Frequency Analysis

A **Caesar cipher** is a simple method of encoding messages. Caesar ciphers use a substitution method where letters in the alphabet are shifted by some fixed number of spaces to yield an encoding alphabet. A Caesar cipher with a shift of 1 would encode an A as a B, an M as an N, and a Z as an A, and so on. And a Caesar cipher with a shift of 5 would encode an A as a F, an M as an R, an V as an A, and a Z as an E, and so on. Given there are 26 letters, there are 25 different variations of rotation cipher. (rotating by 26 only gives you what you start with, and rotating by 27 is the same as rotating by 1). The method is named after Roman leader Julius Caesar, who used it in his private correspondence

Volumes of English texts have been analyzed and the relative rate of occurrences (relative frequency) for each letter has been averaged. With this information you will determine which of the possible rotation provides the most English-looking output.

In this problem you will implement six methods in the `FrequencyAnalysis` class which will probabilistically decode an encrypted message in a Caesar cipher with an unknown number of rotations. The `FrequencyAnalysis` class has a single constructor with a non empty String parameter which contains the encrypted `message` comprised only of spaces, upper case, and lower case letters. The class also contains a `private static double[]` instance variable, `FREQ`, which contains the relative frequency of each letter of the alphabet. `FREQ[0]` contains the relative frequency of the letter 'A', `FREQ[1]` contains the frequency of the letter 'B', etc.

The first method is `getLetterFrequency(String let)` which determines the frequency of the parameter `let` (both upper and lower case. That is 'A' and 'a' are the same letter) in `message` (the `String` passed to the constructor). The frequency is determined by dividing the number of occurrence of the letter `let` by the total number of letters (not counting spaces) in `message`.

The following code shows the results of the `getLetterFrequency` method.

The following code	Returns
`FrequencyAnalysis fa =  new FrequencyAnalysis ("Aolyl pz hsdhfz vul tvyl ibn av mpe");`	
`fa.getLetterFrequency("A")`	0.0714285714285= 2./28
`fa.getLetterFrequency("v")`	0.1071428571428= 3./28

The second method is `rotateBack(String let, int rot)` which rotates the parameter `let` backwards `r` values. The return value should have the same case as the parameter. You may assume `let.length == 1` and `str.equals( " " ) == false`.

The following code shows the results of the `rotateBack` method.

The following code	Returns
`fa.rotateBack("D", 3)  // D is uppercase, return uppercase`	"A"
`fa.rotateBack("w", 13) // w is lowercase, return lowercase`	"n"

The third method is `calculateDeviation(String let, int rot)` which, given a letter `let` (`let.length == 1` and `let != " "`), and a rotation value `rot`, returns the deviation.
For example, given `let == "v"` and `rot = 4`. Find the frequency `let` appears in `message` (the encrypted text) and calculate the deviation from the expected value of `"r"` (the 18th letter or, `"v"`, rotated back 4).
That is: `Math.abs( ( getLetterFrequency("r") - FREQ[18] ) / FREQ[18] )`

```
  Or  (     0.1071428571428        -  0.05987)  /  0.05987 ≈ 0.7895917345
```

if `message.indexOf( let ) < 0, return 0.0`, that is, if `message` does not contain `let`.

The following code shows the results of the `calculateDeviation` method.

The following code	Returns
`FrequencyAnalysis fa =  new FrequencyAnalysis ("Aolyl pz hsdhfz vul tvyl ibn av mpe");`	
`double cd = fa.calculateDeviation("v", 4);` `boolean flag = Math.abs( cd -  0.7895917345) < 0.001;`	`flag == true`
`double cd = fa.calculateDeviation("p", 11);` `boolean flag = Math.abs( cd -  0.02099158) < 0.001;`	`flag == true`
`double cd = fa.calculateDeviation("r", 7);` `boolean flag = Math.abs( cd -  0.0) < 0.001;`	`flag == true`

The fourth method is `totalDeviation(int rot)` which returns the total deviation (of all characters, not spaces) for given rotation `rot`.
For example, given `r = 23`, return `fa.calculateDeviation("a", 23) + fa.calculateDeviation("b", 23) + … + fa.calculateDeviation("z", 23)`

The following code shows the results of the `totalDeviation` method.

The following code	Returns
`FrequencyAnalysis fa =  new FrequencyAnalysis ("Aolyl pz hsdhfz vul tvyl ibn av mpe");`	
`double td = fa.totalDeviation(23);` `Boolean flag = Math.abs( td -  86.07738338432043) < 0.001;`	`flag == true`

The fifth method is `bestBackRotation()` which returns number of rotation that minimizes the total deviation for message.

The following code shows the results of the `bestBackRotation` method.

The following code	Returns
`FrequencyAnalysis fa =  new FrequencyAnalysis ("Aolyl pz hsdhfz vul tvyl ibn av mpe");`	
`fa.bestBackRotation();`	7

The sixth (and final) method is `getMessage()` which, using the best rotation value as determined by `bestBackRotation`, decrypts the message (remember, spaces are NOT rotated).

The following code shows the results of the `getMessage()` method.

The following code	Returns
`FrequencyAnalysis fa = new FrequencyAnalysis ("Aolyl pz hsdhfz vul tvyl ibn av mpe");`	
`fa.getMessage();`	`"There is always one more bug to fix"`

FYI: here is the relative Frequency chart for all 26 letters which has been included in the `Frequency Analysis` class as a `static double[]` instance variable.

letter	frequency	letter	Frequency
A	0.08167	N	0.06749
B	0.01492	O	0.07507
C	0.02782	P	0.01929
D	0.04253	Q	0.00095
E	0.12702	R	0.05987
F	0.02228	S	0.06327
G	0.02015	T	0.09056
H	0.06094	U	0.02758
I	0.06996	V	0.00978
J	0.00153	W	0.02360
K	0.00772	X	0.00150
L	0.04025	Y	0.01974
M	0.02406	Z	0.00074

this page intentionally left blank