CS12320 Main Assignment
The Code Breaker

Ethan Swain
220029553 - ets6

Undergraduate Student
Computer Science
Aberystwyth University

March/April/May 2023

Contents

1 Introduction

This document describes the complete process of the implementation of The Code Breaker program, as outlined in the assignment brief[1]. This includes the design, development, and testing of eleven functional requirements to produce the fully functional program.

The program is executed via a command-line menu, offering the user a wide array of options to encrypt and decrypt text using three different ciphers: Caesar, Keyed Caesar, and Vigenere.

Accompanying this document is a UML Use-Case diagram, UML Class diagram, Test Table, and several screen-shots to provide evidence of testing. In addition, this document presents a detailed breakdown of the design, development, testing, and evaluation processes, displaying an insight into the creation of the program.

1.1 UML Use-Case Diagram

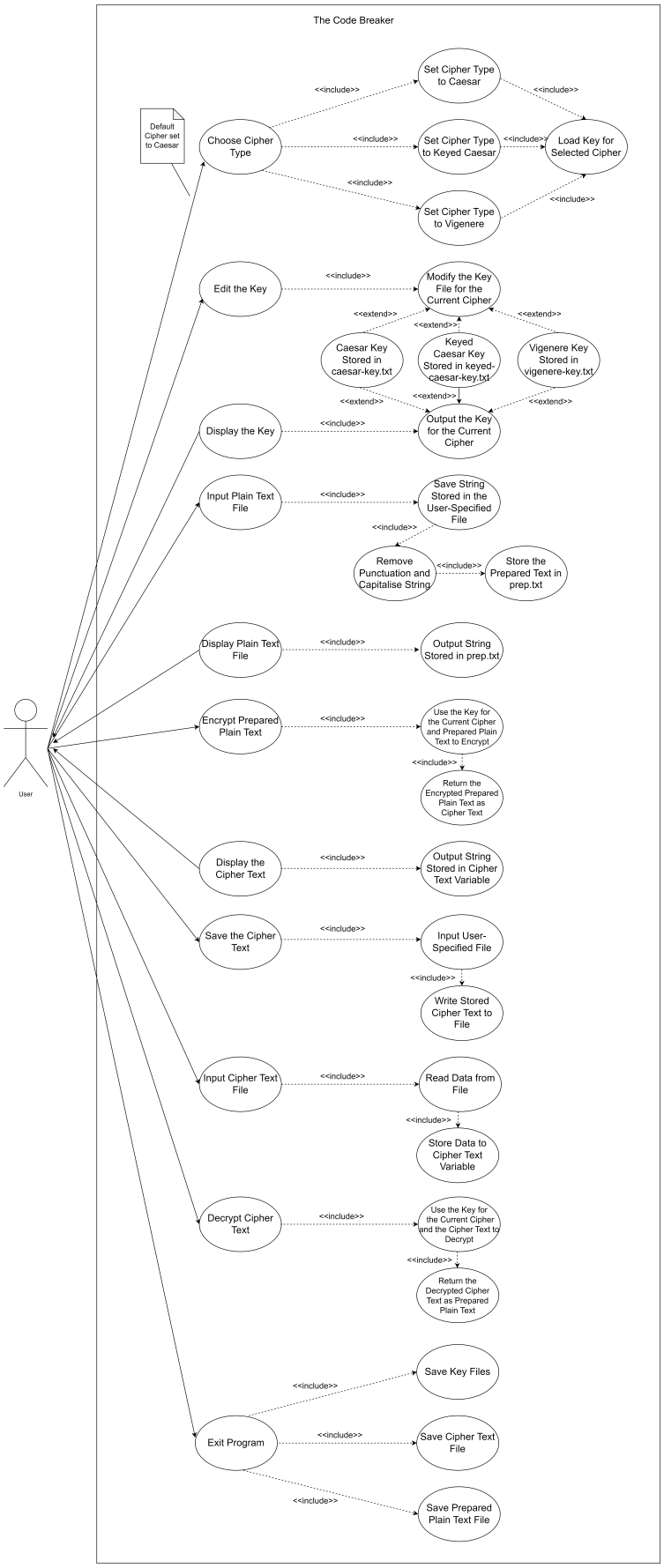

Figure 1.1: UML Use-Case Diagram presenting requirements.

