Programming IT 2018-2019

Assessed Exercise 2

Handed out: 6th Nov 2018
Deadline: 1730, 20th November 2018

Introduction

In cryptography, a cipher can be used to encode a document such that it cannot be read by anyone without the correct cipher keyword.
In this exercise, you will implement one or two different ciphers that will either **encode** or **decode** the text in an input file, writing the results into an output file. In addition, your code will summarise the frequencies of different letters in the output.

Requirements

1. Your code should ask the user for the name of an input file (that you can assume will be located within the same directory as the code).
2. The filename that the user will enter will be something of the form xxxxxP or xxxxxxC. If the filename ends in P, your code should **encode** and if in C, your code should **decode**. Your code will add the necessary extension (".txt") to the filename before attempting to read the file.
3. Your code will then open the file and read and encode/decode each character before writing the coded/decoded version in a file names xxxxxC or xxxxxxD respectively (where xxxxxx is the same as the name given in the input). For example:
    a. If the user inputs "HelloP", the program will load "HelloP.txt" and encode it, saving the result into a file called "HelloC.txt"
    b. If the user inputs "ByeC", the program will load "ByeC.txt", decode it, and write the results into "ByeD.txt"
4. The cipher should encode/decode only capital alphabetic characters (A-Z). Any other characters should be written into the output exactly as they appear in the input.
5. You should implement the MonoAlphabetic cipher first. If you have time, you should also implement the Vignere cipher. If you do implement the Vignere cipher, you should include (in your main method) the code that will create the Vignere cipher object instead of the MonoAlphabetic, but comment it out. E.g. in your code somewhere you will have something like this:

```
MonoAlphabetic m = new MonoAlphabetic(keyword);
// Vignere m = new Vignere(keyword);
```

    allowing us to comment the first line and uncomment the second, and run the Vignere cipher instead.
6. Regardless of the cipher type, or whether your program is encoding or decoding, you should also write out an additional file (xxxxxxF.txt) that includes the letter

frequencies for each alphabetic upper case letter *in the output file* (i.e. in the encoded or decoded file, depending on which operation your program is performing). The file should look like the example below, where the average frequencies are provided on Moodle. The “Freq” column gives the **number** of times the letter is seen in the output. The Freq% is the percentage of times it appears (number divided by the total number of upper-case alphabetic characters. AvgFreq% is as provided and diff is Freq% minus AvgFreq%. At the end of the file should be listed the most frequent letter and its percentage. If more than one letter share the highest frequency, the one **latest** in the alphabet should be shown.

```
LETTER ANALYSIS

Letter  Freq  Freq%  AvgFreq%  Diff
   A    34    7.6      8.2    -0.6
   B    12    2.7      1.5     1.2
   C    19    4.3      2.8     1.5
   D     9    2.0      4.3    -2.3
   E    55    12.4    12.7     -0.3
  ...
   Y    16    3.6      2.0     1.6
   Z     0    0.0      0.1    -0.1

The most frequent letter is E at 12.4%
```

7. After the user has entered the filename, they should be prompted for a keyword that will be used for the encryption (see below). Your code should check that the keyword does not have any repeated letters. For example, TIGER is an acceptable keyword but TIGGER is not.

The MonoAlphabetic Cipher

The Mono Alphabetic cipher is the simpler of the two ciphers in the exercise. A **keyword** is provided and from which an encryption array is created. The encryption array is an array of characters of length 26. The initial positions are occupied by the keyword and the remaining positions are occupied by any characters not in the keyword, in alphabetical order. For example, for a keyword of TIGER, the encryption array would look like:

T	I	G	E	R	A	B	C	D	F	H	J	K	L	M	N	O	P	Q	S	U	V	W	X	Y	Z

To encrypt a character (say ‘H’) we find its position in the alphabet (H is the 8th letter in the alphabet) and then use the character at that position in the array (in this case, C). As mentioned above, characters that are not upper-case alphabetic characters are just copied straight into the output.
As an example, with this keyword, the sentence

`HELLO EVERYONE`
would be encrypted as
`CRJJM RVRPYMLR`
(A useful debugging test: what should the keyword A do? Or ABCDEFG, etc)
To decode a character, we do the opposite. So, to decode the character C we first find C in our encryption array (it's in the 8th position) and then return the 8th letter of the alphabet. R is in the 5th position in the array so it is E, etc.
Note that if you encode and decode and want to end up back with what you started, you need to ensure that you use the same keyword both times!

