| | |
|---|---|
| **Course:** | **INFO1214 Fall 2022 Project** |
| **Professor:** | **Bill Pulling ♦ Tony Haworth ♦ Lynn Koudsi** |
| **Project:** | ***Caesar Salad : Version 1.0*** |
| **Due Date:** | **Code in FOL drop box by Monday, Nov. 28, by 11:59 p.m.** |

## Description

The history of cryptography dates back to the Roman era of Julius Caesar. All communications between his legions had to be delivered by couriers. If the couriers were captured, the enemy could read Caesar's orders and the enemy would be able to take counter measures. To prevent his plans from becoming known to the enemy, Caesar began to encipher (encode) his dispatches by using what became known as a "*Caesar shift*".
The Caesar shift was a very simple, yet, in its time, a very effective encoding scheme. It involved shifting each of the letters in the dispatch from 1 to 25 characters to the right. Here is how it worked:

Caesar would write his orders in plain Latin (the *plaintext*) and then give it to his coding clerk. Depending on the day of the month, the clerk would apply the daily shift value or "key value" (all of Caesar's legions knew the daily key values for each day of the month) to each letter in the plaintext. The clerk would then re-write the orders into the encoded version called the *ciphertext*.

Suppose the message Caesar wants to send to the Ninth Legion is "ATTACK AT DAWN".
Here are the letters of the alphabet: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

It is the fifth day of the month, so let's suppose the daily shift key value is 5. The coding clerk will shift each letter of the message 5 letters to the right in the following way:
'A' is enciphered to 'F'
'T' is enciphered to 'Y'
'T' is enciphered to 'Y'
'A' is enciphered to 'F'
'C' is enciphered to 'H'
'K' is enciphered to 'P

'A' is enciphered to 'F'
'T' is enciphered to 'Y'

'D' is enciphered to 'I'
'A' is enciphered to 'F'
'W' is enciphered to 'B' ( we wrap around back to the start of the alphabet if we go past 'Z')
'N' is enciphered to 'S'

So, if we put the plaintext over the ciphertext, we get this:
Plaintext: ATTACK AT DAWN
Ciphertext: FYYFHP F Y I F B S

Now, if a courier was captured by the enemy, Caesar's encoded orders would be unreadable. The courier did not know the daily key value or how the encoding scheme worked, so even if the enemy tortured him, he could not reveal anything.

It was not until about 800 years later that an Arab mathematician named Al-Khindi developed the concept of *frequency analysis* and was able to apply it to deciphering such encoded messages. In a nutshell, any language based on alphabetic characters uses some letters much more frequently than others. For example, in English the most commonly used letter is the 'E', followed by 'T', 'A','O', 'I', 'N', 'S', and 'R' (www.letterfrequency.org). The least frequently used letter is 'Z'. By counting up the frequency of letters appearing in some ciphertext, code breakers could make some good guesses as to which letter was an 'E', which was a 'T', etc. Then they would start

substituting in their guesses to see if recognizable words started to appear. Once they got a few words, it didn't take long to decode the message.

There is an excellent series of short videos from the Khan Academy that detail how the Caesar shift cipher works. Their URL's are below. Take a few minutes and go view them now before reading any further.
https://www.khanacademy.org/computing/computer-science/cryptography/crypt/v/caesar-cipher

### Requirements: The Caesar Shift Application

For the first part, you will code an app called *Your_Initials_Regular_Caesar* that will allow the user to either encipher some plaintext into a coded message, or read an enciphered message and decode it into plaintext. The use case for this program is as follows:

- Program gives a short explanation that it will either encipher or decipher a message entered by the user.
- Program asks user if they want to encipher or decipher a message by entering a 1 to encipher or a 2 to decipher. Validate this input, either in the main(), or write a method to do this.
- If enciphering:
    - program prompts user to enter the *plaintext* message and press enter when finished. This message is stored as a String of UPPER CASE CHARACTERS. Use a method here to read in the characters, convert them to UPPER CASE, and return the string.
    - Program then calls a method that asks user to enter a keyword of only letters, without spaces or punctuation. This keyword input needs to be validated in the method. If user does enter any spaces, punctuation marks, or numerals, the program should reject the keyword and ask the user to re-enter it, and keep doing this until a valid keyword is entered. The method returns the valid keyword as a string.
    - Program then calls another method to generate a shift value and passes it the validated keyword. This shift value is calculated by converting each character in the keyword to its UPPER CASE version, and then adding up the ASCII values of all the characters. Then, do a modulus division of this total by 26 to get a shift value ranging from 0 to 25. If a zero value is returned because the total was actually a multiple of 26, then add 1 the shift value so that we actually do encipher the plaintext (a shift of zero means the plaintext and the ciphertext would be identical, which would not be good message security!). This method returns the shift value as an int.
    - Program then calls another method to encipher the plaintext. This methods takes as arguments the plaintext string and the shift value generated by the keyword validation method. This shift value is then applied to each character in the plaintext to produce another String that will be the *ciphertext*, or encoded message. The method returns this ciphertext string.
    - The plaintext and ciphertext messages are then displayed on separate lines in the console window so that the user can see how the message has been encoded.
    - Program then asks user to enter a file name with a '.txt' extension so that the ciphertext can be written to a file. This method takes two string arguments, the file name and the ciphertext string, and uses a PrintWriter object to write it to a file in your Eclipse project directory.

