```java
/**
 * {@code Queue} represented as a {@code Sequence} of entries, with
 * implementations of primary methods.
 *
 * @param <T>
 *            type of {@code Queue} entries
 * @correspondence this = $this.entries
 */
public class HelloWorld {

    public static void main(String[] args) {

    }

    /**
     * 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) {
        int result = valueOfTerm(source);
        while (source.length() > 0
                && (source.charAt(0) == '+' || source.charAt(0) == '-')) {
            char operation = source.charAt(0);
```

```
38              source.deleteCharAt(0);
39              if (operation == '+') {
40                  result += valueOfTerm(source);
41              } else {
42                  result -= valueOfTerm(source);
43              }
44          }
45          return result;
46      }
47
48      /**
49       * Evaluates a term and returns its value.
50       *
51       * @param source
52       *            the {@code StringBuilder} that starts with a
   term string
53       * @return value of the term
54       * @updates source
55       * @requires <pre>
56       * [a term string is a proper prefix of source, and the
   longest
57       * such, s, concatenated with the character following s, is
   not a prefix
58       * of any term string]
59       * </pre>
60       * @ensures <pre>
61       * valueOfTerm =
62       *   [value of longest term string at start of #source]
   and
63       * #source = [longest term string at start of #source] *
   source
64       * </pre>
65       */
66      private static int valueOfTerm(StringBuilder source) {
67          int result = valueOfFactor(source);
68          while (source.length() > 0
69                  && (source.charAt(0) == '*' || source.charAt(0)
   == '/')) {
70              char operation = source.charAt(0);
71              source.deleteCharAt(0);
72              if (operation == '*') {
73                  result *= valueOfFactor(source);
74              } else {
75                  result /= valueOfFactor(source);
```

```
 76            }
 77        }
 78        return result;
 79    }
 80
 81    /**
 82     * Evaluates a factor and returns its value.
 83     *
 84     * @param source
 85     *            the {@code StringBuilder} that starts with a
    factor string
 86     * @return value of the factor
 87     * @updates source
 88     * @requires <pre>
 89     * [a factor string is a proper prefix of source, and the
    longest
 90     * such, s, concatenated with the character following s, is
    not a prefix
 91     * of any factor string]
 92     * </pre>
 93     * @ensures <pre>
 94     * valueOfFactor =
 95     *   [value of longest factor string at start of #source]
    and
 96     * #source = [longest factor string at start of #source] *
    source
 97     * </pre>
 98     */
 99    private static int valueOfFactor(StringBuilder source) {
100        int result = 0;
101        if (source.charAt(0) == '(') {
102            source.deleteCharAt(0);
103            result = valueOfExpr(source);
104            source.deleteCharAt(0);
105        } else {
106            result = valueOfDigitSeq(source);
107        }
108        return result;
109    }
110
111    /**
112     * Evaluates a digit sequence and returns its value.
113     *
114     * @param source
```

```
115      *            the {@code StringBuilder} that starts with a
    digit-seq string
116      * @return value of the digit sequence
117      * @updates source
118      * @requires <pre>
119      * [a digit-seq string is a proper prefix of source, which
120      * contains a character that is not a digit]
121      * </pre>
122      * @ensures <pre>
123      * valueOfDigitSeq =
124      *   [value of longest digit-seq string at start of
    #source]  and
125      * #source = [longest digit-seq string at start of #source]
    * source
126      * </pre>
127      */
128     private static int valueOfDigitSeq(StringBuilder source) {
129         String result = "";
130         while (source.length() > 0 &&
    Character.isDigit(source.charAt(0))) {
131             result += Integer.toString(valueOfDigit(source));
132             source.deleteCharAt(0);
133         }
134         return Integer.parseInt(result);
135     }
136
137     /**
138      * Evaluates a digit and returns its value.
139      *
140      * @param source
141      *            the {@code StringBuilder} that starts with a
    digit
142      * @return value of the digit
143      * @updates source
144      * @requires 1 < |source| and [the first character of
    source is a digit]
145      * @ensures <pre>
146      * valueOfDigit = [value of the digit at the start of
    #source]  and
147      * #source = [digit string at start of #source] * source
148      * </pre>
149      */
150     private static int valueOfDigit(StringBuilder source) {
151         return Character.digit(source.charAt(0), 10);
```

```
152     }
153 }
```