**CMSC203**

**Assignment #3**

In [cryptography](https://en.wikipedia.org/wiki/Cryptography), **encryption** is the process of encoding a message or information in such a way that only authorized parties can access it and those who are not authorized cannot. In an encryption scheme, the intended information or message, referred to as plaintext, is encrypted using an encryption algorithm – a cipher – generating ciphertext that can be read only if decrypted.

The Enigma machines were a family of portable [cipher](https://en.wikipedia.org/wiki/Cipher) machines with [rotor](https://en.wikipedia.org/wiki/Rotor_machine) [scramblers](https://en.wikipedia.org/wiki/Scrambler) that became [Nazi Germany](https://en.wikipedia.org/wiki/Nazi_Germany)'s principal [crypto-system](https://en.wikipedia.org/wiki/Cryptography). It was broken by the [Polish General Staff's](https://en.wikipedia.org/wiki/Polish_General_Staff) [Cipher Bureau](https://en.wikipedia.org/wiki/Biuro_Szyfr%C3%B3w) in December 1932, with the aid of French-supplied intelligence material obtained from a German spy.

Enigma machine used by Nazi Germany

during World War II

**Assignment Description**

Write a Java program to encrypt and decrypt a phrase using two similar approaches, each insecure by modern standards.

The first approach is called the Caesar Cipher, and is a simple “substitution cipher” where characters in a message are replaced by a substitute character.

The second approach, due to Giovan Battista Bellaso (b 1505, d 1581), uses a key word, where each character in the word specifies the offset for the corresponding character in the message, with the key word wrapping around as needed.

**Concepts tested by this assignment**

* Using loops
* String and character processing
* ASCII codes

**Classes**

**Data Manager class – CryptoManager.java**

* Implement each of the methods specified in this file. This version as provided will print error messages in the console, because they are just the skeletons.
* Each of the methods are static, so there is no need to create an instance of the Data Manager.
* Document each of your methods with a simple description and document the class with a simple description and your name using in-line comments (//…). (Just a short sentence fragment will suffice for each documentation string.)
* The methods are described below.
* public static boolean stringInBounds (String plainText);

This method determines if a string is within the allowable bounds of ASCII codes according to the LOWER\_BOUND and UPPER\_BOUND characters. The parameter plainText is the string to be encrypted. The method returns true if all characters are within the allowable bounds, false if any character is outside.

* public static String encryptCaesar(String plainText, int key);

This method encrypts a string according to the Caesar Cipher. The integer key specifies an offset and each character in plainText is replaced by the character the specified distance away from it. The parameter plainText is an uppercase string to be encrypted. The parameter key is an integer that specifies the offset of each character. The method returns the encrypted string.

* public static String decryptCaesar(String encryptedText, int key);

This method decrypts a string according to the Caesar Cipher. The integer key specifies an offset and each character in encryptedText is replaced by the character "offset" characters before it. This is the inverse of the encryptCaesar method. The parameter encryptedText is the encrypted string to be decrypted, and key is the integer used to encrypt the original text. The method returns the original plain text string.

* public static String encryptBellaso(String plainText, String bellasoStr);

This method encrypts a string according to the Bellaso Cipher. Each character in plainText is offset according to the ASCII value of the corresponding character in bellasoStr, which is repeated to correspond to the length of plaintext. The method returns the encrypted string.

* public static String decryptBellaso(String encryptedText, String bellasoStr);

This method decrypts a string according to the Bellaso Cipher. Each character in encryptedText is replaced by the character corresponding to the character in bellasoStr, which is repeated to correspond to the length of plainText. This is the inverse of the encryptBellaso method. The parameter encryptedText is the encrypted string to be decrypted, and bellasoStr is the string used to encrypt the original text. The method returns the original plain text string.

* Add additional methods if you wish to make your logic easier to follow.

**GUI Driver class – (provided)**

* A Graphical User Interface (GUI) is provided. Be sure that the GUI will compile and run with your methods. The GUI will not compile if your method headers in CryptoManager.java are not exactly in the format specified. When you first run the application, your methods will all throw exceptions, which will be caught by the GUI and printed out in the console.
* Do not modify the GUI.
* The GUI takes care of capitalizing your input strings.

**JUnit Test/Test Driver**

* Once your methods are implemented, run the test driver. Ensure that the driver file results in the following output, as shown in RED (of course the output will not be in red when you run the test driver):

"THIS TEST SHOULD SUCCEED" Is it in bounds? true

"THIS TEST THAT SHOULD FAIL BECAUSE { IS OUTSIDE THE RANGE" Is it in bounds? false

"This test should fail because of lowercase letters" Is it in bounds? false

Caesar cipher of "THIS IS ANOTHER TEST" should return "WKLV#LV#DQRWKHU#WHVW": WKLV#LV#DQRWKHU#WHVW

Bellaso cipher of "THIS IS ANOTHER TEST" should return "WU\VR9F#N!RF88U-'HED": WU\VR9F#N!RF88U-'HED

Caesar decryption of "WKLV#LV#DQRWKHU#WHVW" should return "THIS IS ANOTHER TEST": THIS IS ANOTHER TEST

Bellaso decryption of "WU\VR9F#N!RF88U-'HED" should return "THIS IS ANOTHER TEST": THIS IS ANOTHER TEST

* Run the JUnit test file (provided). Ensure that the JUnit tests all succeed.
* Include CryptoTest.java and CryptoManagerTest.java with the rest of your submission.

