```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);
            source.deleteCharAt(0);
            if (operation == '+') {
                result += valueOfTerm(source);
            } else {
                result -= valueOfTerm(source);
            }
        }
        return result;
    }

    /**
     * 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) {
        int result = valueOfFactor(source);
        while (source.length() > 0
                && (source.charAt(0) == '*' || source.charAt(0) == '/')) {
            char operation = source.charAt(0);
            source.deleteCharAt(0);
            if (operation == '*') {
                result *= valueOfFactor(source);
            } else {
                result /= valueOfFactor(source);
            }
        }
        return result;
    }

    /**
     * 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>
     */
    private static int valueOfFactor(StringBuilder source) {
        int result = 0;
        if (source.charAt(0) == '(') {
            source.deleteCharAt(0);
            result = valueOfExpr(source);
            source.deleteCharAt(0);
        } else {
            result = valueOfDigitSeq(source);
        }
        return result;
    }

    /**
     * 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) {
        String result = "";
        while (source.length() > 0 && Character.isDigit(source.charAt(0))) {
            result += Integer.toString(valueOfDigit(source));
            source.deleteCharAt(0);
        }
        return Integer.parseInt(result);
    }

    /**
     * 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) {
        return Character.digit(source.charAt(0), 10);
    }
}
```