**Assignment 3**

Josue Ponce

Montgomery College

3/2/2018

Author’s Note

This project report was prepared for CMSC203 CRN #30672, taught by professor Ahmed Tarek

**Table of Contents**

Copy of CryptoManager.java Code…………………………………………………………….....2

CryptoManager Pseudocode………………………………………………………………………7

CryptoManager Test Using CryptoTest.Java……………………………………………………10

Junit Tests for Crypto Manager………………………………………………………………….11

Copy of CryptoManagerTest.java Code…………………………………………………………12

Test Cases for GUI……………………………………………………………………………….15

**Copy of CryptoManager.java Code**

/\*\*

\* CryptoManger class encrypts and decrypts strings using either a caesar Cipher or

\* Bellaso cipher. User gets to choose which method they would like to use for encryption

\* The GUI handles capitalization of strings and user selection prompt.

\*

\* @author Josue Ponce

\* version 1.0

\*/

**public** **class** **CryptoManager** {

/\*\* Attribute LOWER\_BOUND holds lower allowable bounds of ASCII codes. \*/

**private** **static** **final** **char** LOWER\_BOUND = ' ';

/\*\* Attribute UPPER\_BOUND holds upper allowable bounds of ASCII codes. \*/

**private** **static** **final** **char** UPPER\_BOUND = '\_';

/\*\* Attribute RANGE holds UPPER\_BOUND SUBTRACTED BY LOWER\_BOUND. \*/

**private** **static** **final** **int** RANGE = (**int**) UPPER\_BOUND - (**int**) LOWER\_BOUND + **1**;

/\*\*

\* This method determines if a string is within the allowable bounds of ASCII codes

\* according to the LOWER\_BOUND and UPPER\_BOUND characters

\* @param plainText a string to be encrypted, if it is within the allowable bounds

\* @return true if all characters within the string are within the allowable bounds, false if any characters are outside.

\*/

**public** **static** **boolean** **stringInBounds**(String plainText) {

**boolean** neko = **true**;

**for** (**int** i = **0**; i < plainText.length(); i++) {

**if** (!(charInBounds(plainText.charAt(i)))) {

neko = **false**;

}

}

**return** neko;

}

/\*\* This method determines if a characters are within the allowable bounds of ASCII codes.

\* according to the LOWER\_BOUND and UPPER\_BOUND characters

\* @param charAt is a character that will be encrypted if it's in the allowable bonds and false if character is outside.

\* @return true if characters are greater or equal to the LOWER BOUND and Less than or equal to the UPPER\_BOUND.

\*/

**private** **static** **boolean** **charInBounds**(**char** charAt) {

**if** (charAt >= LOWER\_BOUND && charAt <= UPPER\_BOUND)

**return** **true**;

**else**

**return** **false**;

}

/\*\*

\* Encrypts a string according to the Caesar Cipher. The integer key specifies an offset

\* and each character in plainText is replaced by the character \"offset\" away from it

\* @param plainText an uppercase string to be encrypted.

\* @param key an integer that specifies the offset of each character

\* @return the encrypted string

\*/

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

String encryptedText = "";

**if** (stringInBounds(plainText)) {

**for** (**int** i = **0**; i < plainText.length(); i++) {

**char** t = plainText.charAt(i);

**int** encryptedKey = ((**int**) t + key);

**while** (encryptedKey < LOWER\_BOUND || encryptedKey > UPPER\_BOUND) {

encryptedKey += RANGE;

}

encryptedText += (**char**) encryptedKey;

}

}

**return** encryptedText;

}

/\*\*

\* 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.

\* @param encryptedText an encrypted string to be decrypted.

\* @param key an integer that specifies the offset of each character

\* @return the plain text string

\*/

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

String decryptText = "";

**for** (**int** i = **0**; i < encryptedText.length(); i++) {

**char** t = encryptedText.charAt(i);

**int** decryptedNum = ((**int**) t - key);

**while** (decryptedNum < LOWER\_BOUND || decryptedNum > UPPER\_BOUND) {

decryptedNum += RANGE;

}

decryptText += (**char**) decryptedNum;

}

**return** decryptText;

}

/\*\*

\* Encrypts a string according 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

\* @param plainText an uppercase string to be encrypted.

