CSCI 3901 Assignment 1

Due date: 11:59pm Wednesday, September 28, 2022 in Brightspace

Problem 1

Goal
Get practice in decomposing a problem, creating a design for a program, and implementing and testing a program. Practice basic Java programming.

Background
Substitution ciphers[1] are a family of text encryption ciphers in which each letter of your message (called the plain text) is replaced by some other letter to form the encrypted message (called the cypher text). Which letter is used to replace another is defined by the encryption key.

For example, if the encryption key says to replace “e” with “x”, “t” with “b”, “h” with “m”, and “s” with “r” (within a longer key) then the plaintext “these” would become the cyphertext “bmxrx”.

Decrypting a substitution cypher then reverses the operation. You use the encryption key backward: seeing what encrypted letter came from which original letter and then making the substitutions to the cyphertext message.

In a substitution cipher, any mapping from the original letters to the encrypted letters is possible.

A characteristic of text encrypted with a substitution cipher is that the letter frequencies[2] of the original message and of the encrypted message are the same except that the frequencies are attributed to different letters. Consequently, a first step in decoding an encrypted message without the encryption key (a process called breaking the code) has you compute the letter frequencies in the ciphertext and matching the letters to the known letter frequencies in the language that you expect the message to be in. For example, the top 5 letter frequencies in English are “e” at 12.7%, “t” at 9.1%, “a” at 8.2%, “o” at 7.5%, and “n” at 6.7%. Then if a cyphertext had the 5 more frequent letters being w, c, h, m, and q then we would start with an encryption key that had mapped “e” to “w”, mapped “t” to “c”, mapped “a” to “h”, mapped “o” to “m”, and mapped “n” to “m”.

[1] https://en.wikipedia.org/wiki/Substitution_cipher
[2] https://en.wikipedia.org/wiki/Letter_frequency

Each language has different letter frequencies, so you can also get a guess at the language of the message by comparing the letter frequency distribution of your ciphertext message with the distribution for each language; if one language has a much smaller difference in frequency distribution with your ciphertext then you would begin by assuming the message originated in that language.

In any encryption or decryption process, you leave punctuation and white space as-is, for this assignment. You do, however, need to preserve the upper/lower case of the letter.

Problem

Write a Java class, called “SubstitutionCipher” that helps us to decrypt a message that has been encrypted using a substitution cipher, will help us identify if the message is a Caesar cipher, and will help us determine the language of origin of a ciphertext.

You will also submit a small “main” method in a class called “A1” that will demonstrate a minimal use of the Java class.

SubstitutionCipher

The SubstitutionCipher class must have the following methods:

Constructor SubstitionCipher( ) – initializes an instance of the object

Constructor SubstitionCipher(String name, Map<Character, Character>) key) – initializes an instance of the object with an already-known encryption key, to be identified by the given name. In the Map parameter, the key part of the map is the plaintext character and the Map value is the encrypted character.

Boolean originalLanguage(String name, String filename) – the given filename contains a sample of a language, which will be identified by the given name. Use the content of the filename to build the letter frequency table for that language. Return “true” if the object will be able to proceed with the given input for decryptions, and return “false” otherwise.

Boolean ciphertext(String filename) – link the given object to the content of the given file content as content to be decrypted. Return “true” if the object will be able to proceed with the given input for decryptions, and return “false” otherwise.

String decodeText( ) – decode the ciphertext currently associated with the object using the encryption key currently defined in the object. Return the decoded message as a String. Return a null for the String if an error occurred.

Boolean setDecodeLetter(Character plaintextChar, Character ciphertextChar) – update any internal encryption key with having the given plaintext character being mapped to the given ciphertext character. Return “true” if the mapping is one that can be encoded, and “false” otherwise / in case of error.

Map<Character, Character> getKey () – return the encryption key being used by the object. The key to the Map is the plaintext character and the Map value is the encrypted character.

Boolean keyIsValid( ) – return “true” if the currently-stored encryption key is valid for trying to decode the associated ciphertext. A key is valid if each plaintext letter maps to exactly one ciphertext letter and if every letter in the ciphertext message has some mapping in the key.

