**CMSC203**

**Assignment #3**

In [cryptography](https://en.wikipedia.org/wiki/Cryptography" \o "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" \o "Cipher) machines with [rotor](https://en.wikipedia.org/wiki/Rotor_machine" \o "Rotor machine) [scramblers](https://en.wikipedia.org/wiki/Scrambler" \o "Scrambler) that became [Nazi Germany](https://en.wikipedia.org/wiki/Nazi_Germany" \o "Nazi Germany)'s principal [crypto-system](https://en.wikipedia.org/wiki/Cryptography" \o "Cryptography). It was broken by the [Polish General Staff's](https://en.wikipedia.org/wiki/Polish_General_Staff" \o "Polish General Staff) [Cipher Bureau](https://en.wikipedia.org/wiki/Biuro_Szyfr%C3%B3w" \o "Biuro Szyfrów) 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 covered by this assignment**

* Using loops
* String and character processing
* ASCII codes

**Classes**

**Data Manager class – CryptoManager**

* 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.

The methods are described below.

* + public static boolean **isStringInBounds** (String plainText)
* This method determines if a string is within the allowable bounds of ASCII codes according to the LOWER\_RANGE and UPPER\_RANGE 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 **caesarEncryption** (String plainText, int key)
* This method encrypts a string according to the Caesar Cipher.
* The parameter plainText is an uppercase string to be encrypted.
* The parameter integer key specifies an offset and each character in plainText is replaced by the character the specified distance away from it.
* The method returns the encrypted string.
* If the plainText is not in bounds, the method returns:
  + The selected string is not in bounds, Try again.
* public static String **caesarDecryption** (String encryptedText, int key)
* This method decrypts a string according to the Caesar Cipher.
* This is the inverse of the caesarEncryption method.
* The parameter encryptedText is the encrypted string to be decrypted, and key is the integer used to encrypt the original text.
* The integer key specifies an offset and each character in encryptedText is replaced by the character "offset" characters before it.
* The method returns the original plain text string.
* public static String **bellasoDecryption** (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 **bellasoDecryption**(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 bellasoDecryption method.
* The parameter encryptedText is the encrypted string to be decrypted,
* The parameter 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 – FXDriver and FXMainPane**

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.

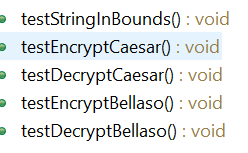
**Do not modify the GUI.**

**JUnit Test**

Two JUnit test files have been provided; CryptoManagerGFATest and CryptoManagerTestPublic.

Once your methods are implemented, run each JUnit test. Ensure that the JUnit tests all succeed.

You must create another JUnit test file, named CryptoManagerTestStudent to test every public method of the CryptoManager class, except setUp and tearDown methods.



You can take a look at provided CryptoManagerTestPublic and create similar test cases, however your test string values must be different.

**Assignment Details**

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.

## **Caesar Cipher:**

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.

## **Giovan Battista Bellaso:**

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.

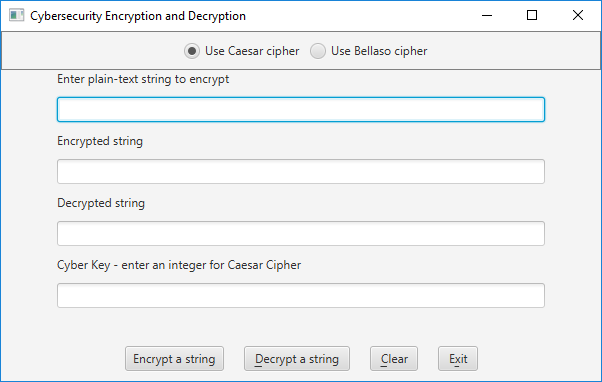
So, for the string ABCDEFG and the key word CMSC:

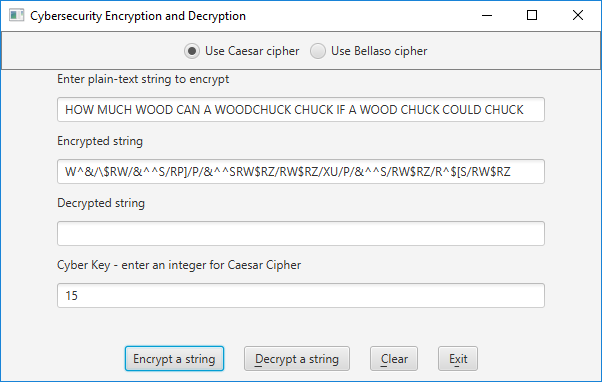
* The key word is first extended to the length of the string, i.e., CMSCCMS.
* Then A is replaced by ‘A’ offset by ’C’, i.e., ASCII 65+67=132. The range of the characters is also specified, and again we’ll say ‘ ‘ to ‘\_’ (i.e., ASCII 32 to ASCII 95). The range is then 95-32+1=64. In our example, the 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”.

