CSCI-605 Advanced Object-Oriented Programming Concepts

Homework 7 - Vigenère Cipher

1 Introduction

For this homework you will be writing a simple command line interface (CLI) for a utility that allows users to encode and decode files using a cipher.

A common reason for extending Java's base I/O stream classes is to *decorate* an underlying stream with a conversion. In this case, we are interested in a conversion that applies cryptographic encoding to an output stream and decoding to an input stream.

We will be using a pre-computer-age encryption method made popular in the 19th century by Blaise de Vigenère but actually invented by Giovan Bellaso approximately three centuries earlier. It in turn is a complication of the classic *Caesar Cipher* used by Julius Caesar, a politician and general of Rome in the first century B.C.E., for his own correspondence.

The Vigenère method was used by the army of the Confederate States of America during the U.S. civil war. Unbeknownst to them, their messages were actually being decrypted by the Union army.

Above you can see a replica of the C.S.A. coding mechanism.

1.1 Goals

Students will gain experience working with

- Input/Output Streams (java.io)
- The Decorator pattern
- Exceptions
  - Defining
  - Raising (throw)
  - Handling (catch)

1.2 Provided Files

1. `sample_run` - a folder containing some sample runs of the program.
2. `data` - a folder containing example files for you to practice encoding and decoding with your program.

2 Input Specification

Your implementation must support the following commands entered by the user via the command line:

- `help` - displays a list of available commands.
- `list <file path>` - displays a list of files in the specified directory.
- `encode <key> <input-file> <output-file>` - encode the input file into the output file using the key indicated.
- `decode <key> <input-file> <output-file>` - decode the input file into the output file using the key indicated.
- `quit` - quits the Vigenère Cipher.

3 Cipher Specification

You will write decorators for `java.io.Writer` and `java.io.Reader` that perform Vigenère-style encryption.

Both classes initialize their objects with two arguments.

1. a base `Writer` or `Reader` instance
2. a keyword, consisting of only characters from the English alphabet, lower or upper case

Each letter in the keyword represents an offset, or adjustment, that may have to be made to a character in the provided text. The offset is the "distance" in the alphabet of that letter from 'a' (or 'A' if the keyword character is upper case). 'a' and 'A' represent 0 distance, while 'z' and 'Z' represent a distance of 25. Since both the lower case and upper case English letters are contiguous in UNICODE16, that means that the distance represented by a lower-case character `c` from the keyword is `c - 'a'`. (Similarly for upper case.)

First, one has to imagine the keyword being duplicated over and over again so that it spans the entire output or input stream. Then envision the sequence of keyword characters being replaced by the distances, as integers, that those characters represent.

For the `VigenereCipherWriter` each letter character being sent in for writing is changed by adding the distance seen at the corresponding location in the keyword "stream".

For the `VigenereCipherReader` each letter character being read is changed by subtracting the distance seen at the corresponding location in the keyword "stream" before passing it to the caller of the read operation.

In both cases the calculation "wraps around" if it exceeds an end of the alphabet.

Non-letter characters do not get transformed in any way. However, there is a character of the keyword stream "overhead", so you have to take this into account when lining up the keyword stream. (It's actually algorithmically simpler than if the non-letter characters were skipped.)

For example, writing "ab cx yz" to a `VigenereCipherWriter` with keyword "ACC" will send the string "ad cz yb" to be written by the base `Writer`. Reading "abC,Xyz" from the base Reader of a `VigenereCipherReader` with keyword "cca" will return the string "yzC,Vyx" to the caller of the reading method.

