CSE 1322L - Assignment 4 (Spring 2026)

Introduction

Whether you are a general in the ancient world, a spy during the Cold War, or a regular citizen in the modern day, at some point you may have needed to send a message to another person. However, you find yourself faced with a few requirements:

- No one besides the recipient must read the message
- You cannot deliver the message personally

If your messenger is conspiring against you or gets captured by your adversary, how can you be sure that your opponent won't use the contents of your message against you? The answer is by using encryption, which will transform the contents of your message such that no one else is able to read it except for you and your intended recipient.

To perform encryption, one must use a cipher, which is a pair of algorithms to encrypt and decrypt information. Plaintext is the technical term for text which is not encrypted. Once plaintext is encrypted (or enciphered) it becomes ciphertext, whereas decrypting (or deciphering) the ciphertext yields back the plaintext. As long as you and your recipient agree on a cipher that your opponent doesn't understand, you can both communicate knowing that, if any of the messages leak while in transit, its contents cannot be read.

In this assignment, you will write an abstract class called "Mendec", which is short for "Message ENcryptor/DECryptor". A mendec just keeps track of some basic information which is common to all types of mendec; its subclasses will implement the actual ciphers.

Caesar Cipher

The creation of this cipher is usually attributed to Julius Caesar, a Roman general from the early years of the Common Era. This cipher works by applying a specific constant shift (here called "offset") to the contents of the plaintext.

If the cipher has an offset of 1, every "a" becomes "b", every "b" becomes "c", etc., with every "z" wrapping around and becoming "a". The table below shows the Caesar cipher with a shift of 1:

Original letter	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
Cipher letter	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	A

With a shift of 2, every "a" becomes "c", every "b" becomes "d", and so on. Once again, the alphabet wraps around at the ends, with every "y" becoming "a" and every "z" becoming "b":

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

The cipher also accepts negative shifts, which work in the opposite direction: a shift of -1 means every "a" becomes "z", every "b" becomes "a", etc:

Original letter	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
Cipher letter	Z	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

As an example, if we use a shift of 3 on the sentence "`Dreams and Nightmares`", we get the following:

```
Dreams and Nightmares → Guhdpv dqg Qljkwpduhv
```

As a side note: all shifts on the English alphabet which are multiples of 26 do nothing: they just shift the original letter back onto itself.

To make this cipher more interesting, we will encode other characters besides the letters of the English alphabet. Below you can see all the groups of characters our cipher must support, as well as their order within the group:

Character Group Table	
Group	**Characters**
1	! " # $ % & ' ( ) * + , - . /
2	0 1 2 3 4 5 6 7 8 9
3	: ; < = > ? @
4	ABCDEFGHIJKLMNOPQRSTUVWXYZ
5	[\ ] ^ _ `
6	abcdefghijklmnopqrstuvwxyz
7	{ | } ~

Each group must wrap around itself on both ends. As an example, trying to encrypt the character ";" with a shift of 3 turns it into ">", whereas a shift of -3 turns it into "?". Trying to encrypt the sentence:

```
I will accept $200, and no less than that!
```

with an offset of 3 yields the following ciphertext:

```
L zloo dffhsw '533/ dqg qr ohvv wkdq wkdw$
```

Decrypting a message using the Caesar cipher is simple: use the additive inverse of the original offset: if, for example, the plaintext was encrypted using an offset of 3, the ciphertext must be "encrypted" using an offset of -3, which will decrypt the ciphertext back into its original plaintext.

Vigenère Cipher

The creation of this cipher is usually attributed to Blaise de Vigenère, a French diplomat from the 1500s. It works by replacing the contents of the plaintext using a key, which is itself also a piece of text. Our Vigenère cipher will use the key to figure out how much of a shift must be applied to each character in the plaintext.

Given the plaintext "`Expect tomorrow to be hot.`" and the key "`abled`", we must first align the key under the plaintext:

```
Expect tomorrow to be hot.
abled
```

As we can see, the key is shorter than the plaintext. In such a case, we repeat the key until it is as long as the plaintext, stopping once we have enough characters to cover the entire plaintext.

```
Expect tomorrow to be hot.
abledabledabledabledableda
```

The characters in the key are then used to determine by how much each character in the plaintext needs to be shifted. The letter "a" means a shift of 0, the letter "b" means a shift of 1, and so on. In our example, since "E" from "Expect" is being shifted using "a" (which is a shift of 0), the original "E" stays as it is. The "x" is being shifted using "b", which is a shift of 1, so "x" becomes "y".

```
Expect tomorrow to be hot. → Ey
abledabledabledabledableda
```

The next letter, "p", needs to be shifted using "L", which is a shift of 11. This would shift "p" beyond "z", which isn't possible. Instead, whenever a shift would place a letter past the alphabet, we wrap around to the start of the alphabet. Thus, shifting "p" using "L" gives us "a":

```
Expect tomorrow to be hot. → Eya
abledabledabledabledableda
```

If we encrypt the rest of the message, we get the following:

```
Expect tomorrow to be hot. → Eyaift esposcsz uz ee ssw.
abledabledabledabledableda
```

Decrypting a Vigenère ciphertext works the same way decrypting a Caesar ciphertext: we must use the additive inverse of the offset provided by each character in the key: every "b" in the key, which was originally an offset of 1, now becomes an offset of -1; every "c" becomes an offset of -2, and so on.