2 Design

2.1 UML Class Diagram

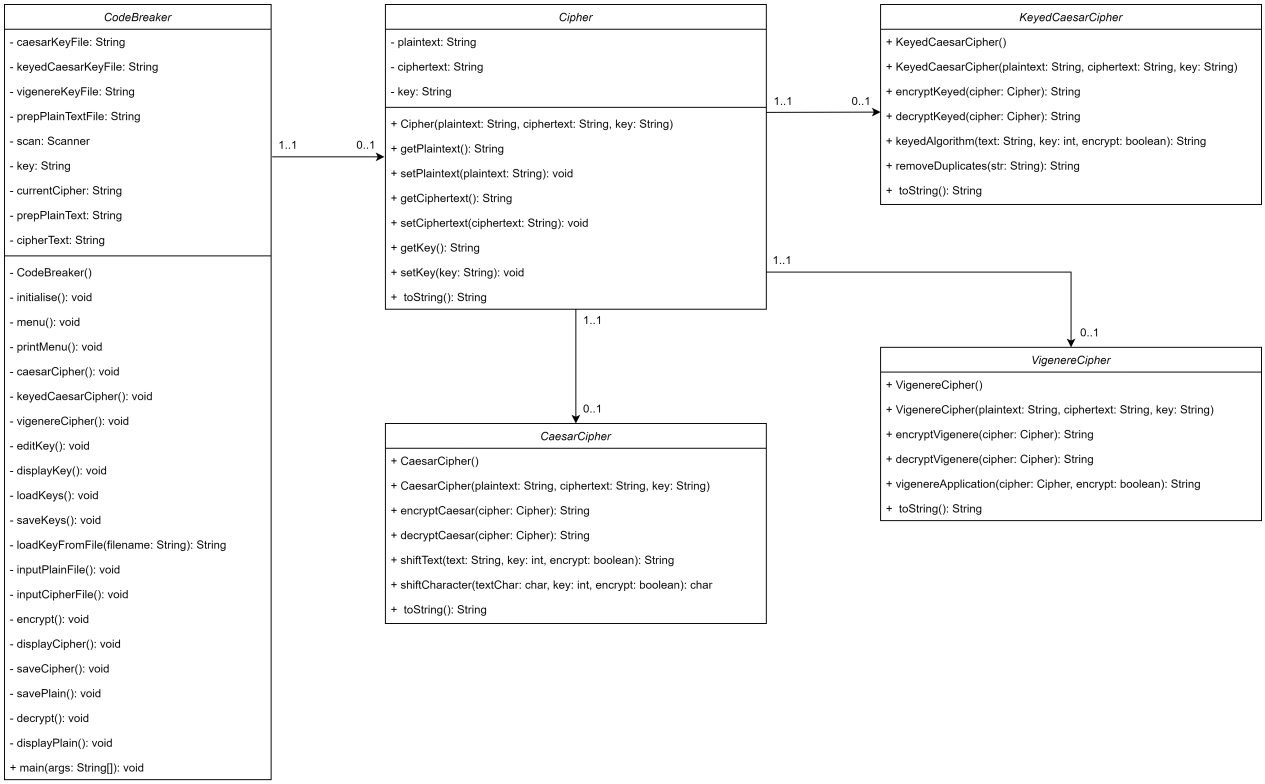

Figure 2.1: UML Class Diagram presenting relationships between classes.

2.2 Textual Class Descriptions and Relationships

2.2.1 CodeBreaker

The *CodeBreaker* class is the main class used to run the program. It incorporates an array of options that revolve around encryption and decryption processes, where each option is displayed to the user through a command-line menu. All options, apart from the encryption and decryption tasks, are implemented within the *CodeBreaker* class, while the encryption and decryption are separated to their corresponding cipher classes.

The *CodeBreaker* class maintains a one-to-many relationship with the *Cipher* class as the program encompasses multiple ciphers, each with their own unique set of keys corresponding to a specific cipher algorithm.

- *CodeBreaker()* - Constructor.
- *initialise()* - Called when the program begins, loads the essential key files.
- *menu()* - Implements a switch-case statement for each functional requirement accessible to the user. The menu is displayed through the *printMenu()* method.
- *printMenu()* - Called at the start of the *menu()* method, presents the menu options in a consistent, readable way.
- *CaesarCipher()* - Sets the chosen cipher to 'Caesar Cipher'; storing the corresponding key for any future operations. The key is loaded from the corresponding key file ("caesar-key.txt").
- *KeyedCaesarCipher()* - Sets the chosen cipher to 'Keyed Caesar Cipher'.
- *VigenereCipher()* - Sets the chosen cipher to 'Vigenere Cipher'.
- *editKey()* - Allows the user to modify the key for the chosen cipher. Validates the key's format through various error checking statements.
- *displayKey()* - Displays the key for the chosen cipher.
- *loadKeys()* - Loads the key for the chosen cipher from its corresponding file.
- *saveKeys()* - Saves and updates the edited key to the corresponding cipher key file.
- *loadKeyFromFile()* - Retrieves the key from the cipher key file by reading its contents.
- *inputPlainFile()* - Allows the user to input a plain text file name. Contents of the file are then read and converted to prepared plain text. The prepared plain text is then stored in the "prep.txt" file.
- *inputCipherFile()* - Allows the user to input a cipher text file name. Cipher text content is then read and converted to uppercase, with all punctuation removed.
- *encrypt()* - Creates a new instance of the *Cipher* class and initialises it with the prepared plain text, cipher text, and the key. Creates a new instance of each child class and passes the *cipher* object to the encryption methods depending on the chosen cipher.
- *displayCipher()* - Displays the cipher text currently stored.
- *saveCipher()* - Allows the user to input a name for a file to save the cipher text to. The cipher text is then stored in the file.
- *savePlain()* - Allows the user to input a name for a file to save the prepared plain text to. The prepared plain text is then stored in the file.
- *decrypt()* - Creates a new instance of the *Cipher* class and initialises it with the prepared plain text, cipher text, and the key. Creates a new instance of each child class and passes the *cipher* object to the decryption methods depending on the chosen cipher.
- *displayPlain()* - Displays the prepared plain text currently stored.
- *main()* - Calls the *initialise()* method and the *menu()* method.

2.2.2 Cipher

The *Cipher* class is a parent class for the three child classes for each cipher algorithm, it contains three private instance variables required for both encryption and decryption: 'plaintext', 'ciphertext', and 'key'. Each of these variables contain getters and setters, which are utilised by the child classes.

The *Cipher* class maintains a one-to-one relationship with each of its child classes; the *CaesarCipher*, *KeyedCaesarCipher*, and *VigenereCipher* classes, through inheritance. Upon instantiation of the child classes, each obtain an instance of the *Cipher* class with its defined variables.

