```java
import components.simplereader.SimpleReader;

/**
 * This program calculates the value of an expression consisting of numbers,
 * arithmetic operators, and parentheses.
 *
 * @author Put your name here
 *
 */
public final class ExpressionEvaluator {

    /**
     * Base used in number representation.
     */
    private static final int RADIX = 10;

    private static final char[] terms = { '0', '1', '2', '3', '4', '5', '6',
            '7', '8', '9', '*', '/', '(', ')' };

    /**
     * Private constructor so this utility class cannot be instantiated.
     */
    private ExpressionEvaluator() {
    }

    /**
     * Evaluates a digit and returns its value.
     *
     * @param source
     *            the {@code StringBuilder} that starts with a digit
     * @return value of the digit
     * @updates source
     * @requires 1 < |source| and [the first character of source is a digit]
     * @ensures <pre>
     * valueOfDigit = [value of the digit at the start of #source]  and
     * #source = [digit string at start of #source] * source
     * </pre>
     */
    private static int valueOfDigit(StringBuilder source) {
        assert source != null : "Violation of: source is not null";

        return Integer.valueOf(source.charAt(0));
    }

    /**
     * Evaluates a digit sequence and returns its value.
     *
     * @param source
     *            the {@code StringBuilder} that starts with a digit-seq string
     * @return value of the digit sequence
     * @updates source
     * @requires <pre>
     * [a digit-seq string is a proper prefix of source, which
     * contains a character that is not a digit]
     * </pre>
     * @ensures <pre>
     * valueOfDigitSeq =
```

```
61      *   [value of longest digit-seq string at start of #source]  and
62      * #source = [longest digit-seq string at start of #source] * source
63      * </pre>
64      */
65     private static int valueOfDigitSeq(StringBuilder source) {
66         assert source != null : "Violation of: source is not null";
67
68         int idx = 0;
69         int number = 0;
70         int value = 0;
71         StringBuilder expr;
72
73         while (idx < source.length()) {
74             Character next = source.charAt(idx);
75
76             if (Character.isDigit(next)) {
77                 value = valueOfDigit(source);
78
79             } else if (next == '(' || next == ')') {
80                 expr = new StringBuilder(number);
81                 value = valueOfExpr(expr);
82                 number = 0;
83             }
84
85             idx++;
86         }
87
88         return value;
89     }
90
91     /**
92      * Evaluates a factor and returns its value.
93      *
94      * @param source
95      *            the {@code StringBuilder} that starts with a factor string
96      * @return value of the factor
97      * @updates source
98      * @requires <pre>
99      * [a factor string is a proper prefix of source, and the longest
100     * such, s, concatenated with the character following s, is not a prefix
101     * of any factor string]
102     * </pre>
103     * @ensures <pre>
104     * valueOfFactor =
105     *   [value of longest factor string at start of #source]  and
106     * #source = [longest factor string at start of #source] * source
107     * </pre>
108     */
109    private static int valueOfFactor(StringBuilder source) {
110        assert source != null : "Violation of: source is not null";
111
112        int idx = 0;
113        int number = 0;
114        int value = 0;
115        StringBuilder expr;
116
117        while (idx < source.length()) {
```

```
118             Character next = source.charAt(idx);
119
120             if (Character.isDigit(next)) {
121                 value = valueOfDigitSeq(source);
122
123             } else if (next == '(' || next == ')') {
124                 expr = new StringBuilder(number);
125                 value = valueOfExpr(expr);
126                 number = 0;
127             }
128
129             idx++;
130         }
131
132         return value;
133     }
134
135     /**
136      * Evaluates a term and returns its value.
137      *
138      * @param source
139      *            the {@code StringBuilder} that starts with a term string
140      * @return value of the term
141      * @updates source
142      * @requires <pre>
143      * [a term string is a proper prefix of source, and the longest
144      * such, s, concatenated with the character following s, is not a prefix
145      * of any term string]
146      * </pre>
147      * @ensures <pre>
148      * valueOfTerm =
149      *   [value of longest term string at start of #source]  and
150      * #source = [longest term string at start of #source] * source
151      * </pre>
152      */
153     private static int valueOfTerm(StringBuilder source) {
154         assert source != null : "Violation of: source is not null";
155
156         int idx = 0;
157         int number = 0;
158         int value = 0;
159         StringBuilder factor;
160
161         while (idx < source.length()) {
162             Character next = source.charAt(idx);
163
164             if (Character.isDigit(next) || next == '(' || next == ')') {
165                 number = number + next;
166             } else if (next == '*') {
167                 factor = new StringBuilder(number);
168                 number = valueOfFactor(factor);
169                 value = number * valueOfTerm(source);
170                 number = 0;
171             } else if (next == '/') {
172                 factor = new StringBuilder(number);
173                 number = valueOfFactor(factor);
174                 value = value / number;
```

```
175                 number = 0;
176             }
177
178             idx++;
179         }
180
181         // This line added just to make the program compilable.
182         return value;
183     }
184
185     /**
186      * Evaluates an expression and returns its value.
187      *
188      * @param source
189      *            the {@code StringBuilder} that starts with an expr string
190      * @return value of the expression
191      * @updates source
192      * @requires <pre>
193      * [an expr string is a proper prefix of source, and the longest
194      * such, s, concatenated with the character following s, is not a prefix
195      * of any expr string]
196      * </pre>
197      * @ensures <pre>
198      * valueOfExpr =
199      *   [value of longest expr string at start of #source]  and
200      * #source = [longest expr string at start of #source] * source
201      * </pre>
202      */
203     public static int valueOfExpr(StringBuilder source) {
204         assert source != null : "Violation of: source is not null";
205
206         int value = 0;
207         int number = 0;
208         int idx = 0;
209         StringBuilder term = new StringBuilder(number);
210
211         while (idx < source.length()) {
212             Character next = source.charAt(idx);
213
214             if (Character.isDigit(next) || next == '*' || next == '/'
215                     || next == '(' || next == ')') {
216                 number = number + next;
217             } else if (next == '-') {
218                 term = new StringBuilder(number);
219                 number = valueOfTerm(term);
220                 value = value - number;
221                 number = 0;
222             } else if (next == '+') {
223                 term = new StringBuilder(number);
224                 number = valueOfTerm(term);
225                 value = value + number;
226                 number = 0;
227             }
228
229             idx++;
230         }
231
```

```
232         // This line added just to make the program compilable.
233         return value;
234
235     }
236
237     /**
238      * Main method.
239      *
240      * @param args
241      *            the command line arguments
242      */
243     public static void main(String[] args) {
244         SimpleReader in = new SimpleReader1L();
245         SimpleWriter out = new SimpleWriter1L();
246         out.print("Enter an expression followed by !: ");
247         String source = in.nextLine();
248         while (source.length() > 0) {
249             /*
250              * Parse and evaluate the expression after removing all white space
251              * (spaces and tabs) from the user input.
252              */
253             int value = valueOfExpr(
254                     new StringBuilder(source.replaceAll("[ \t]", "")));
255             out.println(
256                     source.substring(0, source.length() - 1) + " = " + value);
257             out.print("Enter an expression followed by !: ");
258             source = in.nextLine();
259         }
260         in.close();
261         out.close();
262     }
263
264 }
265
```