true;
    }

    // A utility method to apply an
    // operator 'op' on operands 'a'
    // and 'b'. Return the result.
    public static int applyOp(char op,
                              int b, int a)
    {
        switch (op)
        {
            case '+':
                return a + b;
            case '-':
                return a - b;
            case '*':
                return a * b;
            case '/':
                if (b == 0)
                    throw new
                            UnsupportedOperationException(
                            "Cannot divide by zero");
                return a / b;
        }
        return 0;
    }
}
```