Additionally, each of the child classes of the *Cipher* parent class have a zero-to-one relationship with the *Cipher* class, as the *Cipher* class can be associated with either zero or one instance of the child classes. Furthermore, the child classes have a one-to-one relationship with the *Cipher* class, as the child classes depend on the functionality and attributes of the *Cipher* class to execute the encryption and decryption algorithm.

- *Cipher()* - Constructor containing three private instance variables required for the different encryption and decryption algorithms.
- *getPlaintext()* - Getters and setters.
- *setPlaintext()*
- *getCiphertext()*
- *setCiphertext()*
- *getKey()*
- *setKey()*
- *toString()* - Uses StringBuilder to output a string containing each instance variable.

2.2.3 CaesarCipher

The *CaesarCipher* class inherits the *Cipher* parent class. The encryption and decryption algorithm requires the prepared plain text, cipher text, and numeric key instance variables. The algorithm for encryption and decryption comprises of two method: *shiftText()* and *shiftCharacter()*. These methods work to shift and manipulate each character in either the plain text or the cipher text.

- *CaesarCipher()* - Constructor that takes three String arguments. Calls the constructor of the *Cipher* superclass and calls the three setter methods, with the corresponding arguments.
- *encryptCaesar()* - Performs a Caesar Cipher encryption on a prepared plain text message using a key obtained from the *cipher* object.
- *decryptCaesar()* - Performs a Caesar Cipher decryption on a cipher text message using a key obtained from the *cipher* object.
- *shiftText()* - Iterates through each character in the prepared plain text string and passes the character to the *shiftCharacter()* method.
- *shiftCharacter()* - Shifts each character of a prepared plain text string based on a key and a boolean value indicating whther to encrypt or decrypt.
- *toString()*

2.2.4 KeyedCaesarCipher

The *keyedCaesarCipher* class inherits the *Cipher* parent class and the three instance variables required for the encryption and decryption algorithm. The key is composed of two parts: a numeric shift and a textual shift, which are concatenated together (i.e. in the format '12TEST').

- *KeyedCaesarCipher()* - Constructor that takes three String arguments. Calls the constructor of the *Cipher* superclass and calls the three setter methods, with the corresponding arguments.
- *encryptKeyed()* - Performs a Keyed Caesar Cipher encryption on a prepared plain text message using a key obtained from the *cipher* object.
- *decryptKeyed()* - Performs a Keyed Caesar Cipher decryption on a cipher text message using a key obtained from the *cipher* object.
- *keyedAlgorithm()* - The algorithm for encryption and decryption. Takes in a text string to be encrypted or decrypted, a key string to be used for substitution, and a boolean value indicating whether to encrypt or decrypt the text.
- *removeDuplicates()* - Removes any duplicate characters from the textual key so that only the first occurrence of each character is used.
- *toString()*

2.2.5 VigenereCipher

The *vigenereCipher* class inherits the *Cipher* parent class and the three instance variables required for the encryption and decryption algorithm. The key is composed of a textual shift which decides the shifted alphabet to use.

- *vigenereCipher()* - Constructor that takes three String arguments. Calls the constructor of the *Cipher* superclass and calls the three setter methods, with the corresponding arguments.
- *encryptVigenere()* - Performs a Vigenere Cipher encryption on a plain text message using a key obtained from the *cipher* object.
- *decryptVigenere()* - Performs a Vigenere Cipher decryption on a cipher text message using a key obtained from the *cipher* object.
- *vigenereAlgorithm()* - The algorithm for encryption and decryption. Takes in a text string to be encrypted or decrypted, a key string to be used for substitution, and a boolean value indicating whether to encrypt or decrypt the text.
- *toString()*

2.3 Pseudo-code Example

```
FUNCTION vigenereAlgorithm (Cipher cipher, boolean encrypt)

    IF encrypt == true THEN
        text = cipher.getPlaintext()
    ELSE
        text = cipher.getCiphertext()
    END IF

    keyword = cipher.getKey()
    keyword = keyword.toUpperCase()
    text = text.toUpperCase()
    keywordLength = keyword.length()
    textLength = text.length()
    output = array of textLength characters
    j = 0

    FOR i = 0 to textLength -1 DO
        inputChar = text.charAt(i)
        IF inputChar != a letter THEN
            output[i] = inputChar
            CONTINUE
        END IF
        keywordCharIndex = j % keywordLength
        keywordChar = keyword.charAt(keywordCharIndex)
        shiftAmount = keywordChar - 65
        IF encrypt == true THEN
            outputChar = (inputChar + shiftAmount - 65) % 26 + 65
        ELSE
            outputChar = (inputChar - shiftAmount - 65 + 26) % 26 + 65
        END IF
        output[i] = outputChar
        j++
    END FOR

    result = new String(output)

    IF encrypt is true THEN
        cipher.setCiphertext(result)
        RETURN cipher.getCiphertext()
    ELSE
        cipher.setPlaintext(result)
        RETURN cipher.getPlaintext()
    END IF

END FUNCTION
```

Figure 2.2: Pseudo-code example presenting the most complex algorithm.

