Project 2

Superencipherment
Tvqfsfodjqifsnfou

CSCI 1913: Introduction to Algorithms,
Data Structures, and Program Development

1 Essential information

Project 2 is designed as an approximately 2 week assignment. It builds upon Lab 8 and focuses on three core ideas: Classic String cipher algorithms, Simple, but useful, java objects (I.E. java objects that abstract an algorithm or operation), Polymorphism, and a powerful way to use Polymorphic code.

Project 2 is split into three "phases" which should be done in-order. Aim to be done with phase 1 by Nov 14, Phases 2 and 3 are less overall programming, but rely on your understanding of inheritance and polymorphism.

Project 2 is due on Wednesday November 23rd at 6:00pm. Late work will not be accepted, however a small grace period will be available on gradescope to allow for technical difficulties.

This is version 0.9 of the project writeup.

Changelog:

1. (10/07) v0.9 First version – grading and submission information have not been provided yet.
2. (11/18) v1.0 submission information provided. **AND ONE KEY CLARIFICATION** – you should not modify BaseCipher!

1.1 Learning Goals

This project is designed with a few learning goals in mind

- Practice building several *useful* and *interesting* java classes.
- Practice thinking about complicated behaviors with strings.
- Practice reading formal java documentation and finding the most useful functions from those lists.
- Practice working with polymoprhism
- Writing code that is *explicitly* polymorphic
- Like the last lab – we will be limiting your access to automatic testing. This will help you learn healthy programming habits (namely, running tests on your own, and looking manually at those results). In general, you should never need the autograder to tell you that your code doesn't meet expectations.

2 Introduction

In lab 8 we saw the basic Caesar Cipher. One of the comments in this lab was that Caesar ciphers are easy to break automatically (we even built a basic piece of the code for this) Letters are not used at the same rate in common English text, so you can simply try the 25 valid cipher rotations and see which one leads to a pattern of letter use that most matches English language. While this was a time-consuming task when the Caesar cipher was invented – it can be done reliably, automatically, and FAST in the modern era.

This is all to say that, if we wish to use these ciphers to communicate securely, we need more. In this lab we will start by adding two additional types of ciphers – a letter shuffle cipher, in which the order of letters in the message is changed, and a replacement cipher, in which certain "secret" words are replaced with other less-secret words. Each of these ciphers alone can be easily broken, but taken together, these ciphers can be relatively useful.

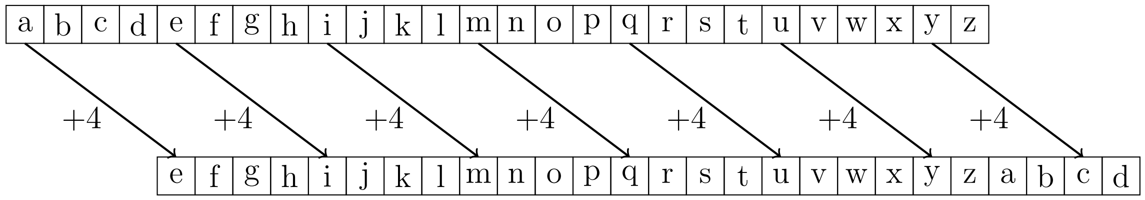
Superencipherment is the idea of creating secure ciphers by combining many simpler ciphers together. If we take our three ciphers and combine them in clever ways we can make ciphers that would be actually difficult to decrypt, especially with minor use. Supporting Superencipherment in Java is easier if we make use of polymorphism. This is the java programming language feature in which we can make all three cipher objects subclasses of a parent "BaseCipher" class. By doing this we can have BaseCipher variables which can refer to any cipher at all. We could even have an array of type BaseCipher which stores any given ordered series of ciphers. This doesn't add much complexity to the individual ciphers, but allows us to write powerful functions that work with any Cipher, or a SuperCipher class, which applies arbitrarily ordered ciphers.

This project is designed in three phases. The first phase is to simply implement the core ciphers, Caesar (which you will be implementing as part of Lab 8), and EvenOdd (a cipher built on reordering letters) a word replacement cipher. This can be done before we have fully explored inheritance and polymorphism in class. The second phase is to modify these classes to use inheritance, you should also fully debug these classes at this point. The third and final phase is to add a SuperCipher class, and an EncryptUtils class which use the other ciphers polymorphically, allowing powerful and customize superencipherment.