\* @param bellasoStr an uppercase string that specifies the offsets, character by character.

\* @return the encrypted string

\*/

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

String encryptedText = "";

**int** bellasoLength = bellasoStr.length();

**for** (**int** i = **0**; i < plainText.length(); i++) {

**char** thisChar = plainText.charAt(i);

**int** encryptedKeyword = ((**int**) thisChar + (**int**) bellasoStr.charAt(i % bellasoLength));

**while** (encryptedKeyword > (**int**) UPPER\_BOUND) {

encryptedKeyword -= RANGE;

}

encryptedText += (**char**) encryptedKeyword;

}

**return** encryptedText;

}

/\*\*

\* Decrypts a string according 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.

\* @param encryptedText an uppercase string to be encrypted.

\* @param bellasoStr an uppercase string that specifies the offsets, character by character.

\* @return the decrypted string

\*/

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

String decryptedText = "";

**int** bellasoStrlength = bellasoStr.length();

**for** (**int** i = **0**; i < encryptedText.length(); i++) {

**char** t = encryptedText.charAt(i);

**int** charIntEncrypt = ((**int**) t - (**int**) bellasoStr.charAt(i % bellasoStrlength));

**while** (charIntEncrypt < LOWER\_BOUND) {

charIntEncrypt += RANGE;

}

decryptedText += (**char**) charIntEncrypt;

}

**return** decryptedText;

}

}

**CryptoManager Pseudocode**

initialize private static final char LOWER\_BOUND to ' '

Initialize private static final char UPPER\_BOUND to '\_'

Initialize private static final int range to UPPER\_BOUND - LOWER\_BOUND +1 converted to integers.

public static boolean stringInBounds(String plainText) {

Initialize boolean neko and set it to true

for (int i = 0; i < plainText.length(); i++)

If (!(charInBounds(plainText.charAt(i))))

neko = false;

END IF

END FOR

return neko

}

private static boolean charInBounds(char charAt)

{

if (charAt >= LOWER\_BOUND && charAt <= UPPER\_BOUND)

return true

else

return false

END IF

}

public static String encryptCaesar(String plainText, int key)

{

Intialize string encryptedText to " "

if (stringInBounds(plainText))

for (int i = 0; i < plainText.length(); i++)

initialize char t and set it to plaintext.charAt(i)

initialize integer encrypted key and set it to casted ((int) t + key)

while (encryptedKey < LOWER\_BOUND || encryptedKey > UPPER\_BOUND)

encryptedKey += RANGE;

END WHILE

encryptedText += (char) encryptedKey;

END FOR

END IF

return encryptedText

}

public static String decryptCaesar(String encryptedText, int key) {

initialize decrypt text to “ ”

for (int i = 0; i < encryptedText.length(); i++)

Initialize char t and set it to encryptedText chatAt(i)

Initialize decrypted num and set it ((int) t-key)

while (decryptedNum < LOWER\_BOUND || decryptedNum > UPPER\_BOUND)

decryptedNum += RANGE;

END WHILE

decryptText += (char) decryptedNum;

END FOR

END IF

return decryptText;

}

public static String encryptBellaso(String plainText, String bellasoStr) {

initialize String encryptedText to ""

initialize integer bellasoLength to bellasoStr.Length.

for (int i = 0; i < plainText.length(); i++)

initialize char meow and set it to plainText.charAt(i)

initialize encryptedKeyword and set it to ((int) meow + (int bellasoStr.charAt(i % bellasoLength))

while (encryptedKeyword > (int) UPPER\_BOUND)

encryptedKeyword -= RANGE;

END WHILE

encryptedText += (char) encryptedKeyword;

END FOR

return encryptedText;

}

public static String decryptBellaso(String encryptedText, String bellasoStr) {

Initialize string decrypted text and set it to “ “

Initialize integer bellasoSrtlength and set it to bellasoStr.Length().

for (int i = 0; i < encryptedText.length(); i++)

Initialize char T and set it to encrytedText.charAt(i).