3 Testing

3.1 Test Table

ID	Requirement	Description	Inputs	Expected Outputs	Pass /Fail	Comments
A1.1	NFR1	When the program starts, a command-line menu is displayed. The menu must respond with the correct output when the user enters the corresponding option.	Run program.	The chosen cipher is set to "Caesar Cipher" as default and the menu is displayed to the user.	P	The key files for each cipher are loaded.
A2.1	FR1	The user can change the chosen cipher from the main menu. By default, the chosen cipher is set to "Caesar Cipher". When a different cipher is selected, the corresponding key is loaded. Menu operations will then perform to the chosen cipher.	Enter '2'.	The chosen cipher is set to "Keyed Caesar Cipher". The chosen cipher is displayed at the top of the menu.	P	
			Enter '3'.	The chosen cipher is set to "Vigenere Cipher". The chosen cipher is displayed at the top of the menu.	P	
			Enter 'H'.	An error messaage is output to the console and menu is reloaded.	P	
A3.1	FR2	The user can modify the key for the chosen cipher. The key must follow the format for the corresponding cipher.	Enter '12' when the chosen cipher is set to "Caesar".	The key is updated to store the value '12'.	P	
			Enter 'LEMON' when the chosen cipher is set to "Caesar"	An error is output to the console and user is prompted to enter an integer.	P	
			Enter '4TEST' when the chosen cipher is set to "Keyed Caesar"	The key is updated to store the value '4TEST' with '4' being the numeric shift and 'TEST' being the textual key.	P	The program was initially unable to correctly separate and read the numeric shift and textual key, and therefore output an error prompting the user to re-enter the key with the correct format. I resolved this issue in my code by modifying

						the algorithm for detecting and separating the integer from the string.
			Enter 'LE57TEST' when the chosen cipher is set to "Keyed Caesar"	An error is output to the console and the user is prompted to enter the key in the correct format.	P	Once I rectified the issue (as mentioned above), I performed the error checking, which worked as intended.
			Enter 'LEMON' when the chosen cipher is set to "Vigenere"	The key is updated to store the value 'LEMON'.	P	
			Enter '53252' when the chosen cipher is set to "Vigenere"	An error is output to the console and the user is prompted to enter a string of letters.	P	
A3.2	FR3	The key for the chosen cipher is displayed to the console. Each key is stored in the key file corresponding to the cipher.	Enter '5' on the menu.	The key corresponding to the chosen cipher is output.	P	
A4.1	FR4	The user is prompted to enter a plain text file. The plain text contained within the file is stripped of punctuation and text is capitalised. This creates prepared plain text which is stored in the file "prep.txt".	Enter an existing file (e.g. "plaintext-test.txt").	The file is loaded, and the content is converted to uppercase, and punctuation is removed.	P	
			Enter a non-existent file (e.g. "invalid-text.txt").	An error message is output to the console and the user is prompted to enter a valid file.	P	
A4.2	FR5	The prepared text (stored in the "prep.txt" file) is output to the console. The text output is in uppercase and contains no punctuation.	The file input contains the text "tEsT h=l!/,a".	The string "TEST HLA" is output. All punctuation has been removed and the text is in uppercase.	P	
			File input contains text with abnormal	The string "GI HJEDA" is output.	P	Initially, the abnormal characters were not being removed, this is because my code was

			characters “g [illegible] i*£ [illegible] hJEda”			looking for a list of specific punctuation and not looking for all non-letter characters. Therefore, not all non-letter characters were being removed. I modified my code so that only characters from a-z, A-Z and spaces can be entered.
A5.1	FR6	The prepared plain text is encrypted using the chosen cipher.	Caesar cipher encryption with prepared plain text containing the string “CAESAR TEST” is entered with a key of ‘28’.	The cipher text “ECGUCT VGUV” is produced.	P	When I first tested this, a random string of characters was output. I then slightly modified my algorithm so that if the value of the key exceeded the length of the alphabet, then the alphabet would wrap around to the start. This led me to modify the algorithms for the other ciphers to do the same.
			Keyed Caesar cipher encryption with prepared plain text containing the string “KEYED TEST” is entered with a numeric shift of ‘4’ and textual shift of “HELLO”.	The cipher text “NFLFD XFWX” is produced.	P	The code was initially outputting the string “QBLBJ XBWX”. This was because the keyed alphabet generated was incorrect, some letters were missing or in the incorrect place. I rewrote the algorithm so that the correct keyed alphabet was produced and used, the code then produced the correct output.
			Vigenere cipher encryption with prepared plain text containing	The cipher text “CMRPBLVP ESZX” is produced.	P	

			the string "VIGENERE TEST" is entered with a key of "HELLO".			
A6.1	FR7	The cipher text is output to the console.	The cipher text contains the string "NMBVRC XQGG".	The cipher text "NMBVRC XQGG" is output.	P	
			The cipher text contains the string "LQRE RE G LMEL".	The cipher text "LQRE RE G LMEL" is output.	P	
A6.2	FR8	The user is prompted to enter a file name to save the cipher text to. The program then creates a file with this name containing the cipher text content.	Enter a filename (e.g. "cipher-text.txt").	A new file is created called "cipher-text.txt" and the cipher text is written to it.	P	
			Enter a filename of a file that already exists (e.g. "existing-cipher.txt").	An error message is output to the console and the user is prompted to enter a different file name.	P	
A6.3	FR9	The user is prompted to enter a cipher text file. The cipher text contained within the file can then be displayed or decrypted.	Enter a file that exists (e.g. "ciphertext-test.txt").	The content of the file is loaded. The user can then display or decrypt this.	P	
			Enter a file that doesn't exist (e.g. "invalid-text.txt".	An error message is output to the console and the user is prompted to enter a valid file.	P	
A7.1	FR10	The cipher text is decrypted using the chosen cipher.	Caesar cipher decryption with cipher text containing the string "PQODKBF OMQEMD" is entered with a key of '12'.	The plain text "DECRYPT CAESAR" is produced.	P	The decryption algorithm for the Caesar cipher converts the positive key into a negative number (e.g. a key of '12' is converted to '-12').
			Keyed Caesar cipher decryption	The plain text "DECRYPT KEYED" is produced.	P	

			with cipher text containing the string "KLJTFYS RLFLK" is entered with a numeric shift of '9' and textual shift of "TEST".			
			Vigenere cipher decryption with cipher text containing the string "GIOFQSX HWYHRQFW" is entered with a key of "DEMOS".	The plain text "DECRYPT VIGENERE" is produced.	P	
A8.1	FR11	The program exits.	Enter 'Q' or 'q' from the menu.	The key files are saved. The program then ends.	P	

Figure 3.1: Test table displaying each functional requirement.