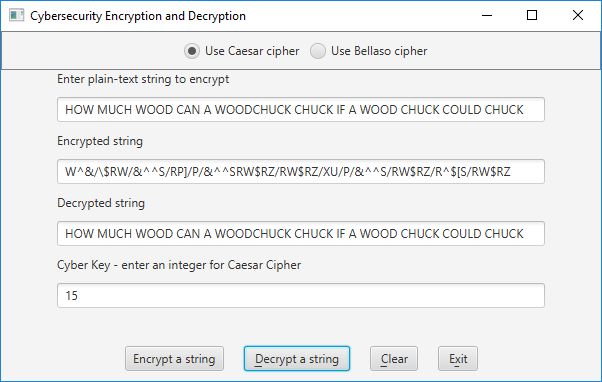
Your program will implement several methods that are specified in the file “CryptoManager.java”. A Graphical User Interface is provided, as well as a test file, that you should use to make sure your methods work correctly. **Be sure to follow the naming exactly, as the tests will not work otherwise**.

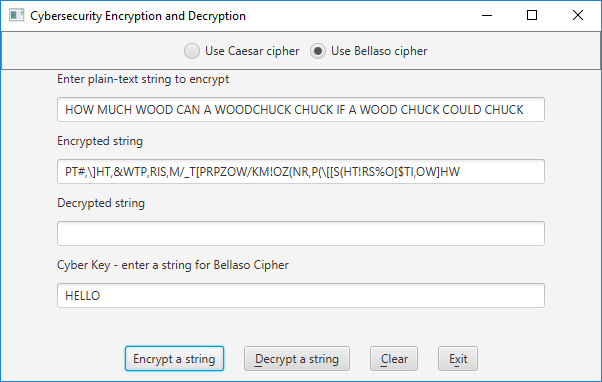
There are several features of Java that we have not yet covered in class. Just follow the syntax specified in this document and in the file CryptoManager.java. First, the required methods are “static”, which just means that they are available from the class even if an instance has not been created. To 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’.

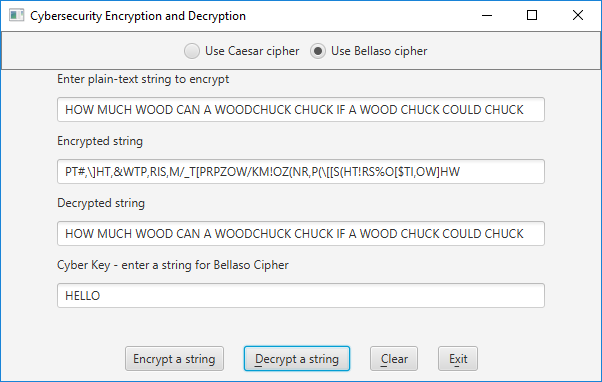
**Examples**











**Deliverables**

**Deliverables / Submissions and Deliverable format:**

* The Java application must compile and run correctly, otherwise project grade will be zero.
* The detailed grading rubric is provided in the assignment rubric excel file.
* Your source code should contain proper indentation and documentation.
* Documentation within a source code should include
  + additional Comments to clarify a code, if needed
  + class description comments at the top of each program containing the course name, the project number, your name, the date, and platform/compiler that you used to develop the project, for example:

/\*

 \* Class: CMSC203

 \* Instructor:

 \* Description: (Give a brief description for each Class)

 \* Due: MM/DD/YYYY

\* Platform/compiler:

 \* I pledge that I have completed the programming

\* assignment independently. I have not copied the code

\* from a student or any source. I have not given my code

\* to any student.

   Print your Name here: \_\_\_\_\_\_\_\_\_\_

\*/

**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.

**Implementation**

**Note**: Only submit the files that are created/modified by per requirement. DO NOT submit the files that are already provided for you.

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

* **FirstInitialLastName\_Assignment3\_Complete.zip**, a compressed file in the zip format, with the following:
* src folder*:*
  + CryptoManager.java
  + CryptoManagerTestStudent.java
  + Word document that includes (use provided template):
    - 1. Pseudocode for each of the methods specified in CryptoManager.java.
      2. Screenshots:
         1. Screen snapshots of outputs from Eclipse based on your Test Plan
         2. Screen shot of src folder files in your GitHub repository
      3. Lessons Learned: Provide answers to the questions listed below:
         1. Write about your Learning Experience, highlighting your lessons learned and learning experience from working on this project.
         2. What have you learned?
         3. What did you struggle with?
    - **FirstInitialLastName\_Assignment4\_JavaFiles.zip**, a compressed file containing one or more Java files **(This folder SHOULD NOT contain any folders and it SHOULD contain Java source file only** **that are created/modified by you per requirement.)**
  + CryptoManager.java
  + CryptoManagerTestStudent.java