The Vignere cipher

The Vignere cipher uses a two dimensional encryption array. It has one row for every character in the keyword. And each row lists the characters in alphabetical order, starting from the appropriate keyword character, and looping back around to the start of the alphabet when it gets to Z. For example, here is the Vignere encryption array for the keyword TIGER:

T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q

To encode a message, we change row every character. So, for the first character, we use the first row and do the same encryption as the mono alphabetic cipher (e.g. in this case, H will become A (the 8th character in the first row). For the next character, we move to the next row (E will become M) and so on. After we have encrypted using the bottom row, we move back to the first one again.
Using the encryption array above, the sentence:
`HELLO EVERYONE`
will be encrypted as
`AMRPF XDKVPHVK`
(Debugging tip, what should happen with a keyword of "A"?)
For decoding, we again do the reverse. The first character we decode, we use the first line, find the position of the character of interest and then return the letter in the alphabet corresponding to that position, and then move to row 2. E.g. `ZWUHSRM` is decoded as `GOODBYE`.

Some suggestions

1. Make classes for the cipher(s).
2. [optional...no extra marks, but useful] The two cipher objects will share functionality. Perhaps some polymorphism could be done here (maybe an interface?)

3. Chars can be cast into ints (and vice versa). E.g. `(int)('A')` is 65. `(int)('Z')` is 90, and `(char)67` is 'C'. This is useful when determining positions in arrays (the nth letter of the alphabet is `(char)(n+65)`). It is also useful when determining if a particular character is an upper-case alphabetic character between A and Z.
4. The `read()` method of `FileReader` returns the integer representation of the next character in a file. It returns -1 when you have reached the end of a file.
5. Keep methods short. If they get long, write more methods!
6. Think about the model, view, controller paradigm. In this case, the cipher objects constitute the model. The view and controller are the file handling stuff. Avoid putting things like file reading and writing inside the cipher classes.
7. The `charAt()` method of Strings is useful (e.g. when determining if the last letter of the file name is a P or a C.
8. Make a class called LetterFrequencies to keep track of the number of times each character has appeared and produce the report.
9. Break the problem down into manageable chunks. Test as you go along. My advice would be to forget about files to start with and get the mono cipher working. The toughest part of this is probably creating the encryption array. My advice is to think through an algorithm for doing this on paper before you start writing some code.
10. On Moodle you will find the average letter frequencies (in a text file - just copy and paste the code from this file into your class) and four example files:
    a. CommonP.txt - example text to be encoded
    b. CommonC.txt - text encoded with the keyword FLAMINGO (MonoCipher)
    c. CommonD.txt - text decoded with the same keyword
    d. CommonF.txt - letter frequencies for the **encoding** step (i.e. the file output when the program created CommonC.txt)

Report

Write a short summary (no more than one page, excluding screen dumps) of the final state of your program in a Word file. It should indicate how closely your program matches the specification. There should be a statement detailing the inputs that you have used to test that the program is working correctly before submitting the final version. This statement should include a description of the results you would expect when executing the program with your data. The report should include:

- Your ***name and student number*** at the top.
- Any assumptions you have made that have affected your implementation.
- Any known deficiencies, e.g., missing functionality or discrepancies between your expected and actual results and how you might correct them. If your program does meet the specification, all that is needed is ONE sentence stating this.
- Details of the test data used and the results expected and obtained should be provided, using screenshots as appropriate. Test data should include a plaintext file, its encrypted and decrypted versions, the letter frequencies file and the cipher arrays, in each of the cases of the monoalphabetic and the Vigenère ciphers. It should also include evidence of testing for incorrect input involving the keyword and message file name.

Submission

You should submit a single zip file containing your .java files and your report (preferable .pdf, .doc also ok) through moodle. Normal penalties for late submission apply.

Marking Scheme

The assignment will be marked in bands on the scale A1,A2,……..F3,G1,G2,H. Marks will be awarded according to the quality of these separate elements:

Element	**Weighting**	**Contents**
Functionality (Mono cipher)	50%	Proportion of the specified functionality implemented, and program correctness
Design (Mono cipher)	15%	Choice of methods and instance variables within classes, design of methods
Documentation and Testing (Mono cipher)	20%	Readability of code (comments, layout, indentation), quality of status report, selection of test data
Vigenère cipher	15%	Functionality, design and documentation in respect of Vigenère cipher

Notes:

- *We are expecting you to stick rigidly to the specification. Don't spend time implementing functionality that was not asked for – it will not receive additional credit and your time would be better spent on other tasks!*
- *If major aspects of the functionality are not even attempted then the credit given for the Design, Documentation and Testing will be reduced. So for example if only half of the functionality is attempted then it will only be possible to obtain half marks for Design, Documentation and Testing*.