3.2 Screenshots

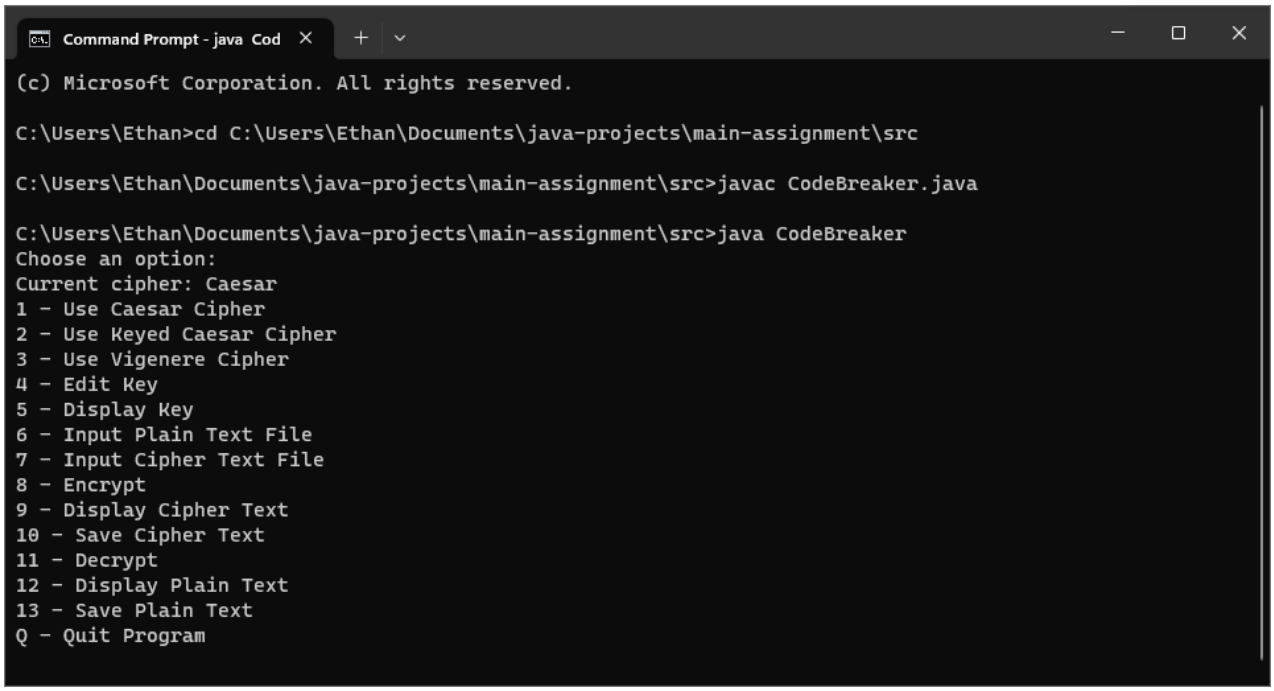

Figure 4.1: Program being launched from the command-line (cmd).

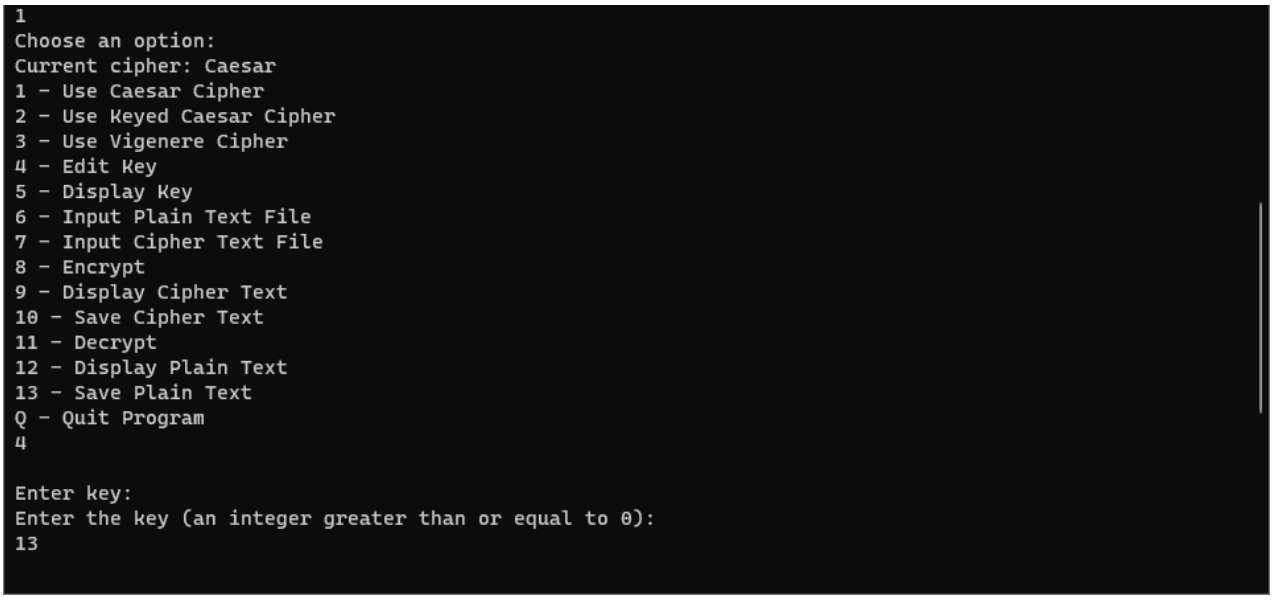

Figure 5.1.1: Choosing the Caesar Cipher and setting the key to be used.

