Data Structures & Algorithms II

Assignment 2 - Substitution Ciphers / Enigma Machine

Due Date: 17th May 2020

Part 1: The Caesar Cipher

The use of codes (or *ciphers*) as a means of hiding the meaning of messages traces its roots to ancient history. The first known military use of codes was by Julius Caesar in 50 - 60 B.C. The *Caesar cipher* specified that each letter in the alphabet would be encoded using the letter three positions later in the alphabet. For example, `'a'` would be encoded as `'d'`, `'b'` would be encoded as `'e'`, `'c'` would be encoded as `'f'`, and so on. The code wraps around at the end of the alphabet, so `'x'`, `'y'` and `'z'` would be encoded as `'a'`, `'b'`, and `'c'`, respectively. The complete mapping of letters is shown below.

```
Caesar cipher:
     abcdefghijklmnopqrstuvwxyz
     ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
     defghijklmnopqrstuvwxyzabc
```

The provided `CaesarCipher` class contains a partial implementation of the Caesar cipher. The encode and decode methods can be used to encode and decode lower-case letters by shifting them three positions in the alphabet. Modify these methods so that they handle upper-case letters and non-letters as well. An upper-case letter should be encoded/ decoded just like its lower-case equivalent, producing the corresponding upper-case letter. For example, `'A'` should be encoded as `'D'`. Characters that are not letters should be left as is. For example, the encoding of a space or exclamation mark should be that same character unchanged.

Hint: various methods of the from the `Character` class such as `isUpperCase()` and `toUpperCase()` should be of use here.

Once you have tested your implementation, you may use it in combination with the provided `Encrypter` and `Decrypter` classes. These classes use a `Cipher` to encode or decode text strings or entire files.

Part 2: Substitution Ciphers

Although the Caesar cipher was effective in its time (when very few people could read at all), its simple pattern of encoding letters seems pretty obvious today. However, it can be generalized to create more effective ciphers. The Caesar cipher is an example of a *substitution cipher,* a code in which one letter of the alphabet is substituted for another. They key `"defghijklmnopqrstuvwxyzabc"` defines the letter mapping used by the Caesar cipher. Using a different key, a completely different substitution cipher is defined.

For example:

```
Mystery cipher:
     abcdefghijklmnopqrstuvwxyz
     ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
     qwertyuiopasdfghjklzxcvbnm
```

Since there are 26! (or roughly 4×10^{26}) different arrangements of the 26 letters in the alphabet, there are 26! different keys and so 26! different substitution ciphers that can be defined.

Implement the `SubstitutionCipher` class so that it can be used to implement any substitution cipher, not just the Caesar cipher. The key to the particular cipher should be a parameter to the constructor. That key should be stored in a field and then used to encode and decode characters. For example, if the string `"qwertyuiopasdfghjklzxcvbnm"` was specified in the constructor of a Cipher object, then that object would encode `"Java code"` as `"Pqcq egrt"`.

Again, you may use it in combination with the provided `Encrypter` and `Decrypter` classes once it has been tested.

Part 3: Simple Rotating Ciphers

While substitution ciphers are reasonably effective at encoding/decoding messages, they are susceptible to various attacks that make them unworthy for serious encryption. Not all letters are equally likely in text, so the relative frequency of characters in a coded message can provide clues for decoding. For example, `'e'` is the most frequently used letter in English text. If the letter `'w'` appeared most frequently in a coded message, then one might guess that the substitution cipher maps `'e'` to `'w'`.

One approach to counter this type of analysis is to add rotation to the cipher. After each letter is encoded, the key is rotated so that the first character is moved to the end. For example, using the Caesar cipher key `"defghijklmnopqrstuvwxyzabc"`, the letter `'a'` would be encoded as `'d'`. But, after this encoding the key would be rotated to be `"efghijklmnopqrstuvwxyzabcd"`. Thus, if the next letter to be encoded was another `'a'`, it would get encoded as an `'e'` this time.