Boolean guessKeyFromFrequencies(String language) – using the frequencies from the given language (given before with originalLanguage()), create an encryption key by mapping the letter frequencies between the language and the ciphertext frequencies. If letter frequencies are tied in one distribution then resolve the order of frequencies with the letter order in the alphabet. Return “true” if you were able to create a valid encryption key and “false” otherwise.

String matchLanguage( ) – return the name of the language whose letter frequency distribution most closely matches that of the ciphertext. To compare frequency distributions, sort the distributions in descending order and sum the absolute difference between frequency percentages of the distributions. The language with the smallest sum is the best match. Table 1 shows an example of the calculation difference, where the best language match comes out as language A. Return the name of the language or return null in the case of an error.

Table 1 Sample matching of ciphertext letter frequency distribution to different languages.

ciphertext		Language distributions				Differences		
		Lang. A	Lang. B	Lang. C		Lang. A	Lang. B	Lang.C
.11		.12	.14	.16		.01	.03	.05
.08		.09	.08	.10		.01	.00	.02
.08		.08	.07	.07		.00	.01	.01
.07		.07	.07	.06		.00	.01	.01
.06		.06	.07	.05		.00	.01	.01
Sum of difference						.02	.06	.10

A1

The A1 class has the minimal code to show that you can generate some decryption. The program will ask the user for two file names from the keyboard. The first file name will be sample text from one language from which we can infer letter frequencies. The second file name is a file with ciphertext. Given these two inputs, your program will print its best guess at a decryption of the ciphertext to the screen.

Assumptions

You may assume that

- All text that is encrypted uses letters a-z (possibility upper case). All other characters like punctuation or accented characters will be left as-is in the encrypted file.
- No line in a file will have more than 120 characters in it.

Constraints

- Write your solution in Java. You are allowed to use data structures from the Java Collection Framework.
- If in doubt for testing, I will be running your program on timberlea.cs.dal.ca. Correct operation of your program shouldn't rely on any packages that aren't available on that system.

Marking scheme

- Documentation (internal and external) – 3 marks
- Program organization, clarity, modularity, style – 4 marks
- Able to manage a key manually and decode a message– 5 marks
- Ability to derive a key from a given language– 3 marks
- Ability match a language to a ciphertext– 3 marks
- Ability to decode a message with your main() method – 2 marks

The majority of the functional testing will be done with an automated script or JUnit test cases

Test cases for SubstitutionCipher class

Input Validation

SubstitititutionCipher(String name, Map<Character, Character> key):

- Null provided for name
- Null provided for the key

originalLanguage(String name, String filename) :

- Null for name
- Null for filename

ciphertext(String filename) :

- Null for filename

setDecodeLetter(Character plaintextChar, Character ciphertextChar) :

- plaintextChar is null
- ciphertextChar is null
- plaintextChar is not an encodable letter

- ciphertextChar is not an encodable letter

guessKeyFromFrequencies(String language) :
- language is null value

Boundary Cases

SubstitionCipher(String name, Map<Character, Character> Key):
- Name is empty string
- Key is an empty Map

originalLanguage(String name, String filename) :
- Name is empty string
- Filename is empty string
- Name is not already in use in the object
- Name is already used in the object
- File doesn't exist / can't be opened
- File is empty
- File has a single character
- File has a single line of characters
- File contains a line of text with the maximum of 120 characters on a line

ciphertext(String filename) :
- Filename is empty string
- File doesn't exist / can't be opened
- File is empty
- File has a single character
- File has a single line of characters
- File contains a line of text with the maximum of 120 characters on a line

decodeText():
- Decode a single character
- Decode a line of 120 characters
- Ciphertext uses first encodable letter
- Ciphertext uses last encodable letter

setDecodeLetter(Character plaintextChar, Character ciphertextChar) :
- plaintextChar is first encodable letter
- plaintextChar is last encodable letter
- ciphertextChar is first encodable letter
- ciphertextChar is last encodable letter