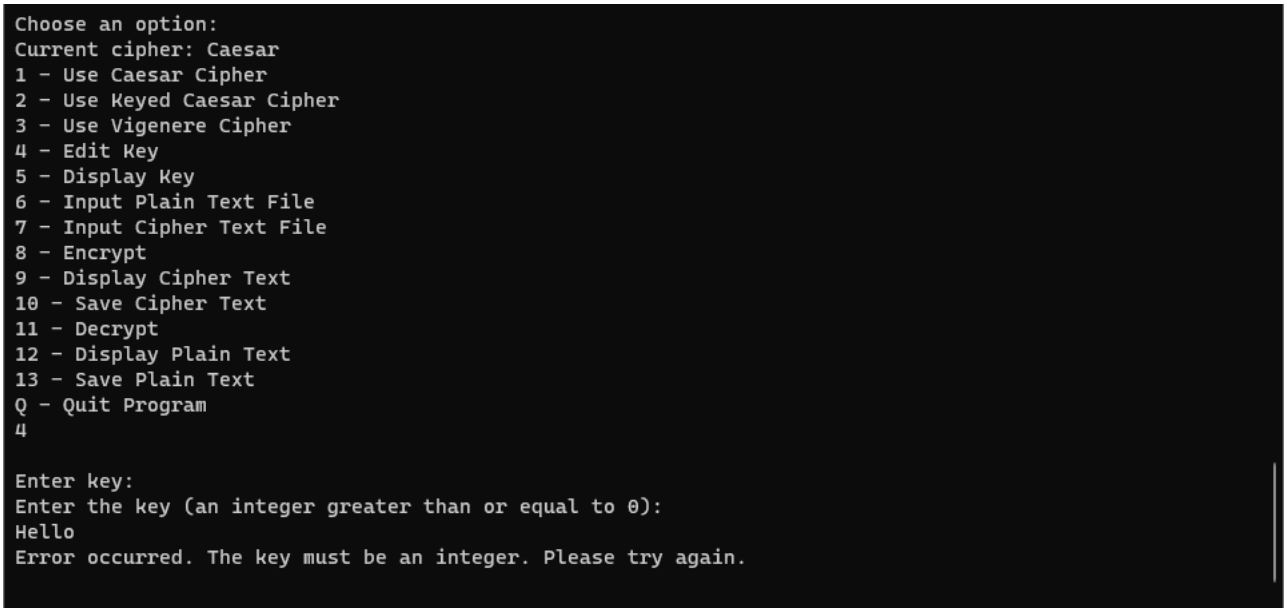

Figure 5.1.2: Entering an invalid key. Error output.

Figure 5.1.3: Displaying the key.

```
Choose an option:
Current cipher: Caesar
1 - Use Caesar Cipher
2 - Use Keyed Caesar Cipher
3 - Use Vigenere Cipher
4 - Edit Key
5 - Display Key
6 - Input Plain Text File
7 - Input Cipher Text File
8 - Encrypt
9 - Display Cipher Text
10 - Save Cipher Text
11 - Decrypt
12 - Display Plain Text
13 - Save Plain Text
Q - Quit Program
6

Enter file name for plain text:
plaintext-input.txt
```

Figure 5.2.1: Inputting a plain text file.

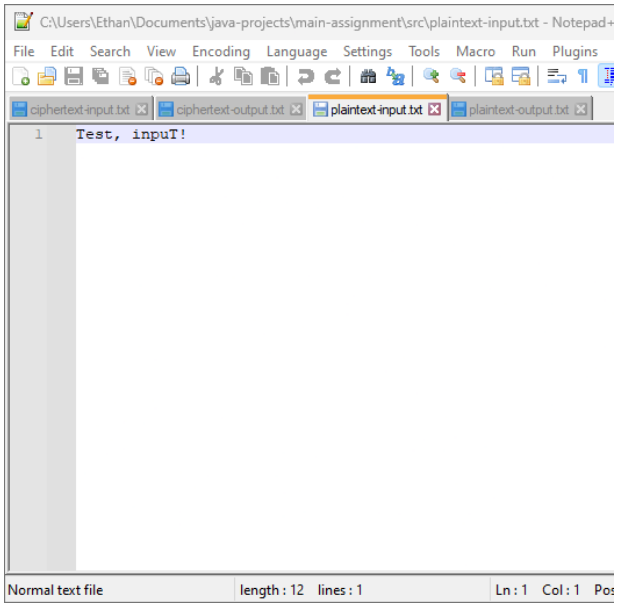

Figure 5.2.2: Contents of the plain text file.

```
Choose an option:
Current cipher: Caesar
1 - Use Caesar Cipher
2 - Use Keyed Caesar Cipher
3 - Use Vigenere Cipher
4 - Edit Key
5 - Display Key
6 - Input Plain Text File
7 - Input Cipher Text File
8 - Encrypt
9 - Display Cipher Text
10 - Save Cipher Text
11 - Decrypt
12 - Display Plain Text
13 - Save Plain Text
Q - Quit Program
6

Enter file name for plain text:
unknown-file.txt

Error occurred. Cannot read plain text file.
```

Figure 5.2.3: Inputting a plain text file that doesn't exist. Error output.

