Java Academy Technical Task:

Time for solving: 2 days

Solutions are accepted on GitHub only

Problem Statement:

Given the set of characters that compose the logic word (L, O, G, I, C), you need to implement an algorithm that reads an input, identify words which contains any of the LOGIC characters and prints the frequency of those characters based on the combination of LOGIC characters and length of the words. The frequency is how many LOGIC characters are in each word divided by the total number of LOGIC characters in the whole input. At the end you also need to print the frequency of LOGIC characters based on the total number of characters of the given input.

For both requirements you should consider words excluding spaces and special characters.

For example, for the phrase: "I love to work in global logic!", we see that the LOGIC characters appear 15 times. We need to group them by combination of LOGIC characters and size of the word. We also see that our phrase contains 31 chars in total, however 7 of them are special chars so in order to compute the second requirement we take into account only 24;

In this case, your algorithm would print:

```
{(i), 1} = 0.07 (1/15)
{(i), 2} = 0.07 (1/15)
{(o), 2} = 0.07 (1/15)
{(o), 4} = 0.07 (1/15)
{(l, o), 4} = 0.13 (2/15)
{(l, o, g), 6} = 0.27 (4/15)
{(l, o, g, i, c), 5} = 0.33 (5/15)
TOTAL Frequency: 0.63 (15/24)
```
(This is just a basic example of output)

Assumptions:

- The list of special chars are: `(  !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~);` (Yes it includes the blank space :);
- The analysis should be case insensitive meaning that G and g count as the same char;
- The output should be displayed formatted to 2 decimals rounded to the nearest decimal for all combinations (e.g. 1/15 = 0.07 and 5/15 = 0.33);
- The output should be ordered from the lowest frequency first to the highest frequency last (as in example above);
- Words that have the same unique combination of LOGIC characters and same length must be added to the same group, regardless of how many times the characters appear (e.g. “plate” and “level” both fall in the same category: {(l), 5} and in this case the “l” char must be counted 3 times);

Notes:

The guidelines above are the main requirements of the task, we encourage you to go beyond it, think about how to make the application configurable and extensible (what if we need to change the list of characters we are looking for?), think about reading the input / storing the output from / into files or maybe using a persistence layer for configuring / storing results or even showing the results in a web layer if you want. Think about assumptions as well and feel free to add yours if you think they are necessary. Those are just some examples of what can be done, we leave it open to you to add any features that you think would improve the solution. Any added feature (given that it makes sense in the context of the task) will count as a plus in the evaluation.