CS 100 Project 1: Encryption
Summer 2001
Due in lecture, Thursday July 26

Objectives

Completing this project will help you:

- Gain experience using file input and output
- Handle exceptions
- Manipulate one-dimensional arrays
- Write good documentation

First, skim the whole project. Then carefully read all instructions before starting. You must use a **fixed-width** font (e.g., Courier or Monaco) for your homework and you must **staple** your assignment together.

Submission for this project's java files will be done ONLINE. The documentation for your project should still be printed out and turned in at lecture. To submit your files, go to

http://submit.cs.cornell.edu/cs100/scripts/submit.pl

This link will also be available from the Assignments page of our website. For submission purposes, your username will be your Netid. Your password, however, is **not** the password you use for Bear Access. Instead you should use the password you received via email (to your netid@cornell.edu).

IMPORTANT! We are testing your programs by actually running them with different inputs. This means two things:

a) Make sure your program **compiles** and **runs** before submitting it. If there are parts of your program that are still causing problems, comment them out. A program that compiles and runs is vastly superior to one that does not. We may refuse to grade submissions that do not compile.

b) Your program should **conform exactly** to the specification we provide. We will make available sample output that shows exactly what your program should do, for a given input. Match this exactly so that we can easily compare it to the proper answer. Also, **do not change** any parts of a program that are indicated "do not modify".

1. Encryption and Decryption

Security on the Internet is a big concern these days. This is an issue that arises when you want to use a credit card to make an online purchase, access your bank account online, or send a secret love letter to a friend. Whenever you have sensitive information that you don't want other people to see, you need to be able to protect it. One way to do this is to use an **encryption method**.

Computers have made encryption (and decryption) much easier than it once was, but the need for secure communications is by no means new. People have been sending encrypted messages since ancient history. But doing encryption and decryption with a computer is **much** faster than doing it by hand!

1. **Read** the accompanying handout (excerpt from "The Code Book", by Simon Singh, pages 10-25). Note that **plaintext** refers to the message you wish to encrypt, and **ciphertext** refers to the text after it has been encrypted.

2. Download **Encrypt.java**, **CaesarShift.java**, **FrequencyAnalysis.java**, **Decrypt.class**, and **SavitchIn.java** from the website. Create a new project and add all five files to it. (That is not a typo – you will add a **.class** file to the project.)

- **Encrypt.java**: this is the class that will do simple substitution encryption. This means that you take in a substitution string with length 26 that indicates how to substitute characters in the plaintext to generate the ciphertext. Thus, the first character in the substitution string indicates what to replace 'a' with, the second character indicates what to replace 'b' with, and so on.

  This program should prompt the user for the name of an **input plaintext file**, an **output encrypted file**, and a **substitution string, in that order.** Do not change the order. You must check that the substitution string is a valid one – it must be of length 26, and it cannot have any duplicate letters (do your checks in this order). If not, your program should **immediately exit** (with an informative error message). Otherwise, you should store

this substitution information in an **array** for later use (yes, Strings are a kind of array. But we want you to use a character array to store it, for this project. If you have questions about this, ask!).

You may also find the methods in the **Character** class useful (see p. 322 in Savitch).

Then, your program should **encrypt the input file** by reading it in and replacing each character with the appropriate one (from the substitution string). Hint: you may find it very useful to convert characters to integers. Remember you can do this with a cast – but be aware that 'a' does not convert to 0, and 'a' does not have the same value as 'A'. See the ASCII table in Savitch (Appendix 3).

Finally, you should **write out the encrypted text** to the output file.

Some important notes:

1. Any characters in the input that are not in the alphabet (punctuation, spaces, etc.) should be passed through **unchanged**.
2. Your conversion should be **case-sensitive**. If 'r' is supposed to be replaced by 'd' and you encounter a capital 'R' in the plaintext, it should come out as a 'D' in the ciphertext (not a 'd').
3. Java will force you to catch **IOException** and **FileNotFoundException** to do file input and output.
4. If you use any debugging output, make sure it goes to the screen and **not** to the output file.