Initialize integer charIntEncrypt and set it to ((int) t - (int) bellasoStr.charAt(i % bellasoStrlength)).

while (charIntEncrypt < LOWER\_BOUND)

charIntEncrypt += RANGE;

END WHILE

decryptedText += (char) charIntEncrypt;

END FOR

return decryptedText;

}

**CryptoManager Test Using CryptoTest.Java**

Crypto manager was tested using a separate program called crypto test. Crypto test is a test driver that ensures all the methods implemented in crypto manager worked as intended. The highlight results below represent the sample output of the program. The non-highlighted sections represent what the program’s expected output should be. The program passed the crypto test because the expected output matched the program’s sample output which means all the methods have been implemented correctly.

**---------------------------------------------------------------------------------------------------------------------**

"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

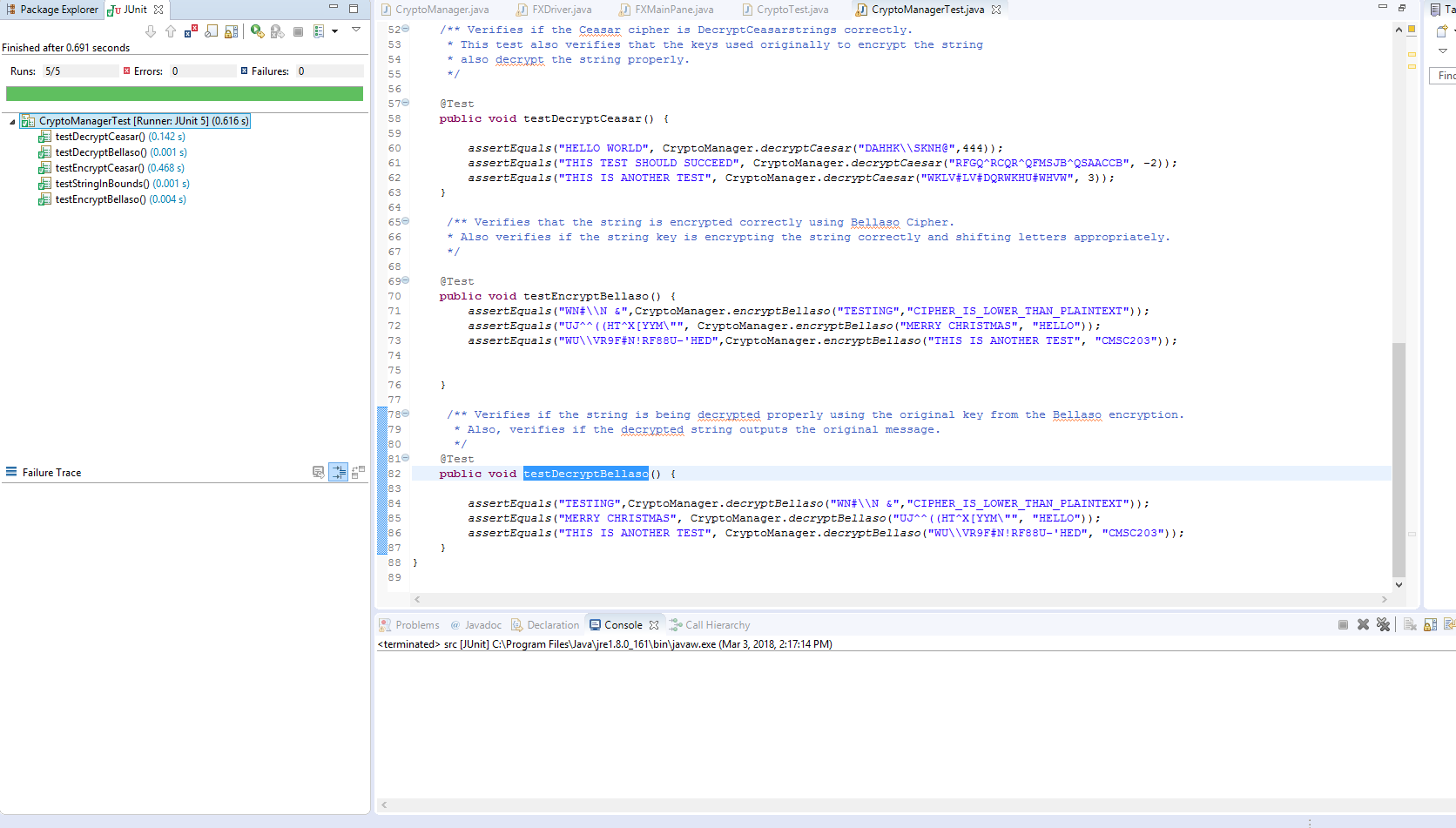
Caesar decryption of " WKLV#LV#DQRWKHU#WHVW" should return "THIS IS ANOTHER TEST": THIS IS ANOTHER TEST