- If deciphering:
    - Program asks if user will be typing in the ciphertext or entering it from a file.
    - If typing it in:
        - User enters the ciphertext and this is stored by the program as UPPER CASE characters. Use the same method that you used to read in a plaintext message to be encoded.

        - Program asks user to enter the keyword. Use your keyword validation method again to validate it.
        - Use your method to take the keyword and calculate the shift to apply to the ciphertext.
        - Write another method that will properly apply the shift value to the ciphertext and decipher each letter to its plaintext value. The plaintext characters are used to build a new String that will be the plaintext message. The method returns this plaintext string
        - Display the original ciphertext and the deciphered plaintext in the console window.
    - If reading from a file:
        - Program asks user for the name of the file, such as "message.txt".
        - Write a method that will take the file name as the argument and build a Scanner object that will read the contents of the file. This method will return the contents of the file as a string of which represents the ciphertext to be decoded.
        - Program asks user to enter the keyword. Validate this using your method..
        - Program then calls method to calculate the shift value and passes it the validated keyword.
        - Use the decipher method to apply the shift value to the ciphertext and decipher each letter to its plaintext value. Build a new String using the deciphered characters and return this string
        - Display the original ciphertext and the deciphered plaintext in the console window.

**Use Methods and Minimize Code in the main() Method**

We want you to try to minimize the amount of code in your main() method, and do most of the work in various methods mentioned above. To review, the required methods that you will need to write include:

- If the user is using keyboard entry, a method to read in the user's plaintext or ciphertext message from the keyboard that will convert it to UPPER CASE characters and then return this as a String that will be passed as an argument to either an encipher() method or a decipher() method.
- A method to ask the user to enter a keyword. The method must validate the keyword. If the user passes in an invalid keyword, the method tells them it is invalid and asks them to enter another keyword. Once a valid keyword is obtained, the method returns it as a String.
- A method to generate the shift value to be applied to either encipher the plaintext or decipher the ciphertext. This method will accept the valid keyword and calculate its value using the algorithm described on page 2. It will return this shift value as an int.
- A method to encipher plaintext into ciphertext. This method will take as arguments the plaintext string and the shift value. It will then apply the shift value to each character in the plaintext string and build a new string that is the ciphertext string. NOTE: spaces and punctuation marks will not have the shift value applied to them. Only apply the shift value to the letters. Leave the spaces and punctuation marks as they are. The method returns the ciphertext string.
- A method to write the ciphertext to a file in your local Eclipse project directory. This method will take two arguments of type String. The first argument will be the name of the file with the extension for a text file ('.txt' extension), such as "secrectMessage.txt". The second argument will be the ciphertext string. The method will instantiate and use a PrintWriter object to write the ciphertext to a file.
- For deciphering, if the user specifies that the ciphertext will be read in from a file, you will need to ask the user to enter the file name and extension (i.e. "secretMessage.txt") and save this as a string. Then you need to write a method that will take this file name as an argument. The method will build a Scanner

object using the file name so that you can read the file's contents. The file's contents must be read and converted to a String of UPPER CASE letters. This string is then returned as the ciphertext, which will later be passed to the decipher() method.

- A method to decipher ciphertext into plaintext. This method will take as arguments the ciphertext string and the shift value. It will then apply the shift value to each letter character in the ciphertext and build a new string that is the plaintext string. Again, do not apply the shift to any spaces or punctuation marks. The method returns the plaintext string.
- NOTE: all methods should be in a separate helper class named ***Your_Initials*_Project_Methods**.java. Make sure that you include this file when you submit your project.