NOTE: Our Vigenère cipher will only handle letters of the English alphabet. All other characters must be left unchanged.

Transposition Cipher

A transposition cipher isn't a specific cipher, but rather a type of cipher. The Caesar and Vigenère ciphers are substitution ciphers, meaning that they replace the contents of the plaintext with different content, which then becomes the ciphertext. Conversely, transposition ciphers turn plaintext into ciphertext by scrambling the plaintext.

The specific transposition cipher we will implement requires 2 pieces of information: the cipher's width and the plaintext. Given the plaintext below:

```
Let's dance tonight!
```

If we use a width of 5, we must first write the characters as normal (from left to right, top to bottom), but we must not allow a line to have more characters than the cipher's width. In our example, each row of the plaintext must have exactly 5 characters:

```
"Let's"
"_danc"
"e ton"
"ight!"
```

Finally, to extract the ciphertext, we must read the sentence above column by column, i.e.: from top to bottom, then left to right:

```
L eied gtath'notscn!
```

In the examples above, the contents of column 1, 3, and 5 have been underlined for clarity.

Note that, in some circumstances, the message may not have enough letters to fill the last row. In this situation, pad out the extra space with periods ("."). Given the plaintext "`Let's dance!`" and a width of 5, the last row would be short 3 characters. In this case, we pad it with 3 periods:

```
"Let's"
" danc"
"e!..."
```

Which then becomes the following ciphertext:

```
L eed!ta.'n.sc.
```

To decrypt the ciphertext back into plaintext, write the ciphertext from top to bottom, left to right, and then read it as normal (left to right, top to bottom).

This cipher can be easily implemented using a 2D array of characters, where the second dimension is the cipher's width and the first dimension has however many rows are necessary to fit the original plaintext. Our "`Let's dance!`" plaintext needs enough space to hold 12 characters. Given that the width of the cipher is 5, then the 2D array to hold the message would have the following syntax:

```
char[][] text = new char[3][5]; // 15 indices total
```

If we lay out the array into a table, its contents would be:

L	e	t	'	s
	d	a	n	c
e	!	.	.	.

Requirements

The following must be present in your project:

- A class called "Mendec". <u>This class must be abstract</u> and it must have the following properties:
    - 2 string fields, called "name" and "description"
        - The "name" is given by the user, so they can easily remember what the mendec is for
        - The "description" has a technical explanation of the mendec. It's set by the program
    - An integer field called "id"
    - A static integer field called "nextId" initialized at 0
    - All fields (except the static field) must have getters
    - **Mendec(String, String)**: assigns the arguments to the class fields, assigns "nextId" to "id", then increments "nextId" by 1
    - **abstract String decrypt(String)**: will be used to implement decryption
    - **abstract String encrypt(String)**: will be used to implement encryption
- A class called "CaesarCipher", which <u>is a subclass of Mendec</u>. It must have the following properties:
    - A integer field called "offset"
    - **CaesarCipher(String, int)**: Calls the superclass constructor using the String argument and the string "`A Caesar Cypher with an offset of {offset}.`". The integer argument is used to set the object's field.
    - **String encrypt(String)**: Encrypts the argument using the Caesar Cipher algorithm as described above, returning the ciphertext. Your implementation needs to be able to encrypt all the characters in the Character Group Table. All other characters must be left as is.
    - **String decrypt(String)**: Decrypts the argument using the Caesar Cipher algorithm as described above, returning the plaintext. Your implementation needs to be able to decrypt all the characters in the Character Group Table. All other characters must be left as is.
    - **String toString()**: returns the following string:

      ```
      #{id}: {description} - Caesar Cipher
      ```
- A class called VigenereCipher, which <u>is a sublclass of Mendec</u>. It must have the following properties:
    - A string field called "key"
    - **VigenereCipher(String, String)**: Calls the superclass constructor using the first argument and the string "`A Vigenere Cipher with key '{key}'.`". The last string argument is converted to upper case and then used to set the object's field. <u>You can assume the second string argument will never be empty and only contains letters of the English alphabet</u>.
    - **String encrypt(String)**: Encrypts the argument using the Vigenere Cipher algorithm as described above, returning the ciphertext. Remember that only

characters in the English alphabet must be encrypted; all other character must be left as-is.
    - **String decrypt(String)**: Decrypts the argument using the Vigenere Cipher algorithm as described above, returning the plaintext. Remember that only characters in the English alphabet must be decrypted; all other character must be left as-is.
    - **String toString()**: returns the following string:
      `#{id}: {description} - Vigenere Cipher`
- A class called TransposerCipher, which is a subclass of Mendec. It must have the following properties:
    - An integer field called "width"
    - **TransposerCipher(String, int)**: Calls the superclass constructor using the first argument and the string "`A Transposer Cipher with witdh {witdh}.`". The integer argument is then used to set the object's field. You can assume the integer argument is always be positive.
    - **String encrypt(String)**: Encrypts the argument using the Transposer Cipher algorithm as described above, returning the ciphertext. You must use a 2D array of characters to implement this cipher.
    - **String decrypt(String)**: Decrypts the argument using the Transposer Cipher algorithm as described above, returning the plaintext. You must use a 2D array of characters to implement this cipher.
    - **String toString()**: returns the following string:
      `#{id}: {description} - Transposer Cipher`
- In your driver's main(), create an arraylist of Mendecs. Then, in a loop, implement the menu options below:
    - **Encrypt text**: Prompt the user for a Mendec id. If the id doesn't exist, print an error message. Otherwise, prompt the user for a message to encrypt, pass said message to the encrypt() method of the object whose id matches what the user entered, and print out the returned ciphertext.
    - **Decrypt text**: Prompt the user for a Mendec id. If the id doesn't exist, print an error message. Otherwise, prompt the user for a message to decrypt, pass said message to the decrypt() method of the object whose id matches what the user entered, and print out the returned plaintext.
    - **Add mendec**: Ask the user which of the three available encryption types they want. Prompt the user for the necessary information to create an object of that type, then store the object in the arraylist.
    - **Show mendecs:** Ask the user if they would like to see all mendecs or only a specific type of mendec (and, if so, which type). Print the toString() of all the objects in the arraylist whose instance matches the mendec type the user picked.
    - **Describe mendec:** Prompt the user for a Mendec id. If the id doesn't exist, print an error message. Otherwise, print the toString() and description of the object whose id matches what the user entered.

    - **Remove mendec**: Prompt the user for a Mendec id. If the id doesn't exist, <u>print an error message</u>. Otherwise, remove the object whose id matches what the user entered.
    - **Quit**: Terminates the program

Deliverables

- Assignment3.java (driver)
- Mendec.java
- CaesarCipher.java
- VigenereCipher.java
- TransposerCipher.java
- UML.pdf (uml diagram)

Considerations

- You will get partial credit for partial work, as long as the rubric permits it.
- Despite what deliverables say, all of your classes can be submitted in a single file.
- Do not use any of the ciphers described in this write-up to send important secret messages. They can all easily be broken by brute-force.
- It is possible to implement our Transposition Cipher without having to pad the last row.
- Don't forget the UML diagram!
- There is a reason why the table of character groups under Caesar Cipher is arranged the way it is.
- Don't forget that, from the computer's perspective, a char is just an integer. Performing math using a char and saving the result back into the char will give you a different char:

```
char myChar = 'a';
myChar += 2;
System.out.println(myChar); // prints 'c'
```

- Remember that you can check the instance of an object using the "`instanceof`" operator