- **CaesarShift.java**: this class is a special case of the general substitution algorithm you implemented for Encrypt.java. The Code Book talks about **Caesar shift** encryption, where the substitution string has each simply shifted by a fixed amount, but the order of the alphabet is preserved.

  This program should again prompt the user for an **input** and **output** file. In addition, it should prompt the user for a **shift constant**, which will indicate how far to shift the alphabet. Then, after calculating what the substitution string should be, you can call your substitution method from Encrypt.java to perform the encryption. Again, **write out the encrypted text** to the output file.

  Some important notes:

  1. Always shift the substitution string to the **left**. Thus if the user specifies a Caesar shift of **3**, then 'a' gets mapped to 'd'.
  2. If you use any debugging output, make sure it goes to the screen and **not** to the output file.

- **FrequencyAnalysis.java**: this class should analyze an encrypted text to determine how frequently each letter occurs. As you read in the excerpt from "The Code Book", this is a useful way to approach a new encrypted text to try to figure out what the substitution string is.

  Your program should again prompt the user for an **input** and **output** file. This time, it will read in the contents of the input file and **determine how often each letter occurs**. This should **not** be case-sensitive; the string "JimMy" has two m's in it. The output (which should be written to the output file, and to the screen as well if you like) should look like what you see on page 21 of "The Code Book", except that you don't need to have two letters per line. We will post example output that you should emulate.

  If you use any debugging output, make sure it goes to the screen and **not** to the output file.

- **Decrypt.class**: this is the class file for an interactive decryption program. You do not have to write anything for this (it has already been written for you!). To use it, set the Target to Decrypt and run the program. You'll be able to run it even though you don't have a Decrypt.java file. This program lets you experiment with how encryption and decryption work.

  The program prompts you for the name of an **input text file** (this will be the encrypted text) and the name of an **output file** (to write the decrypted version to), and then lets you **interactively specify** that a certain character should be replaced by another character. This is just like solving a cryptogram, which you may have done before. You can easily test out different values until you've discovered what the correct substitution string is. After running the program, look at the output file to get the final decrypted text.

3. Test your newly implemented methods using input files we'll provide and comparing the output to what we indicate the correct output should be. Use all tools at your disposal to track down (and fix) problems: use the CodeWarrior debugger, output to the screen, consultants, etc.

It is **your responsibility** to ensure that you've tested your code appropriately. We will be running our own set of additional "beta" tests on your submitted code. It is to your advantage to try to anticipate what you think we might test, and run those tests before handing in your homework.

4. Submit Encrypt.java, CaesarShift.java, and FrequencyAnalysis.java at the submission website (see above).

Bonus point:
The Code Book also mentions **keyphrase encryption**. This is very convenient – it is much easier to remember a short phrase than an entire 26-character substitution string. For a bonus point, write a program called **Keyphrase.java** that prompts the user for input and output files and a keyphrase, then generates the appropriate substitution string and calls the substitution method in Encrypt.java to encrypt the input text.

To have your bonus work graded, submit Keyphrase.java at the submission website.

2. Documentation

Any good project should also be supported with proper documentation. For the purposes of this assignment, you must turn in two things (in a **regular text file**, **Documentation.txt**, not a Word document or something else):

1. Full **documentation of the files** you've implemented, including their **interfaces** (public methods and variables) and **internal implementations** (private methods and variables). This should be detailed enough so that another programmer could read your documentation and then be able to use your provided methods. Include a description of your **design choices** – while developing a project, there are several decisions that have to be made (For example, what kind of prompt will you provide to the user? What kind of error conditions will you handle, and how? What parts of the code should you split up and create a separate method for?), and it is important to describe and explain these choices to your reader.

2. A **report on the tests** you conducted while developing your project. What kinds of input did you test? Include a **sample input** and **the resulting output** for running the **general substitution** encryption algorithm, another one for a sample run of the **Caesar shift** algorithm, and a third for the **frequency analysis** program. Discuss what you observed and why you selected those inputs in particular (what were you hoping to test?).