Here is the longer example:

```
Plaintext:   F o u r   s c o r e   a n d   s e v e n . .
Keyword:     L i n c o l n L i n c o l n L i n c o l n L
Ciphertext:  Q w h t   d p z z r   o y q   a r x s y . .
```

4 Implementation

4.1 Design

For this homework you will need to implement the following classes:

- `VigenereException` - A custom checked exception. This is the only exception that the utility and the cipher classes (writer and reader) should explicitly throw. Many of the classes that you use will throw IOException (or one of its subclasses). You should catch and rethrow any of these exceptions as an instance of your custom exception.
- `VigenereCipherWriter` - A class implementing the Vigenère cipher encoding by "decorating" an underlying Writer.
- `VigenereCipherReader` - A class implementing the Vigenère cipher decoding by "decorating" an underlying Reader.
- `VigenereUtility` - A utility class with methods for listing a directory files, encoding, and decoding files.
- `VigenereCipher` - The main class for your user interface. This class is solely responsible for prompting the user, reading any user input (commands entered via System.in), and providing any output to the user (through System.out and System.err). Your user interface should handle any exceptions that occur without crashing your program; instead, provide the user with appropriate output messages and continue running.

4.2 Cipher Implementation

The `VigenereCipherWriter` and `VigenereCipherReader` classes implement Vigenère cipher encoding and decoding, respectively, by "decorating" an underlying Writer and Reader, while adhering to the Writer and Reader abstractions. The underlying, or "base" object is sometimes called a *delegate*. In addition, your classes extend `Reader` and `Writer`.

Since `Reader` and `Writer` are not interfaces but rather abstract classes, you have some flexibility in how much you write in your subclasses. Below are the design expectations. Any

additional methods you override can simply help improve performance, with which we are not concerned in this lab assignment.

- `write(char[],int,int)`, `flush()`, and `close()` are abstract in `Writer` and therefore must be implemented in `VigenereCipherWriter`.
- `read(char[],int,int)` and `close()` are abstract in `Reader` and therefore must be implemented in `VigenereCipherReader`.
- If `mark`ing is supported by your underlying `Reader` base class, your decorating `VigenereCipherReader` must also support it. This means you must provide implementations for `mark()`, `markSupported()`, and `reset()` that largely rely on those methods in your base.
- Because your underlying `Writer` base class might support `flush()`, your decorating `VigenereCipherWriter` must support it by calling the base class's method.
- It is highly likely that you must also add functionality to your base `Reader`'s `skip()` method, since you need to maintain a position in your repeating keyword stream.

4.3 Hints

Carefully read all of the Java documentation, before writing any code. This includes reading the Java documentation for the `Reader` and `Writer` classes.

Arithmetic on `char` values is performed by promoting the `char` values to `int`; when necessary, explicitly cast the result back to `char`. However, directly subtracting two `char`s is legal and produces the integer distance between their codes. For similar reasons, adding or subtracting an `int` to/from a `char` is also legal.

5 Test Data

kublakhan.txt is the text of the first stanza of Samuel Taylor Coleridge's poem "Kubla Khan; or, A Vision in a Dream: A Fragment".

kublakhan.opium.txt is the output of encoding the **kublakhan.txt** file using the keyword **opium**.

```
encode opium kublakhan.txt kublakhan.opium.txt
```

Your submission should exactly encode **kublakhan.txt** to **kublakhan.opium.txt** and exactly decode **kublakhan.opium.txt** to **kublakhan.txt**, using the key "opium".

AliceChapter1.txt and AliceChapter1.rabbit.txt make up the same idea for the first chapter of "Alice in Wonderland" with the key "rabbit".

AliceInWonderland.txt and **AliceInWonderland.caterpillar.txt** make up the same idea for the entire book "Alice in Wonderland" (by Lewis Carroll)with the key "caterpillar".

6 Submission

You will need to submit all of your code to the MyCourses assignment before the due date. You must submit your `hw7` folder as a ZIP archive named "`hw7.zip`" (if you submit another format, such as 7-Zip, WinRAR, or Tar, you will not receive credit for this homework). Moreover, the zip file must contain only the java files of your solution. Do not submit the .txt files provided in this homework nor compiled files.

7 Grading

The grade breakdown for this homework is as follows:

- Design 15%
- Functionality 75%
  - Program functionality: 20%
  - Custom Exception Implementation: 10%
  - Vigenere Cipher Reader: 25%
  - Vigenere Cipher Writer: 20%
- Style 10%