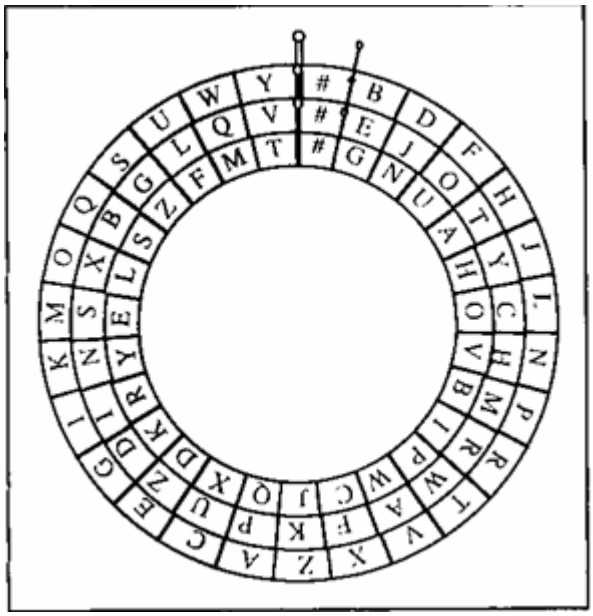
Implement the `RotatingCipher` class so that it performs a rotation after each character is encoded/decoded. You should define a private method named `rotate()` that rotates the key field. Then, this method should be called by both encode and decode after a character has been processed.

Once you have completed your modifications, create a rotating substitution cipher with the key `"qwertyuiopasdfghjklzxcvbnm"` and use it to decode the provided file "`unknown.txt`". If the decoded file is readable, then you will know that your implementation works as desired.

Part 4: A Simple Enigma Model

In World War II, the Nazi military employed an encryption scheme that addressed this weakness of simple substitution ciphers. This scheme, implemented by typewriter-sized devices known as Enigma machines, gave the Nazis a tactical advantage that greatly contributed to their early success in the war. In fact, the eventual breaking of this coding scheme by researchers at Bletchley Park, England (including Alan Turing) is hailed as one of the turning points of the war.

Enigma machines used interchangeable rotors that could be placed in different orientations to obtain different substitution patterns. More significantly, the rotors rotated after each character was encoded, changing the substitution pattern and making the code very difficult to break. The behavior of the rotating rotors can be modeled, in a simplified form, by a device consisting of labeled, concentric rings. For example, the model below has three rings labeled with the letters of the alphabet and '#' (representing a space).



To encrypt a character using this model, find the character on the inner rotor (i.e., the inside ring) and note the character aligned with it on the outer rotor (i.e., the outside ring), then find that character on the middle rotor (i.e., the middle ring) and output the one aligned with it on the outer rotor. After a character is encrypted, turn the inner rotor clockwise one step. Whenever the inner rotor returns to its original orientation, the middle rotor turns once in lock-step, just like the odometer in a car.

For example, in this configuration the character `'A'` would be encrypted as `'N'`, since `'A'` on the inner rotor is aligned with `'H'` on the outer rotor, and `'H'` on the middle rotor is aligned with `'N'` on the outer rotor. After performing this encryption, the inner rotor is rotated clockwise, so the letter `'A'` would next be encrypted as `'D'`.

Note that decrypting a message requires following the same steps, only in reverse (i.e., find the character on the outer rotor, note the character aligned with it on the middle rotor, find that character on the outer rotor, then output the character aligned with it on the inner rotor).

For this assignment, you are to design a Java class named `Enigma` that simulates this three-ring model. You may assume that all Enigma models have the same outer rotor, as shown in the above diagram. That is, the outer rotor consists of the 26 capital letters and the `'#'` symbol (representing a space) in the following clockwise order: `#BDFHJLNPRTVXZACEGIKMOQSUWY`. Since the other rotors are interchangeable, however, their contents and alignment relative to the outer rotor must be specified when constructing an Enigma model. For example, the initial settings of the inner and middle rotors in the above diagram are `#GNUAHOVBIPWCJQXDKRYELSZFMT` and `#EJOTYCHMRWAFKPUZDINSXBGLQV`, respectively. Using an `Enigma` object, it should be possible to encode and decode text messages, with the appropriate rotation of the rotors occurring after each character encoding/decoding.

Once you have your model working, test it by decoding the message OKNNWRDHGERPILRLAMFZF#FMUC using the diagram settings. Some longer sample files will also be provided.