Count Executable Lines of Code (ELOC)

Task Description

The application counts ELOC for Java source code, as well as for any other programming language, where a single line comment starts with “//” and multiple lines of comment (or alternatively inline comment(s)) start with "/*” and end with "*/”.

The application assumes that each comment is properly written, i.e. single line comment is correctly opened, and multiple lines of comment correctly opened and closed. If there is any character outside the comments (excluding white space) it is counted as ELOC. The application does not assume that the source code outside the comments can compile (which would require quite a significant application of its own).

At first glance writing such an application is quite simple. And indeed, it would be the case, only if the ELOC and the comments conform to the best practices for clean code. However, this application assumes only that each comment is properly written and then takes on all possible weird scenarios, which even though could not adhere to the best practices for clean code, present perfectly valid source code, which would compile and run (if, of course, the ELOC adheres to syntax for the specific programming language).

Such corner cases make the task more difficult than it might seem at first.

Content

The source code of the application is pretty much self-explanatory. Each method has a descriptive name about what it would do, and each field has a descriptive name about its purpose. Here is a synopsis about it.

class CountELOC
contains the source code for the application.

public int countExecutableLinesOfCode(String text)
starts the application

private boolean hasExecutableCode(String line)
helper method: if it returns “true”, the count for ELOC is increased by 1

private boolean startsMultipleLinesCommentAtSomeLaterPointOnTheLine(String line)
helper method: checks for a valid start of multiple lines of comment at all positions following the current character. With valid start of multiple lines of comment, it is meant starts that are not closed on the same line and are not within a single line comment. When this method is called, there is enough information to end the loop, i.e. ELOC is found, we just need to know whether to set "isInMultipleLinesComment" to "true" or "false" before ending the loop.

Examples that would return "true":

/*some comment */ current position with ELOC /*some comment */ /*some comment */ /*

some comment (opened at preceding lines) */ current position with ELOC /*some comment */ /*some comment */ /*

some comment (opened at preceding lines) */ current position with ELOC /*

some comment (opened at preceding lines) */ current position with ELOC /* ////

Examples that would return "false":

/*some comment */ current position with ELOC /*some comment */ /*some comment */

some comment (opened at preceding lines) */ current position with ELOC /*some comment */ /*some comment */

some comment (opened at preceding lines) */ current position with ELOC

some comment (opened at preceding lines) */ current position with ELOC /// /*

class TestCountELOC
contains JUnit tests for the application.

Each test case description is contained in the corresponding test method name.

To make the input for each case as easily observable as possible, Java Text Blocks are applied, excluding the case for empty String, and of course for “null” String. And for the same purpose, if the method is expected to return ELOC greater than zero, the number of ELOC is fixed at five (**private static final int EXPECTED_ELOC = 5**). All sorts of combinations are made with these five ELOC.

If the method is expected to return ELOC equal to zero, the comparison is made against field **private static final int NO_ELOC = 0**.

Each ELOC and comment (if it is not blank) contains the printable ASCII chars starting at code point 32 and ending at code point 127, inclusive range (**private static String PRINTABLE_CHARS_ASCII**).

These are inserted at the end of the Text Block through replaceAll(value, external value) method, inbuilt in Java class String. As of December 2023, there is no way to directly insert external values in Java Text Blocks. In JavaScript, for instance, it is possible to do that with Template Strings, such as `some text ${external value} some text`. Although not the same, a sort of alternative is Java String concatenation but, as mentioned, using Strings instead of Text Blocks would make the input less easily observable.

The replaceAll(value, external value) method applies Matcher.quoteReplacement(PRINTABLE_CHARS_ASCII) for the external value in order to avoid the special meaning of control characters in the external value.