**Assignment Details**

The first approach is called the Caesar Cipher, and is a simple “substitution cipher” where characters in a message are replaced by a substitute character. The substitution is done according to an integer key which specifies the offset of the substituting characters. For example, the string ABC with a key of 3 would be replaced by DEF.

If the key is greater than the range of characters we want to consider, we “wrap around” by subtracting the range from the key until the key is within the desired range. For example, if we have a range from space (‘ ‘) to ‘\_’ (i.e., ASCII 32 to ASCII 95), and the key is 120, we note that 120 is outside the range. So we subtract 95-32+1=64 from 120, giving 56, which in ASCII is the character ‘8’. If the key is even higher, we can subtract the range from the key over and over until the key is within the desired range.

Since our specified range does not include lower-case letters, the GUI (provided) will change strings to upper case. You can find the ASCII table at <http://www.asciitable.com/>, or many other places on the Internet.

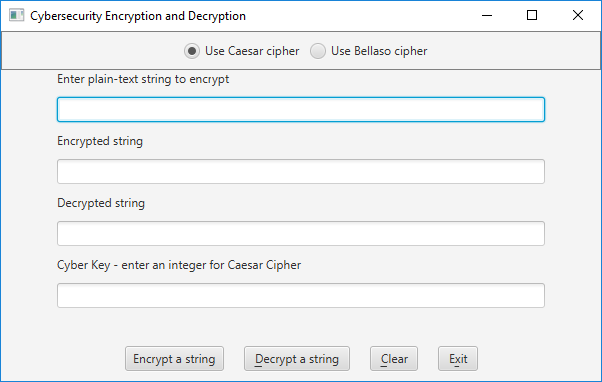
The second approach, due to Giovan Battista Bellaso (b 1505, d 1581), uses a key word, where each character in the word specifies the offset for the corresponding character in the message, with the key word wrapping around as needed.

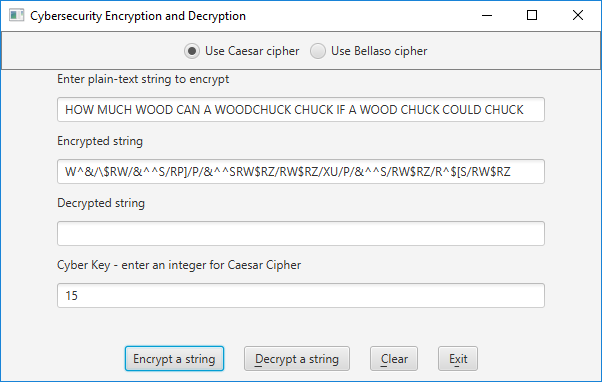
So for the string ABCDEFG and the key word CMSC, the key word is first extended to the length of the string, i.e., CMSCCMS. Then A is replaced by ‘A’ offset by ’C’, i.e., ASCII 65+67=132. The range of the characters is also specified, and again we’ll say ‘ ‘ to ‘\_’ (i.e., ASCII 32 to ASCII 95). The range is then 95-32+1=64. In our example, the offset is “wrapped” by reducing 132 by the range until it is the allowable range. 132 is adjusted to 132-64=68, or character ‘D’ in the encrypted phase. Then the same logic is applied to the second letter of the plain text ‘B’ shifted by the second letter of the key word ‘M’. This results in the character ‘O’ as the second letter in the encrypted phase, and so on. In each approach, if the resulting integer is greater than 95 (the top of our range), the integer is “wrapped around” so that it stays within the specified range. The result is “DOVGHSZ”.

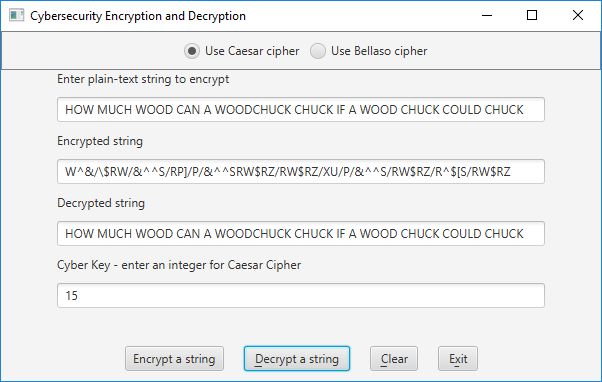
Your program will implement several methods that are specified in the file “CryptoManager.java”. A Graphical User Interface is provided, as well as a test file, that you should use to make sure your methods work correctly. Be sure to follow the naming exactly, as the tests will not work otherwise.

