Assignment 3

CMSC203

Spring 2018

Concepts tested by this program:

Using loops

String and character processing

ASCII codes

Cryptography

**General**

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 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 ‘ ‘ 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=63 from 120, giving 57, which in ASCII is the character ‘9’. 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 may 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 + 1). The range is then 95-32+1=64. In our case, 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’.

It is strongly suggested that you implement the project in the order stated below, so that one set of work will build on another.

## **Specifications**

.

**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.
* 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.
* The design documents are due in one week.

**Data Manager – CryptoManager.java**

* Implement each of the methods specified in this file. The version 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 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); E

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**

* 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 methods 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.

**Testing**

* Once your methods are implemented, run the test driver. Ensure that the driver file results in the following output:

"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

* Include CryptoTest.java with the rest of your submission.

**Deliverables / Submissions:**

**Week 1:** Design – pseudo-code and a test diagram.

**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\_Assignment2.zip (a compressed file containing the following)

* + src (a directory) *contains your (.java) files*
    - File1.java (example)
    - File2.java (example)
    - File\_Test.java (example)

LastNameFirstName\_Assignment6\_**Moss**.zip (a compressed file containing only .java files)

*NO FOLDERS!!*

File1.java (example)

File2.java (example)

**Grading Rubric**

**CMSC 203 Project #3**

**PROGRAMMING** (100 pts)

Compiles 40 pts \_\_\_\_\_

Accuracy

Passes test driver (provided) 10 pts \_\_\_\_\_

Passes student tests 10 pts \_\_\_\_\_

Passes private instructor tests 10 pts \_\_\_\_\_

Execution: runs without errors (either run-time or logic errors) 30 pts \_\_\_\_\_

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

**REQUIREMENTS** (Subtracts from Programming total)

**Documentation:**

Documentation within source code is missing or incorrect -5 pts \_\_\_\_\_

Description of what class does is missing

Author’s Name, @author, is missing

Methods not commented properly using Javadoc @param, @return

**Programming Style:**

Incorrect use of indentation, statements, structures -5 pts \_\_\_\_\_

**Design:**

Design does not match implementation -10 pts \_\_\_\_\_

**Implementation:**

Data Manager – CryptoManager

* stringInBounds method not implemented -8 pts
* encryptCaesar method not implemented -8 pts
* encryptBellaso method not implemented -8 pts
* decryptCaesar method not implemented -8 pts
* decryptBellaso method not implemented -8 pts

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

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