getKey() :
- call when key has no characters in it

keyIsValid() :
- call when key has no characters in it

guessKeyFromFrequencies(String language) :
- language is empty string
- invoke when no key stored in the object
- invoke when there is a key already stored in the object

matchLanguage() :
- no languages stored
- only one language stored

Control Flow

SubstitionCipher(String name, Map<Character, Character> Key):
- Map is a valid key that could be used for some ciphertext
- Map is not a valid key because some letter is mapped twice

originalLanguage(String name, String filename) :
- Inserting a first language set
- Inserting a second language set
- File has multiple lines of text

ciphertext(String filename) :
- No ciphertext already stored in the object
- Ciphertext already associated with this object

decodeText():
- Decode multi-line ciphertext
- Decode ciphertext with a valid key
- Decode ciphertext with an invalid key
- Ciphertext contains punctuation characters
- Ciphertext contains upper and lower case letters
- Successful decode key coming from constructor
- Successful decode key coming from manual configuration with setDecodeLetter
- Successful decode key coming from guessKeyFromFrequencies

setDecodeLetter(Character plaintextChar, Character ciphertextChar) :
- Letter mapping is valid and not already in the key
- Letter mapping is valid, but plaintextChar already has a ciphertext character in the key
- ciphertextChar not already in the key
- ciphertextChar is already in the key
- plaintextChar is lower case letter
- ciphertextChar is lower case letter
- plaintextChar is upper case letter

- ciphertextChar is upper case letter

getKey() :
- object contains a valid key
- object contains an invalid key

keyIsValid() :
- call when key is valid
- call when key contains more the same ciphertext letter more than once
- call when some letter of the ciphertext does not appear as a ciphertext letter in the key

guessKeyFromFrequencies(String language) :
- language is one stored in the object with originalLanguage
- language is one not stored in the object with originalLanguage
- language file contains more characters than those seen in the ciphertext
- language file contains fewer characters than those seen in the ciphertext
- All frequencies in the ciphertext and in the language file distinct
- Some ties in frequencies in the ciphertext
- Some ties in frequencies in the language file
- Invoke when the frequencies in the language map are identical to those in the ciphertext
- Invoke when the set of characters in the language file is greater than in the ciphertext
- Invoke when the set of characters in the ciphertext is greater than in the language file
- Invoke when the language file contains punctuation
- Invoke when the language file contains upper and lower case letters

matchLanguage() :
- Many languages defined from which to choose
- One language frequency is an exact match for the ciphertext
- All languages match the ciphertext equally
- Some language has a larger character set than appears in the ciphertext
- Some language has a smaller character set than appears in the ciphertext

Data Flow

getKey() :
- no key created yet in the object

keyIsValid() :
- call when no key created in the object

guessKeyFromFrequencies(String language) :
- invoke when no ciphertext associated with the object

guessKeyFromFrequencies(String language) :

- invoke when no key stored in the object

matchLanguage():

- invoke when no ciphertext stored with the object
- invoke when no languages stored with the object

Orderings of methods (many of these are likely to overlap with previous tests):

- try a typical invocation:
  - create object
  - call ciphertext
  - call originalLanguage
  - call matchLanguage
  - call guessKeyFromFrequencies
  - call keyIsValid
  - call getKey
  - call decodeText
- Check for spots where order shouldn't matter
  - Call originalLangauge then ciphertext before matchLanguage
  - Call guessKeyFromFrequencies before matchLanguage
  - Call getKey before keyIsValid
  - Call decodeText before getKey
  - Call decodeText before keyIsValid
- Check for orders that leave information unknown
  - Call guessKeyFromFrequencies before ciphertext
  - Call guessKeyFromFrequencies before originalLanguage
  - call keyIsValid before guessKeyFromFrequencies, setDecodeLetter, or constructor with frequencies
  - call getKey before guessKeyFromFrequencies, setDecodeLetter, or constructor with frequencies
  - call decodeText before ciphertext
  - call decodeText before guessKeyFromFrequencies, setDecodeLetter, or constructor with frequencies