Sample Output (user input in red)

```
[Message Encryptor/Decryptor]
1. Encrypt message
2. Decrypt message
3. Add mendec
4. Show mendecs
5. Describe mendec
6. Remove mendec
0. Quit
Enter option: 3

1. Caesar Cipher
2. Vigenere Cipher
3. Transposition Cipher
Use which cipher? 1
Enter mendec name: Secret club
Enter offset: -5
Mendec added.

1. Encrypt message
2. Decrypt message
3. Add mendec
4. Show mendecs
5. Describe mendec
6. Remove mendec
0. Quit
Enter option: 3

1. Caesar Cipher
2. Vigenere Cipher
3. Transposition Cipher
Use which cipher? 2
Enter mendec name: Younger brother
Enter key: fullmoon
Mendec added.

1. Encrypt message
2. Decrypt message
3. Add mendec
4. Show mendecs
5. Describe mendec
6. Remove mendec
```

```
0. Quit
Enter option: 3

1. Caesar Cipher
2. Vigenere Cipher
3. Transposition Cipher
Use which cipher? 3
Enter mendec name: Girlfriend
Enter width: 7
Mendec added.

1. Encrypt message
2. Decrypt message
3. Add mendec
4. Show mendecs
5. Describe mendec
6. Remove mendec
0. Quit
Enter option: 3

1. Caesar Cipher
2. Vigenere Cipher
3. Transposition Cipher
Use which cipher? 1
Enter mendec name: Other girlfriend
Enter offset: 50
Mendec added.

1. Encrypt message
2. Decrypt message
3. Add mendec
4. Show mendecs
5. Describe mendec
6. Remove mendec
0. Quit
Enter option: 4

Show which mendecs? Options are:
ALL
CAE for Caesar Cipher
VIG for Viginere Cipher
TRA for Transposer Cipher
```

```
Enter option: ALL

#0: 'Secret club' - Caesar Cipher
#1: 'Younger brother' - Vigenere Cipher
#2: 'Girlfriend' - Transposer Cipher
#3: 'Other girlfriend' - Caesar Cipher

1. Encrypt message
2. Decrypt message
3. Add mendec
4. Show mendecs
5. Describe mendec
6. Remove mendec
0. Quit
Enter option: 4

Show which mendecs? Options are:
ALL
CAE for Caesar Cipher
VIG for Viginere Cipher
TRA for Transposer Cipher
Enter option: CAE

#0: 'Secret club' - Caesar Cipher
#3: 'Other girlfriend' - Caesar Cipher

1. Encrypt message
2. Decrypt message
3. Add mendec
4. Show mendecs
5. Describe mendec
6. Remove mendec
0. Quit
Enter option: 5

Enter mendec ID: 1

#1: 'Younger brother' - Vigenere Cipher
A Vigenere Cipher with key 'fullmoon'.

1. Encrypt message
2. Decrypt message
```

```
3. Add mendec
4. Show mendecs
5. Describe mendec
6. Remove mendec
0. Quit
Enter option: 1

Enter mendec ID: 1

Enter plaintext: If Alice calls, tell her I'm with you.

Your ciphertext is 'Nz Lxwqr wlwxg, yyww vse C'x kwgm jzg.'

1. Encrypt message
2. Decrypt message
3. Add mendec
4. Show mendecs
5. Describe mendec
6. Remove mendec
0. Quit
Enter option: 2

Enter mendec ID: 3

Enter ciphertext: G uml,r zc dpcc slrgj 7;15NK$ Jcr,q ayraf rfc 8nk
kmtgc glqrcyb$

Your plaintext is 'I won't be free until 7:15PM. Let's catch the 8pm
movie instead.'

1. Encrypt message
2. Decrypt message
3. Add mendec
4. Show mendecs
5. Describe mendec
6. Remove mendec
0. Quit
Enter option: 1

Enter mendec ID: 3

Enter plaintext: Usual spot, usual place. Wear the blue one this time.
```

```
Your ciphertext is 'Sqsyj qnmr" sqsyj njyac$ Ucyp rfc zjsc mlc rfgq
rgkc$'

1. Encrypt message
2. Decrypt message
3. Add mendec
4. Show mendecs
5. Describe mendec
6. Remove mendec
0. Quit
Enter option: 2

Enter mendec ID: 1

Enter ciphertext: Xbp mzfrfxj wbcjx hsqfs dif'ds ttcyr.

Your plaintext is 'She already knows where you're going.'

1. Encrypt message
2. Decrypt message
3. Add mendec
4. Show mendecs
5. Describe mendec
6. Remove mendec
0. Quit
Enter option: 2

Enter mendec ID: 2

Enter ciphertext: Nomnr ttipene.euehsavt   ndetreaaYai  mtrcgolntCay
tausvohs.c i'oi r .

Your plaintext is 'Never contact me again. You're also uninvited to
the Christmas party..'

1. Encrypt message
2. Decrypt message
3. Add mendec
4. Show mendecs
5. Describe mendec
6. Remove mendec
```

```
0. Quit
Enter option: 6

Enter mendec ID: 2

Mendec removed.

1. Encrypt message
2. Decrypt message
3. Add mendec
4. Show mendecs
5. Describe mendec
6. Remove mendec
0. Quit
Enter option: 1

Enter mendec ID: 0

Enter plaintext: Going forward, I am now free on the weekends!

Your ciphertext is 'Bjdib ajmrvmy' D vh ijr amzz ji ocz rzzfziyn+'

1. Encrypt message
2. Decrypt message
3. Add mendec
4. Show mendecs
5. Describe mendec
6. Remove mendec
0. Quit
Enter option: 4

Show which mendecs? Options are:
ALL
CAE for Caesar Cipher
VIG for Viginere Cipher
TRA for Transposer Cipher
Enter option: ALL

#0: 'Secret club' - Caesar Cipher
#1: 'Younger brother' - Vigenere Cipher
#3: 'Other girlfriend' - Caesar Cipher
```

```
1. Encrypt message
2. Decrypt message
3. Add mendec
4. Show mendecs
5. Describe mendec
6. Remove mendec
0. Quit
Enter option: 0

Shutting off...
```