Below are some sample outputs from a prototype:

```
Doing an ENCIPHER
Prototype for a Caesar Shift Encoding Application.


This program will help you encipher a message or decipher a coded message.

If enciphering, enter 1 and press ENTER:
If deciphering, enter 2 and press ENTER:
Enter 1 or 2: 1

Enter the plaintext message you wish to encipher: Attack at dawn!
Enter a keyword of only letters, with no digits, spaces, or punctuation marks: Wednesday

The plaintext and enciphered text are as follows:
ATTACK AT DAWN!
BUUBDL BU EBXO!

Enciphered text will now be written to a text file.
Enter a file name followed by the extension '.txt' (example: secret.txt): secret.txt
Ciphertext has been written to local directory as secret.txt

Doing a DECIPHER:

Prototype for a Caesar Shift Encoding Application.

This program will help you encipher a message or decipher a coded message.

If enciphering, enter 1 and press ENTER:
If deciphering, enter 2 and press ENTER:
Enter 1 or 2: 2
We will be deciphering a message.

If ciphertext will be entered via keyboard, enter 1.
If ciphertext will be entered from a file, enter 2.
Enter 1 or 2: 1

Enter the ciphertext message you wish to decipher: buubdl bu ebxo!
Enter a keyword of only letters, with no digits, spaces, or punctuation marks: Wednesday

The ciphertext and deciphered plaintext are as follows:
BUUBDL BU EBXO!
ATTACK AT DAWN!
```

**Using a file as the source of the ciphertext…**

```
Prototype for a Caesar Shift Encoding Application.

This program will help you encipher a message or decipher a coded message.

If enciphering, enter 1 and press ENTER:
If deciphering, enter 2 and press ENTER:
Enter 1 or 2: 2
We will be deciphering a message.

If ciphertext will be entered via keyboard, enter 1.
If ciphertext will be entered from a file, enter 2.
Enter 1 or 2: 2
Enter the name of the file containing the ciphertext: secret.txt
Enter a keyword of only letters, with no digits, spaces, or punctuation marks: Wednesday

The ciphertext and deciphered plaintext are as follows:
BUUBDL BU EBXO!
ATTACK AT DAWN!
```

**Hot and Spicy Version: (4 bonus marks available if you are up to the challenge)**
**The Super Caesar Cipher**

In order to make it more difficult to decode, what we'll call the Super Caesar cipher was developed. In this enhanced version of the Caesar cipher, each letter in the plaintext is shifted by a different shift value based on some keyword, which could be an actual word or just a random selection of letters.

To demonstrate, let's pick a short keyword for a simple example and use the word BAND, which has four letters. Now, let's assign a numeric value to each letter in the keyword using a ZERO-BASED INDEXING scheme like we do with arrays. So, the letter A would be 0 (zero), B would be 1, C would be 2, D would be 3, E would be 4, etc.

Here are the letters of the alphabet and their corresponding values using ZERO-BASED INDEXING:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

The value of each letter in our keyword BAND is assigned its corresponding numeric value so it would look like this:
B A N D
1 0 13 3

These corresponding letter values tell us how far to shift each letter of our plaintext.
Suppose our plaintext message was "MEET AT ZOO".
Using our keyword BAND, the B has a value of 1, so the first letter of our plaintext will be shifted to the right by 1 character. Therefore, the M of MEET becomes N.
MEET
N
For the second letter of MEET, E, we'll use the second letter of BAND, the A, which has a value of 0, so the E is shifted 0 letters to the right and just stays as an E. (NOTE: a shift of zero is okay in this Super Caesar encoding scheme for a single letter, as each letter gets shifted by a different amount).
MEET
N E

For the third letter of MEET, another E, we use the third letter of BAND, the N, which has a value of 13, so we shift the E 13 letters to the right to get an R.
MEET
N E R

For the fourth letter of MEET, T, we use the fourth letter of BAND, the D, which has a value of 3, so the T is shifted 3 letters to become a W.
M E E T
N E R W

Now we have cycled through our keyword BAND once, so we go back to the start of it and use the B again to encipher the A of AT. We shift the A 1 letter to the right, so it becomes a B
M E E T  A T
N E R W  B

For the second letter of AT, the T, we use the A of BAND and apply a shift of 0(zero) letters to the right and just leave it as a T.
M E E T  A T
N E R W  B T

For the Z in ZOO, we use the third letter of BAND, N, and apply a shift of 13. Since we are at the right end of the alphabet, we wrap around back to the start and count 13 characters to arrive at the M.
NOTE: Don't be fooled by the index numbers and use the N! Remember, we started indexing at 0, so starting a 0 and counting 13 spaces gets you to the index 12, which the letter M.
M E E T  A T  Z O O
N E R W  B T  M