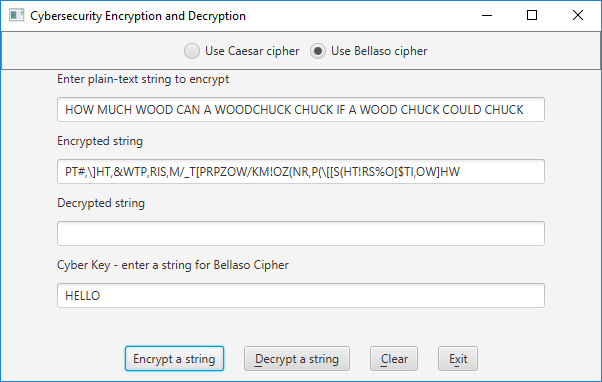
There are several features of Java that we have not yet covered in class. Just follow the syntax specified in this document and in the file CryptoManager.java. First, the required methods are “static”, which just means that they are available from the class even if an instance has not been created. To call a static method, for example, “public static void myMethod();” the syntax is CryptoManager.myMethod();. Another feature that may be useful in this project is the method charAt(i) on a string, which returns a character at position i of a string (zero-based). So “thisString”.charAt(3); would return the char ‘s’.

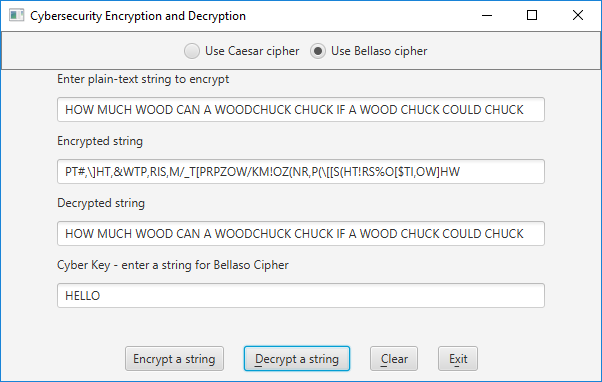
**Examples**











Submit the initial files and the resulting files in GitHub in your repo from Lab 1, as well as in Blackboard (see below)

**Deliverables / Submissions:**

**Deliverables**

**Week 1:** Design

* Turn in pseudo-code for each of the methods specified in CryptoManager.java. Your pseudo-code should be part-way between English and java. There is no need to spell out all the details of variable declaration, etc., but by the same token, the pseudo-code needs to have enough detail that a competent Java programmer could implement it. Alternately, turn in a UML class diagram specifying the methods and fields.
* Turn in a test table with at least two tests for the Caesar Cipher and two for the Bellaso Cipher. Also, include at least one string that will fail because it has characters outside the acceptable ones.

**Week 2:** Submit two compressed files containing the follow (see below):

**Deliverable format:** The deliverables will be packaged as follows. Two compressed files in the following formats:

LastNameFirstName\_Assignment3.zip (a compressed file containing the following)

* + Final design
  + Learning Experience
  + Screen Shot of files in GitHub
  + src (a directory) *contains your (.java) files*
    - File1.java (example)
    - File2.java (example)
    - File\_Test.java (example)

LastNameFirstName\_Assignment3\_**Moss**.zip (a compressed file containing only .java files)

*NO FOLDERS!!*

File1.java (example)

File2.java (example)

**Grading Rubric**

**CMSC 203 Project #3**

Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**TESTING**

Passes public tests (provided) without runtime exceptions thrown 25 pts \_\_\_\_\_

Passes private instructor tests 75 pts \_\_\_\_\_

Possible Sub-total 100 pts\_\_\_\_\_

**REQUIREMENTS** (Subtracts from TESTING total)

**Documentation:**

Documentation within source code is missing or incorrect - 10 pts \_\_\_\_\_

Description of what class does is missing

Author’s Name is missing

Methods not commented

MOSS files were missing -5 pts

Learning Experience - 10 pts \_\_\_\_\_

In 3+ paragraphs, highlight your lessons learned and learning experience from working on this project. What have you learned? What did you struggle with? What would you do differently on your next project? What parts of the project were you successful with, and what parts (if any) were you not successful with?

**Programming Style:**

Incorrect use of indentation, naming conventions, etc. (see coding/style standards) - 15 pts \_\_\_\_\_

**Design:**

Data Manager – CryptoManager -20 pts \_\_\_\_\_

1. Methods do not follow provided instructions (above)

2. Class interacts directly with user

Possible decrements: -60 pts \_\_\_\_\_

Possible total grade: 100 pts \_\_\_\_\_