Coding Exercise – Vigenère cipher

Vigenère cipher

A Vigenère cipher is a polyalphabetic substitution cipher. It is done by choosing the rotating substitution mappings with the given encryption **key** and a **source character set**. Any characters outside of the source character set will be copied as-is.

To illustrate how it works, consider the following example where the key is **"encrypt"** and the source character set is the lowercase alphabet letters **[a-z]**. In this example, we will encrypt the input string **"top secret"**.

The characters in the encryption key, "encrypt", determines the sequence of character mappings. Each mapping in the sequence is calculated with the characters from the encryption key by shifting the character set to the character of the key to produce the mapping from the input character to the cipher character.

```
                     Source Character Set
 key   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        Mapping
-----  ---------------------------------------------------       ---------
  e    e f g h i j k l m n o p q r s t u v w x y z a b c d  -->   t -> x
  n    n o p q r s t u v w x z y a b c d e f g h i j k l m  -->   o -> b
  c    c d e f g h i j k l m n o p q r s t u v w x z y a b  -->   p -> r
  r    r s t u v w x z y a b c d e f g h i j k l m n o p q  -->   s -> j
  y    y 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  -->   e -> c
  p    p q r s t u v w x y z a b c d e f g h i j k l m n o  -->   c -> r
  t    t u v w x y z a b c d e f g h i j k l m n o p q r s  -->   r -> k

 *** rotate the mapping back to the first character of the key here ***

  e    e f g h i j k l m n o p q r s t u v w x y z a b c d  -->   e -> i
  n    n o p q r s t u v w x z y a b c d e f g h i j k l m  -->   t -> g

 *** encryption ends ***
```

Take the first mapping with encryption key character 'e' from the above table. If the **first** input character is 'a', it will be mapped to cipher character 'e'. If the **first** input character is 'b', it would be mapped to the character 'f', 'c' to 'g', and so on.

After encrypting an input character, the cipher advance to the next mapping calculated from the encryption key for the next input character.

Take the second mapping with encryption key character 'n' from the above table. If the **second** input character is 'a', it will be mapped to cipher character 'n'. If the **second** input character is 'b', it would be mapped to the character 'o', 'c' to 'p', and so on.

This pattern continues for each input character until the last character of the encryption key is reached. At this point, the mapping will rotate back to the first mapping of the sequence.

If an input character is outside of the source character set, it would be copied as-is to the encrypted output, **and the mapping does not move to the next one in the sequence**.

Since the [space] character is outside of our example character set, it would be copied as-is, and the mapping will not advance.

Thus, in our example, the text input **"top secret"** will be encrypted as **"xbr jcrkig"**.

Requirements

This package contains a skeleton project files for the exercise. Please use only built-in libraries included in Oracle's JDK 1.7 or 1.8.

For this exercise, we are expanding the Vigenère cipher's character set to the below:

- This character set includes all the characters that can be entered using a standard US keyboard (including space, tab and new line characters, etc.).
- This character set is already in the skeleton code as "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz \t\n\r~!@#$%^&*()_+-=[]\{}|;':",./<>?"

For characters outside of the character set, copy them as-is to the output and do not map them. The encryption mapping should not advance for non-mapped characters.

The process needs to be reversible: the encrypted text can be decrypted with the same encryption key back to the original input.

The main method, and your implementation, should take exactly 3 parameters - [action] [key] [target].

- [action]: case insensitive values of either *encrypt*, *decrypt*, *encryptDir*, or *decryptDir*
- [key]: string of any character sequence to be used as the Vigenère encrypt key
- [target]:
    - For *encrypt* / *decrypt* actions: the value is the input text to be processed. For your output, perform the action and output the result, and only the result, using `System.out.println`.
    - For *encryptDir* / *decryptDir* actions: the value is a directory path where all the files to be processed. Your code must read and encrypt / decrypt each file in the directory and any files in any sub directories. There can be nested directories. For your output, create a new directory at the same level of the top directory for the by appending the top directory name with ".encrypted" or ".decrypted". Keep the file / directory names the same for anything under the top directory.
        - Example: If top directory path is "/path/to/directory" and [action] is *encrypted*, then the output results will be written to a directory at path "/path/to/directory.encrypted".

Your code must also meet the following assumptions. When you feel anything is ambiguous, please feel free to make additional reasonable assumptions. Please document your assumptions in the code so the reviewer knows any additional assumptions you made:

- The depth of directories and number of files can be arbitrarily large, e.g. 1,000 levels deep, 100,000,000 files
- Each input file can be arbitrarily large, e.g. 100GB in size

- The cipher can be easily updated, extended or replaced in the future
- The directory walking strategy can be easily changed or tweaked, e.g. filtering by file name, depth first, etc.
- Efficiency is required but concurrent processing is not required for this exercise, i.e. assume single thread processing

Output Format Examples

To be considered correct, your outputs must have formats that matches the following samples. All output examples refer to the sample source character set [a-z] above, and is not what your code actual output will be.

1. For parameters: [action] = *encrypt*, [key] = *encrypt*, [target] = *top secret*, the output format should be:

   ```
   xbr jcrkig
   ```

2. For parameters: [action] = *decrypt*, [key] = *encrypt*, [target] = *xbr jcrkig*, the output format should be:

   ```
   top secret
   ```

3. Consider a directory at */path/to/directory* with the structure:

   ```
   $ ls -a /path/to
   directory

   $ ls -a /path/to/directory
   file1.txt    file2.txt   folder1

   $ ls -a /path/to/directory/folder1
   file1_1.txt

   $ cat /path/to/directory/file1.txt
   top secret
   ```

   After executing parameters: [action] = *encryptDir*, [key] = *encrypt*, [target] = */path/to/directory*, the directory structure and text formats should now look as follow:

   ```
   $ ls -a /path/to
   directory    directory.encrypted

   $ ls -a /path/to/directory.encrypted
   file1.txt    file2.txt   folder1

   $ ls -a /path/to/directory.encrypted/folder1
   file1_1.txt

   $ cat /path/to/directory.encrypted/file1.txt
   xbr jcrkig
   ```

   Everything in */path/to/directory* should be left unchanged.

What We Look For

This exercise is your chance to demonstrate your Java coding skill. Show us you can put them in action and not just the words.

- Correctness
    - Encrypt / decrypt text correctly
    - Output matches expected format
- Completeness
    - All requirements are implemented
- Coding quality
    - OOP principals
    - Best programming practices
    - Readability
    - Maintainability

Submission

- When finished, please **zip** up your code in the directory structure exclude the classes files or temporary files.
- Please make sure your code submitted in the package can be compiled.