Figure 5.3.1: Encrypting the prepared plain text and displaying the cipher text output.

Figure 5.3.2: Decrypting the cipher text result back to prepared plain text.

```
Choose an option:
Current cipher: Caesar
1 - Use Caesar Cipher
2 - Use Keyed Caesar Cipher
3 - Use Vigenere Cipher
4 - Edit Key
5 - Display Key
6 - Input Plain Text File
7 - Input Cipher Text File
8 - Encrypt
9 - Display Cipher Text
10 - Save Cipher Text
11 - Decrypt
12 - Display Plain Text
13 - Save Plain Text
Q - Quit Program
7

Enter file name for cipher text:
ciphertext-input.txt
```

Figure 5.4.1: Inputting a cipher text file.

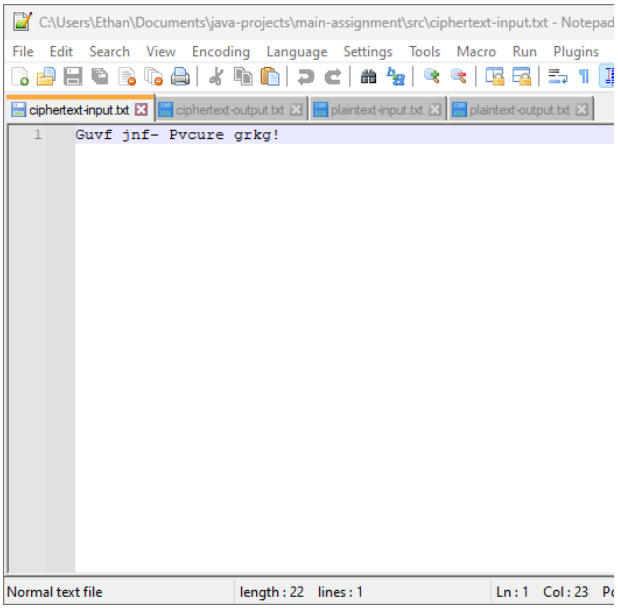

Figure 5.4.2: Contents of the cipher text file.

```
11
Choose an option:
Current cipher: Caesar
1 - Use Caesar Cipher
2 - Use Keyed Caesar Cipher
3 - Use Vigenere Cipher
4 - Edit Key
5 - Display Key
6 - Input Plain Text File
7 - Input Cipher Text File
8 - Encrypt
9 - Display Cipher Text
10 - Save Cipher Text
11 - Decrypt
12 - Display Plain Text
13 - Save Plain Text
Q - Quit Program
12

Prepared plain text content: THIS WAS CIPHER TEXT
```

Figure 5.4.3: Decrypting the contents of the cipher text file and displaying the plain text output.

```
Choose an option:
Current cipher: Caesar
1 - Use Caesar Cipher
2 - Use Keyed Caesar Cipher
3 - Use Vigenere Cipher
4 - Edit Key
5 - Display Key
6 - Input Plain Text File
7 - Input Cipher Text File
8 - Encrypt
9 - Display Cipher Text
10 - Save Cipher Text
11 - Decrypt
12 - Display Plain Text
13 - Save Plain Text
Q - Quit Program
13

Enter file to save the plain text to:
plaintext-output.txt
```

Figure 5.5.1: Entering a file name to save the prepared plain text to.

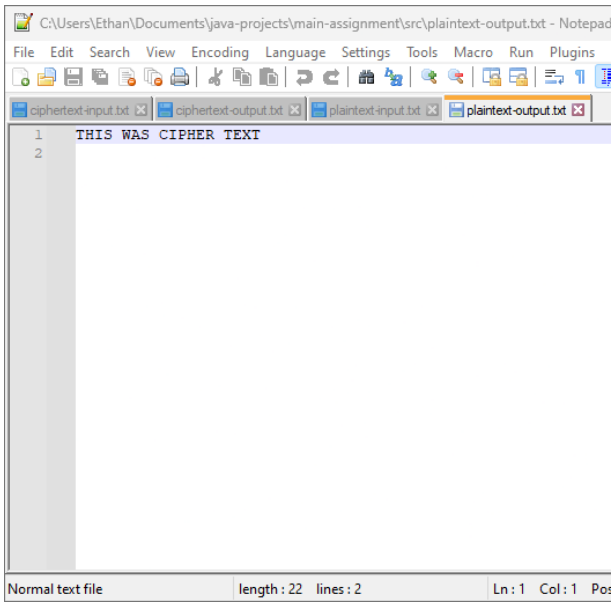

Figure 5.5.2: The contents of the prepared plain text output file.

```
Choose an option:
Current cipher: Caesar
1 - Use Caesar Cipher
2 - Use Keyed Caesar Cipher
3 - Use Vigenere Cipher
4 - Edit Key
5 - Display Key
6 - Input Plain Text File
7 - Input Cipher Text File
8 - Encrypt
9 - Display Cipher Text
10 - Save Cipher Text
11 - Decrypt
12 - Display Plain Text
13 - Save Plain Text
Q - Quit Program
10

Enter file to save cipher text to:
ciphertext-output.txt
```

Figure 5.5.3: Entering a file name to save the cipher text to.