3 Details of the encryption algorithms themselves

The three base encryption algorithms you are asked to implement are the Caesar cipher (described in lab 8), the word replacement cipher, and the evenOddCipher. The code for these has a lot of similarities. Like in lab8, all of these algorithms should turn uppercase letters into lowercase letters. This helps keep the ciphers working, especially when they need to work together.

3.1 CaesarCipher


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
+4 +4 +4 +4 +4 +4 +4
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

The only added feature of this class is that it will need to support java-standard equality checking methods. See Lab 8 for more information on this cipher.

3.2 Word Replacement Cipher

The word replacement cipher is relatively simple, it swaps each occurrence of one word for a pre-chosen replacement.

For example, if the cipher replaces “cat” to “dog”, it would encode the text “I got a new cat! I love it’s ears!” to “I got a new dog!, I love it’s ears!”. Decoding simply involves reversing the replacement (so changing dog back to cat).

This cipher is great at disguising secret text when code words are well chosen, and really shines when a relatively long list of code words is adopted. This is also quite hard to decode, as there is often no pattern or rhyme to these replacements, making them hard to automatically detect or undo. In our implementation we will focus only on performing one word’s worth of replacement, with multiple code words being treated as many separate instances of a word replacement Cipher. The downside for these benefits is that this encryption is not guaranteed to be reversible (unlike the others).

As an example, consider the text “I love cats and dogs equally”, this encodes to “I love dogs and dogs equally” (cat replaced with dog) and decodes again to “I love cats and cats equally”. (note that *Every* occurrence of secret words should be replaced). This is an inherent weakness of the cipher, and not a bug we will be addressing. The best way to fix this is by careful consideration of the code book. Our version will also work simply on "strings" and not words, it would be valid, therefore, to replace each "th" in any word with " taco " (spaces included), while this breaks word boundaries, it works fine for software.

The following website contains a list of methods available on Java's String class `https://docs.oracle.com/javase/8/docs/api/java/lang/String.html` You should plan on looking over this website to find java's built-in String replacement methods. A few notes on this:

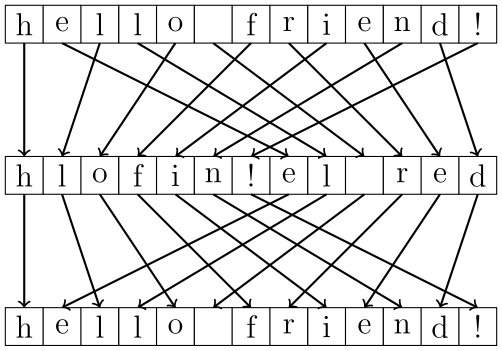
- If you've never sat down to look at java's formal documentation it can be a bit intimidating at first, nonetheless, you should know enough by now to understand the information being presented. The ability to read formal documentation like this and learn from it is viewed as a key programmer skill – do not run from this if you wish to excel in this field. Therefore, to be clear: *we are expecting each student to go to the formal documentation to find the most appropriate method to use here.*

- This problem is *deceptively complicated* – The function whose name suggests it's most-useful is actually incorrect. You may need to read the full description, and even look breifly to see what related ideas "regular expressions" and "CharSequences" mean in order to know which function would work correctly.

- Don't forget to *put the science in computer science!* – write yourself small test programs to make sure you know the right way to use the replacement methods, and reach out for assistance if you really can't sort through the formal documentations to find your answer after a few hours of trying.

3.3 EvenOdd Cipher

Even Odd Cipher is a "Scramble" cipher, that disguises text by shuffling the letters up. The idea is quite simple, first list all the even letters, then list all the odd letters.


h e l l o   f r i e n d !
h l o f i n ! e l   r e d
h e l l o   f r i e n d !

The code for encoding in this algorithm is pretty simple, simply make an empty string, add all the even indexed letters (0, 2, 4, 6, 8, ...), then add the odd indexed letters (1, 3, 5, 7, ...)

