

Assignment 1

COMP 250 - Winter 2018

Due date : February 13, 2018

1 General Instructions (read carefully)

- Office hours are displayed on the calendar on the webpage of the course.
- You are provided some starter code that you should fill in as requested. **Add your code only where you are instructed to do so.** You can add some helper methods. Do not modify the code in any other way and in particular : do not change the methods or constructors that are already given to you, do not add new packages and do not touch the method headers. **Any failure to comply with these rules will give you an automatic 0.**
- **Submission instructions**
 - Assignment is due on February 13 11:59 PM.
 - Late assignments will be accepted up to 3 days late and will be penalized by 20 points per day. Note that submitting one minute late is the same as submitting 23 hours late.
 - Don't worry if you realize that you made a mistake after you submitted : you can submit multiple times but only the latest submission will be evaluated. We encourage you to submit a first version a few days before the deadline (computer crashes do happen and myCourses may be overloaded during rush hours).
 - Submit only the file `Message.java` to the myCourses Assignment 1 folder.
- The starter code includes a tester class. If your code fails those tests, it means that there is a mistake somewhere. Even if your code passes those tests, it may still contain some errors. We will test your code on a more challenging set of examples. We therefore highly encourage you to

modify that tester class, expand it and share it with other students on the myCourses discussion board. Do not include it in your submission. However, your code will only be tested on valid inputs, for instance, 10200011 is not a valid input if the method requires a number in base 2. The tester file is only here to help you, the reference is the subject of the assignment.

- You will automatically get 0 if your code does not compile.
- **Failure to comply with any of those rules will be penalized.** If anything is unclear, it is up to you to clarify it by asking either directly a TA during office hours, or on the discussion board on myCourses.

2 Cipher / Decipher

You have probably already tried to whisper things to your neighbour so that people around you would not hear you. Well, other people have been doing that for a pretty long time and they came up with a clever solution : cryptography. The idea is that people may hear you or find the piece of paper where you wrote your message, so it is not enough to hide it, you should in fact change its form (cipher or encrypt the message) and the person who the message is for should be the only one able to get the original form back (decipher or decrypt).

Some people found some very ingenious ways of hiding messages though : during the antiquity, Histiaeus shaved the head of his most-trusted slave, tattooed a message (asking to revolt against the Persians) on his bald head, waited for the hair to grow back again and then sent the slave to Aristagoras. This may work when there is plenty of time, but it is very inconvenient in many ways.

In this assignment you will implement several classical ciphers. They all rely on the fact that the people trying to communicate share a key (hence it is also called symmetric ciphers). You will only need to modify the file `Message.java`. It implements the class `Message` that has two fields : a `String message` corresponding to the message and an `int` field corresponding to the length of the message.

2.1 First function

Question 1. (10 pts) First, you will need to implement the method `makeValid` : it will remove all characters that are not letters from 'a' to 'z' and change upper case letters to their corresponding lower case letters.

This is important because leaving spaces, punctuation or upper case letters in a message will give information to the people who want to understand your message but should not have access to it (this is called cryptanalysis).

2.2 Caesar Cipher

The Caesar cipher is one of the easiest and oldest ciphers. In this case the key is a number n . To encrypt a message, you shift all letters by n , so for instance if the key is 2,

```

a becomes c
b becomes d
:
y becomes a
z becomes b

```

As you can see, the encrypted message still contains only letters as once we reach **z**, we start over again from **a**. With that key, the message **goodluck** becomes **iqqfnwem** and the message **zebra** becomes **bgdtc**.

Question 2. (15 pts) Fill in the code for **caesarCipher**. Be careful, the key could be any signed integer and in java, $36\%26 = 10$ but $(-36)\%26 = -10$!

To decipher a message, you only have to shift back the letters, we have implemented that method for you. You can see that this cipher is not very robust : if you want to read a message that is not addressed to you, you only have 25 keys to test. But there is a cleverer way to do this. In many languages like English, there are letters that appear more often than others (**e** in English). If the message is long enough, it becomes very simple to find the key : first look at which letter appears more often. This letter is the most likely to correspond to letter **e** in the original message, so you can decide on what is the most likely key and decipher with the key you have just found.

Question 3. (15 pts) Fill in the code for **caesarAnalysis**.

2.3 Vigenere Cipher

There is a variant of the Caesar cipher that was much more complicated to crack and it took three centuries to break it. Instead of being a single integer, the key is an array of integers. Let's show you how it works on an example. You want to encrypt "I love Computer Science". The corresponding **Message** has message **ilovecomputerscience**. You want to encrypt it using the key **\{ 4, 11, 4, 15, 7, 0, 13, 18 \}**. First you write down the key under the message as such

| | | | | | | | | | | | | | | | | | | | |
|---|----|---|----|---|---|----|----|---|----|---|----|---|---|----|----|---|----|---|----|
| i | l | o | v | e | c | o | m | p | u | t | e | r | s | c | i | e | n | c | e |
| 4 | 11 | 4 | 15 | 7 | 0 | 13 | 18 | 4 | 11 | 4 | 15 | 7 | 0 | 13 | 18 | 4 | 11 | 4 | 15 |

The number under the letter tells you by how much you should shift each letter. **i** with 4 gives the letter **m**, then **l** with 11 gives **w**, etc. In the end you get the message **mwsklcbetfxtyspaiygt**.

Question 4. (15 pts each) Fill in the code for **vigenereCipher** and **vigenereDecipher**.

If you look on the wikipedia page, you will see that the key is usually a word, but this works essentially in the same way and in our example, our key would be **elephant**.

You can see that the analysis of which letter is more frequent cannot be performed here. But if the message was long enough and if you knew the length of the key, you could break it into smaller messages and perform the Caesar analysis on them. And get back the key. The difficulty is now finding the length of the key.

Question 5. Optional (0 pt) You can code the Analysis of Vigenere (either knowing the length of the key or not depending on how much you want to challenge yourself)

2.4 Transposition Cipher

Both the Caesar and Vigenere ciphers are what we call substitution ciphers : you substitute a letter for an other one. There are other ciphers that change the position of the letter and not the letter itself. They are called transposition ciphers.

You are going to code an easy one and its corresponding decipher method. The key is a single integer. Let's show you how it works on an example. Consider the sentence "This course is amazing" and the key 6. First, write down the sentence in an array with 6 columns and fill in the empty boxes by a *

| | | | | | |
|---|---|---|---|---|---|
| t | h | i | s | c | o |
| u | r | s | e | i | s |
| a | m | a | z | i | n |
| g | * | * | * | * | * |

Now, instead of reading it row by row, read it column by column and you get "tuaghrom*isa*sez*cii*osn*". Note that the length of the message has changed as you have added * to it.

In order to decipher, you basically have to construct that same array and then read it row by row again. For instance, if you received a message **yadoloumnaoers*et*** that was constructed with the same key (6), first, compute the size of the array : there are 6 columns and 3 rows that you can fill one column after the other. For instance, after reading 7 letters and at the end of the sentence :

| | | | | | | | | | |
|---|---|---|--|---|---|---|---|---|---|
| y | o | u | | y | o | u | a | r | e |
| a | l | : | | a | l | m | o | s | t |
| d | o | | | d | o | n | e | * | * |

And you can now read the original message : **youarealmostdone**.

Question 6. (15 pts each) Fill in the code for `transpositionCipher` and `transpositionDecipher`.

Question 7. Optional (0 pt) You can also code the Analysis of this transposition cipher or other transposition ciphers (many more are explained on the wikipedia page).