```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 String[] terms = { "0", "1", "2", "3", "4", "5", "6",
            "7", "8", "9", "*", "/", "(", ")" };

    private static final String[] factors = { "0", "1", "2", "3", "4", "5", "6",
            "7", "8", "9", "(", ")" };

    private static final String[] digits = { "0", "1", "2", "3", "4", "5", "6",
            "7", "8", "9" };

    public static SimpleWriter out = new SimpleWriter1L();

    /**
     * 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 =
     *   [value of longest digit-seq string at start of #source]  and
     * #source = [longest digit-seq string at start of #source] * source
     * </pre>
     */
    private static int valueOfDigitSeq(StringBuilder source) {
        assert source != null : "Violation of: source is not null";

        int idx = 0;
        StringBuilder value = new StringBuilder();
        StringBuilder term = new StringBuilder();
        StringBuilder next = new StringBuilder();

        while (idx < source.length()) {
            next.delete(0, next.length());
            next.append(source.charAt(idx));
            term.append(next);

            for (String check : digits) {
                if (next.toString() == check) {
                    value.append(Integer.toString(valueOfDigit(next)));
                } else {
                    idx = source.length();
                }
            }

            idx++;
        }

        return Integer.valueOf(value.toString());
    }

    /**
     * Evaluates a factor and returns its value.
     *
     * @param source
     *            the {@code StringBuilder} that starts with a factor string
     * @return value of the factor
     * @updates source
     * @requires <pre>
     * [a factor string is a proper prefix of source, and the longest
     * such, s, concatenated with the character following s, is not a prefix
     * of any factor string]
     * </pre>
     * @ensures <pre>
     * valueOfFactor =
     *   [value of longest factor string at start of #source]  and
     * #source = [longest factor string at start of #source] * source
     * </pre>
     */
```

```java
    private static int valueOfFactor(StringBuilder source) {
        assert source != null : "Violation of: source is not null";

        int value = 0;
        StringBuilder digitSeq = new StringBuilder();
        StringBuilder next = new StringBuilder();
        StringBuilder source2 = new StringBuilder();

        if (source.charAt(0) == '(') {
            value = valueOfExpr(source);
        } else {
            value = valueOfDigitSeq(source);
        }

        return value;
    }

    /**
     * Evaluates a term and returns its value.
     *
     * @param source
     *           the {@code StringBuilder} that starts with a term string
     * @return value of the term
     * @updates source
     * @requires <pre>
     * [a term string is a proper prefix of source, and the longest
     * such, s, concatenated with the character following s, is not a prefix
     * of any term string]
     * </pre>
     * @ensures <pre>
     * valueOfTerm =
     *   [value of longest term string at start of #source]  and
     * #source = [longest term string at start of #source] * source
     * </pre>
     */
    private static int valueOfTerm(StringBuilder source) {
        assert source != null : "Violation of: source is not null";

        int value = 0;
        int idx = 0;
        StringBuilder factor = new StringBuilder();
        StringBuilder next = new StringBuilder();
        StringBuilder source2 = new StringBuilder();

        while (idx < source.length()) {
            next.delete(0, next.length());
            next.append(source.charAt(idx));
            factor.append(next);

            for (String check : factors) {
                if (next.toString() == check) {
                    // do nothing lol
                } else if (next.toString() == "*") {
                    source2 = source;
                    source2.delete(0, idx);
                    value += valueOfFactor(factor) * valueOfTerm(source2);

                } else if (next.toString() == "/") {
                    source2 = source;
                    source2.delete(0, idx);
                    value += valueOfFactor(factor) / valueOfTerm(source2);
                } else {
                    idx = source.length();
                }
            }

            idx++;
        }

        return value;
    }

    /**
     * Evaluates an expression and returns its value.
     *
     * @param source
     *            the {@code StringBuilder} that starts with an expr string
     * @return value of the expression
     * @updates source
     * @requires <pre>
     * [an expr string is a proper prefix of source, and the longest
     * such, s, concatenated with the character following s, is not a prefix
     * of any expr string]
     * </pre>
     * @ensures <pre>
     * valueOfExpr =
     *   [value of longest expr string at start of #source]  and
     * #source = [longest expr string at start of #source] * source
     * </pre>
     */
    public static int valueOfExpr(StringBuilder source) {
        assert source != null : "Violation of: source is not null";

        int value = 0;
        int idx = 0;
        StringBuilder term = new StringBuilder();
        StringBuilder next = new StringBuilder();
        StringBuilder source2 = new StringBuilder();

        while (idx < source.length()) {
            next.delete(0, next.length());
            next.append(source.charAt(idx));

            for (String check : terms) {
                out.println(next.toString());
                out.println(check);

                if (next.toString() == check) {
                    term.append(next);
                }
            }

            if (next.toString() == "-") {
                source2 = source;
```

```java
                source2.delete(0, idx);
                value += valueOfTerm(term) - valueOfExpr(source2);
                break;
            } else if (next.toString() == "+") {
                source2 = source;
                source2.delete(0, idx);
                value += valueOfTerm(term) + valueOfExpr(source2);
                break;

            }

            idx++;
        }



        // This line added just to make the program compilable.
        return value;

    }

    /**
     * Main method.
     *
     * @param args
     *            the command line arguments
     */
    public static void main(String[] args) {
        SimpleReader in = new SimpleReader1L();
        SimpleWriter out = new SimpleWriter1L();
        out.print("Enter an expression followed by !: ");
        String source = in.nextLine();
        while (source.length() > 0) {
            /*
             * Parse and evaluate the expression after removing all white space
             * (spaces and tabs) from the user input.
             */
            int value = valueOfExpr(
                    new StringBuilder(source.replaceAll("[ \t]", "")));
            out.println(
                    source.substring(0, source.length() - 1) + " = " + value);
            out.print("Enter an expression followed by !: ");
            source = in.nextLine();
        }
        in.close();
        out.close();
    }

}

```