The code for decoding is a bit more challenging, you will want to compute the "crossover" point in the string where it goes from "even" to "odd". This is at $\lceil length/2 \rceil$ (note, that's round up, not round down). Then you need to take the first even letter, then the first odd letter, and so forth.

The charAt and length string methods may be specifically useful for this task. You can look these up in the String documentation mentioned above. That said, you've likely worked with these functions before now.

3.4 Super ciphers

A super cipher is a cipher that is formed by doing multiple ciphers in steps. For example you might have a super cipher formed by first replacing "cat" with "dog", then applying a Caesar cipher (+4) to the result, and finally applying an even-odd cipher to that.

`"I like cats a lot"` becomes `"i like dogs a lot"`, which then becomes `"m pmoi hskw e psx"`, which then becomes `"mpo swepx mihk  s"` (two spaces between k and s).

Decryption needs to go in backwards order, first undo the even-odd, then undo the Caesar, then undo the word replace. If you get the order wrong some decryptions will work fine, but any decryption combining even-odd and replacement ciphers will not work.

In our code, we will do this with an array of type BaseCipher. Due to polymorphism, this array can store any object that is a subtype of BaseCipher. If we make the other three ciphers extend BaseCipher, then we can represent the above chain of ciphers as

```
new BaseCipher[] {new WordReplacementCipher("cat", "dog"),
                  new CaesarCipher(4),
                  new EvenOddCipher()};
```

The SuperCipher class itself will not actually be in charge of doing much encryption, instead it's job is to arrange other ciphers to encrypt the text (in the right order)

4 Phase 1

The first phase of this project is to implement CaesarCipher (as seen in Lab 8), WordReplacementCipher, and EvenOddCipher classes, at least at a basic level. For now, don't worry about the BaseCipher class, or the SuperCipher class.

The formal requirements of these are as follows: **Note – all functions are listed in this PDF with only the type of their parameters specified** (you will need to pick parameter names).

4.1 CaesarCipher

A Phase 1 CaesarCipher will implement the following:

- A public constructor that takes one parameter, (an int) the amount to rotate by.

- A public method `isValid()` that returns a boolean indicating if the amount to rotate by is valid (between 1 and 25).
- A public method `encrypt(String)` that returns a String, the result of applying a Caesar cipher to the input.
- A public method `decrypt(String)` that returns a String, the result of reversing a Caesar cipher on the input.
- A public method `toString()` that returns a String, which describes the object, the format for this must match what is seen in the tests.
- A public method `equals(Object)` that returns a boolean, which is true if it is passed a another Caesar cipher with the same rotation amount, and false in all other circumstances.

Once implemented most, but not all, of the CaesarCipherTest should run correctly.

4.2 WordReplacementCipher

A Phase 1 WordReplacementCipher will implement the following:

- A public constructor that takes two parameters, both strings, representing the word to replace from, and the word to replace to.
- A public method `encrypt(String)` that returns a String, the result of applying the word replacement to the input.
- A public method `decrypt(String)` that returns a String, the result of reversing the word replacement on the input.
- A public method `toString()` that returns a String, which describes the object, the format for this must match what is seen in the tests.
- A public method `equals(Object)` that returns a boolean, which is true if it is passed a another WordReplacement cipher with the same from and to words.

Once implemented most, but not all, of the WordReplacementCipherTest should run correctly.

4.3 EvenOddCipher

A Phase 1 EvenOddCipher will implement the following:

- A public method `encrypt(String)` that returns a String, the result of applying the even odd cipher to the input.

- A public method `decrypt(String)` that returns a String, the result of reversing the even odd cipher on the input.
- A public method `toString()` that returns a String, which describes the object, the format for this must match what is seen in the tests.
- A public method `equals(Object)` that returns a boolean, which is true if it is passed a another EvenOddCipher.

Once implemented most, but not all, of the EvenOddCipherTest should run correctly.

5 Phase 2

The second phase of this project is to make all three Cipher objects extend the provided BaseCipher class. This will require only minor changes to the other cipher classes, in particular the constructors will likely need changing. You should also try to leverage the inherited code from the base class wherever possible, if the base classes implementation of a method is sufficient, just use that version. **NOTE** we will be reviewing your code manually and will dock points for unnecessarily overridden functions.