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

--------------------------------------------------------------------------------------------------------------------

**Junit Test for Crypto Manager**

Junit tests were created to check the application’s functionality and ensure it runs and all the methods implemented in crypto manager worked as intended. Assertion methods were used to test the following methods in crypto manager: StringInBounds(), encryptCeasar(), decryptCeasar(), encryptBellaso(), and decryptBellaso() .After running the junit tests, all of the results returned green which means the application is functioning as intended, the methods were implemented correctly, and outputting the correct results. The screenshot down below shows all Junit tests passing.



*Note.* Screenshot of all junit test cases passing with no failure.

**Copy of CryptoManagerTest.java Code**

**import** **static** org.junit.Assert.\*;

**import** **org.junit.jupiter.api.AfterAll**;

**import** **org.junit.jupiter.api.AfterEach**;

**import** **org.junit.jupiter.api.BeforeAll**;

**import** **org.junit.jupiter.api.BeforeEach**;

**import** **org.junit.jupiter.api.Test**;

/\*\*

\* CryptoManagerTest ensures that the methods in Crypto Manager work as intended.

\* @author Josue Ponce

\* version 1.0

\*/

**class** **CryptoManagerTest** {

**@BeforeAll**

**static** **void** **setUp**() **throws** Exception {

}

**@AfterAll**

**static** **void** **tearDown**() **throws** Exception {

}

/\*\* Verifies if a string is within the allowable bounds of ASCII codes .

\* For the first assert the method will return true because it is inbounds.

\* if the string is out of range it will return false.

\* if the string is in lower case letters the test will fail returning false.

\*/

**@Test**

**public** **void** **testStringInBounds**() {

assertTrue(CryptoManager.stringInBounds("THIS TEST SHOULD SUCCEED"));

assertFalse(CryptoManager.stringInBounds("THIS TEST THAT SHOULD FAIL BECAUSE { IS OUTSIDE THE RANGE"));

assertFalse(CryptoManager.stringInBounds("This test should fail because of lowercase letters"));

}

/\*\* Verifies if the Ceasar cipher is encrypting strings correctly.

\* This test also verifies that the string keys are all shifting the string

\* appropriately regardless of how large the integer is or if the integer is

\* a negative or positive number.

\*/

**@Test**

**public** **void** **testEncryptCeasar**() {

assertEquals("WKLV#LV#DQRWKHU#WHVW", CryptoManager.encryptCaesar("THIS IS ANOTHER TEST", **3**));

assertEquals("RFGQ^RCQR^QFMSJB^QSAACCB", CryptoManager.encryptCaesar("THIS TEST SHOULD SUCCEED", -**2**));

assertEquals("DAHHK\\SKNH@", CryptoManager.encryptCaesar("HELLO WORLD",**444**));

}

/\*\* Verifies if the Ceasar cipher is DecryptCeasarstrings correctly.

\* This test also verifies that the keys used originally to encrypt the string

\* also decrypt the string properly.

\*/

**@Test**

**public** **void** **testDecryptCeasar**() {

assertEquals("HELLO WORLD", CryptoManager.decryptCaesar("DAHHK\\SKNH@",**444**));

assertEquals("THIS TEST SHOULD SUCCEED", CryptoManager.decryptCaesar("RFGQ^RCQR^QFMSJB^QSAACCB", -**2**));

assertEquals("THIS IS ANOTHER TEST", CryptoManager.decryptCaesar("WKLV#LV#DQRWKHU#WHVW", **3**));

}

/\*\* Verifies that the string is encrypted correctly using Bellaso Cipher.

\* Also verifies if the string key is encrypting the string correctly and shifting letters appropriately.