For the first O in ZOO, we use the fourth letter of BAND, D, and apply a shift of 3 characters to get an R.
M E E T  A T  Z O O
N E R W  B T  M R

We have now cycled through our keyword BAND twice, so we go back to its first letter B and use it again for the second O in ZOO. We apply a shift of 1 letter to the O and get the P. So, our final ciphertext looks like this:
M E E T  A T  Z O O
N E R W  B T  M R P

If you are interested in attempting the spicy version for a few bonus marks, then code it as a separate program called *Your_Intials_Super_Caesar*.java. Most of your methods will stay the same, but you will need to adjust your encipher() and decipher() methods to utilize the Super Caesar shift.

If you do try the spicy version make sure you submit BOTH programs (regular and super versions), to the drop box as one zipped file.

## Suggestions:

1) Do the pseudo-code for this program first and include it in a comment block at the top of your program, just below your program documentation header. You can then copy and paste the portions of the pseudo-code to use as comments into each section of your code, but leave the original pseudo-code intact at the top of your code.

   This pseudo-code block may be more than one page long if you feel it is necessary, but the most important thing is that it clearly outlines the steps you will follow as you write your program. Also, if your program does not compile, or it crashes during testing, marks will be deducted here.

   Build the program one method at a time. Build a method, get it working, and test it to make sure it does what it is supposed to do.

2) All naming conventions for variables, constants, and methods must be followed.

3) Instructions to user must be clear and concise.

### Submitting your work:

Submit your Java SOURCE FILES to the Project dropbox in FOL by the deadline indicated on page 1 of this document. If you teacher prefers a zipped .java file, please zip it up.

### Submit your project on time!

Code submissions to the FOL drop box must be made on time! Late projects will be subject to late penalties of 10% per day up to a maximum of 5 days. After 5 days a mark of zero will be given.

### Submit your own work!

It is considered cheating to submit work done by another student or from another source as your own work. This is called plagiarism. Helping another student cheat by letting them copy your source code files is called abetting plagiarism. Both are considered to be academic offences.

YOU MUST WRITE YOUR OWN CODE!

Students are encouraged to share ideas and to work together on practice exercises, and you can help someone else to fix a bug in their code, but any code or documentation prepared for your project must be done by you.

Penalties for committing plagiarism, or helping another student plagiarize by sharing source code with them, include a zero grade and an academic offence being filed against the student or students involved. All submissions will be analyzed using plagiarism detection software especially designed to analyze Java code.

**Marking Key**

| | Marks Available | Marks Assigned |
|---|---|---|
| STYLE: Appropriate pseudo-code statements are done.<br>Documentation header is complete.<br>Program compiles without errors and runs without crashing.<br>Naming conventions followed, variable, constant, and method names are all representative of what each does. | 2 | |
| **Part One: The Regular Caesar Cipher** | | |
| App provides user instructions and asks user if they want to do a coding or decoding operation. User choice of 1 or 2 is validated. | 1 | |
| Has a getText() method to read in either plaintext or ciphertext and convert it to UPPER CASE characters and return it as a string. | 2 | |
| Has a keyword() method that prompts user to enter a keyword. Method validates the keyword and returns it as a string. | 2 | |
| Has a getShift() method that takes a valid keyword as an argument, calculates a shift value ranging from 1 to 25, and returns this value as an int. | 2 | |
| Has an encipher() method that takes a plaintext string and a shift value as arguments and applies the shift to produce a ciphertext String.<br>The ciphertext string is returned as a string | 3 | |
| Has a writeFile() method that accepts a file name with an extension and the ciphertext string and writes the ciphertext to a file in your Eclipse project folder. | 2 | |
| Has a decipher() method that takes a ciphertext string and a shift value as arguments and applies the shift to produce a plaintext String. The plaintext string is returned as a string | 3 | |
| Has a readFile() method that reads a ciphertext file and returns the file contents as a String of UPPER CASE letters, which can then be passed to the decipher()method. | 2 | |
| Output: program prints out both plaintext and ciphertext strings so that user can see the effect of whatever operation was performed | 1 | |
| TOTAL | 20 | |
| **Part 2: The Super Caesar Shift Program( 4 bonus marks available)** | | |
| In a separate program, modify the Regular_Caesar program so that each letter of plaintext will be encoded using a different shift based on the letters of the keyword. | 4 | |
| TOTAL | | |