NOTE You should not, not do you need to, change the BaseCipher for this step.

Once phase 2 is complete the tests for the base classes should run completely. This is a good moment to stop and check each class carefully to be sure it's correct.

6 Phase 3

The third and final phase involves leveraging the work in the last phase. By making each simple cipher a subtype of BaseCipher you can now use polymorphism to make more powerful ciphers. If you do this right you should be able to make these functions work with any of your cipher classes with no specific or special code. So you should be able to call the encryptMany function with a Caesar cipher *without the encrypt many function having any knowledge of the Caesar cipher!* In fact, the functions in Phase 3 should even work with *new cipher classes you program after phase 3* (without any change to the original code for phase 3). There are three classes to implement here:

6.1 EncryptUtils

This class has two static methods that handle repeated encryption or decryption over sequences of strings. A complete EncryptUtils must implement the following functions:

- a public static function `encryptMany(BaseCipher, String[])` which takes a BaseCipher variable (which of course may be any subtype as well), and an array of strings. The function should return a new array of strings that is the result of encrypting each String using the cipher. The input array should remain unchanged.

- a public static function `decryptMany(BaseCipher, String[])` which takes a BaseCipher variable (which of course may be any subtype as well), and an array of strings. The function should return a new array of strings that is the result of decrypting each String using the cipher. The input array should remain unchanged.

Once written the test EncryptUtilsTest file should pass.

6.2 SuperCipher

The SuperCipher class represents a supercipher formed by applying a chain of other ciphers. It will store an array of type BaseCipher indicating which ciphers and in what order the super cipher should apply. This class should also extend BaseCipher.

A complete SuperCipher class must:

- extend BaseCipher
- have a public constructor that takes one parameter, an array of type BaseCipher indicating which ciphers and in which order this supercipher should use.
- a public method `isValid()` which returns a boolean. A SuperCipher is valid if, and only if, each base cipher is valid.
- a public method `encrypt(String)` which returns a String, the result of applying each cipher provided to the constructor, in the order provided.
- a public method `decrypt(String)` which returns a String, the result of reversing each cipher provided to the constructor, in the correct order.
- a public method `toString()` which returns a String. The format for this can be found in the test files.
- a public method `equals(Object)` which returns a boolean. A superCipher is equal to another SuperCipher if and only if it's chain of ciphers match (I.E. are the same length, and are equal at each position). You may find the equals function on java's built in class Arrays useful for this.

Once done, the SuperCipherTest should pass.

6.3 FinalDecoding

The final decoding class contains a mostly complete main method. This describes some intercepted messages and what we know about how the messages were encoded. Your job is to setup a supercipher object to use when decrpyting these messages. It will be pretty obvious if the messages decrypt correctly.

7 Files on Canvas

The following files will be posted on canvas for you to use when starting:

- `BaseCipher.java` - a complete base cipher class
- `CaesarCipherTest.java` - a test for the CaesarCipher class
- `EncryptUtilsTest.java` - a test for the EncryptUtils class
- `EvenOddCipherTest.java` - a test for the EvenOddCipher class
- `FinalDecoding.java` - the final decoding class
- `SuperCipherTest.java` - a test for the SuperCipher class
- `WordReplacementCipherTest.java` - a test for the WordReplacementCipher class.

8 Testing and Grading

For submission, please turn in the following files:

- `CaesarCipher.java`
- `EvenOddCipher.java`
- `WordReplacementCipher.java`
- `SuperCipher.java`
- `EncryptUtils.java`
- `FinalDecoding.java`

You should not submit a new BaseCipher (No where in the instructions were you told to modify the class, and no modifications are needed). The autograder has been *Specifically designed to not at your BaseCipher* submission. To pass tests, therefore, your code must work with the provided BaseCipher Object.

Points will be assigned as follows:

- (35 points) Autograder
- (10 points) Code Style
- (6 points) Caesar Cipher
- (14 points) Even Odd Cipher
- (10 points) Word Replacement Cipher

- (14 points) Super Cipher
- (8 points) EncryptUtils
- (3 points) FinalDecoding