\*/

**@Test**

**public** **void** **testEncryptBellaso**() {

assertEquals("WN#\\N &",CryptoManager.encryptBellaso("TESTING","CIPHER\_IS\_LOWER\_THAN\_PLAINTEXT"));

assertEquals("UJ^^((HT^X[YYM\"", CryptoManager.encryptBellaso("MERRY CHRISTMAS", "HELLO"));

assertEquals("WU\\VR9F#N!RF88U-'HED",CryptoManager.encryptBellaso("THIS IS ANOTHER TEST", "CMSC203"));

}

/\*\* Verifies if the string is being decrypted properly using the original key from the Bellaso encryption.

\* Also, verifies if the decrypted string outputs the original message.

\*/

**@Test**

**public** **void** **testDecryptBellaso**() {

assertEquals("TESTING",CryptoManager.decryptBellaso("WN#\\N &","CIPHER\_IS\_LOWER\_THAN\_PLAINTEXT"));

assertEquals("MERRY CHRISTMAS", CryptoManager.decryptBellaso("UJ^^((HT^X[YYM\"", "HELLO"));

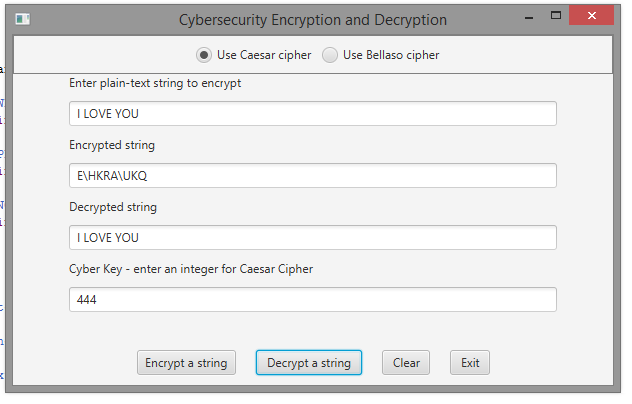
assertEquals("THIS IS ANOTHER TEST", CryptoManager.decryptBellaso("WU\\VR9F#N!RF88U-'HED", "CMSC203"));

}

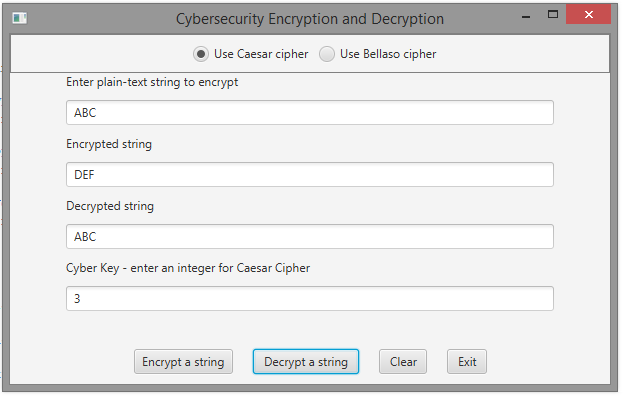
}

**Test Cases for GUI**

The following test cases were conducted to ensure that the program’s GUI was free of screen tearing or glitches. The tests conducted also ensures that the program was displaying the correct results to the user regardless of the cipher used for encryption. The following screenshots are of successful test runs. All the tests passed because there were no issues with the GUI or sample output.



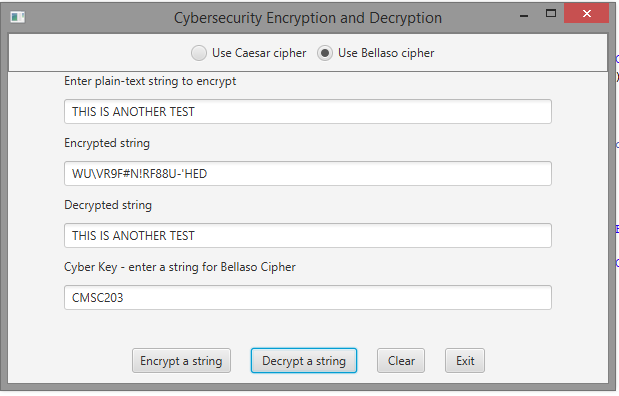
*Note.* Screenshot of successful test run of the application’s GUI using Caesar cipher.



*Note.* Screenshot of successful test run of the application’s GUI using Caesar cipher.



*Note.* Screenshot of successful test run of the application’s GUI using Bellaso cipher.



*Note.* Screenshot of successful test run of the application’s GUI using Bellaso cipher.