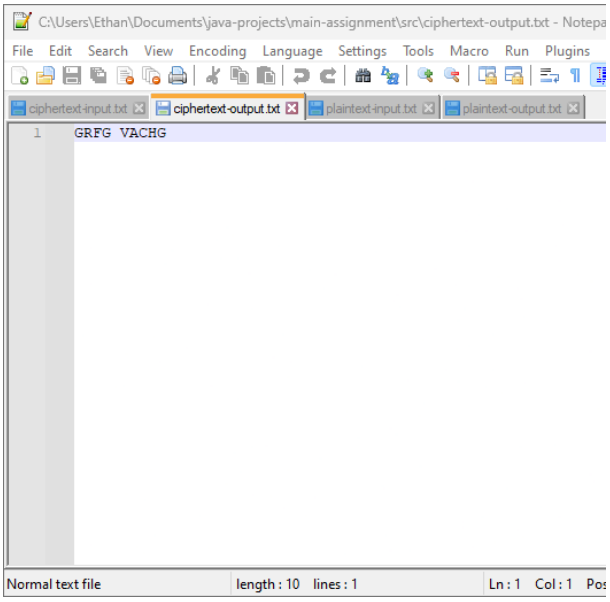

Figure 5.5.4: The contents of the cipher text output file.

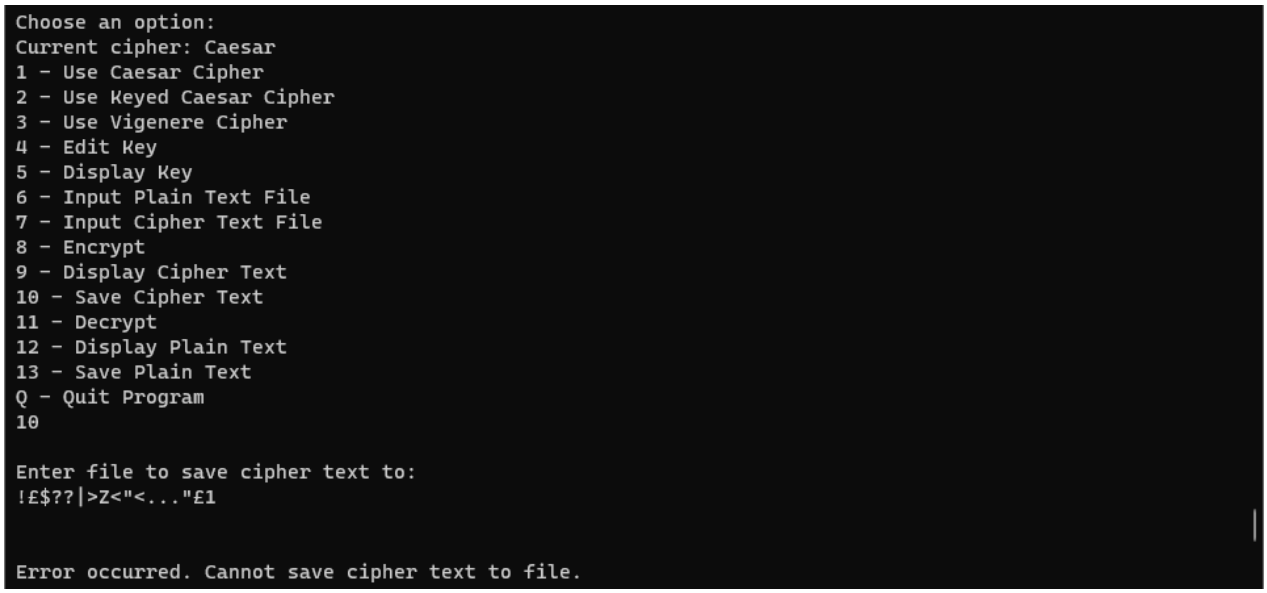

Figure 5.5.5: Entering an invalid file name. Error output.

Figure 6.1: Changing the chosen cipher to Keyed Caesar Cipher.

```
4

Enter key:
49INPUTTEST

Choose an option:
Current cipher: Keyed Caesar
1 - Use Caesar Cipher
2 - Use Keyed Caesar Cipher
3 - Use Vigenere Cipher
4 - Edit Key
5 - Display Key
6 - Input Plain Text File
7 - Input Cipher Text File
8 - Encrypt
9 - Display Cipher Text
10 - Save Cipher Text
11 - Decrypt
12 - Display Plain Text
13 - Save Plain Text
Q - Quit Program
5

Current key: 49INPUTTEST
```

Figure 6.2.1: Setting the key to be used and displaying it.

```
Choose an option:
Current cipher: Keyed Caesar
1 - Use Caesar Cipher
2 - Use Keyed Caesar Cipher
3 - Use Vigenere Cipher
4 - Edit Key
5 - Display Key
6 - Input Plain Text File
7 - Input Cipher Text File
8 - Encrypt
9 - Display Cipher Text
10 - Save Cipher Text
11 - Decrypt
12 - Display Plain Text
13 - Save Plain Text
Q - Quit Program
4

Enter key:
WRONG31FORMAT
Error occurred. The key must be in the format of an integer followed by a string (e.g. 1TEST). Please
try again.
```

Figure 6.2.2: Entering an invalid key. Error output.

```
8
Choose an option:
Current cipher: Keyed Caesar
1 - Use Caesar Cipher
2 - Use Keyed Caesar Cipher
3 - Use Vigenere Cipher
4 - Edit Key
5 - Display Key
6 - Input Plain Text File
7 - Input Cipher Text File
8 - Encrypt
9 - Display Cipher Text
10 - Save Cipher Text
11 - Decrypt
12 - Display Plain Text
13 - Save Plain Text
Q - Quit Program
9

Cipher text content: LNKL EDGML
```

Figure 6.3.1: Encrypting the prepared plain text (shown in Figure 5.2.2) and displaying the cipher text output.

Figure 6.3.2: Decrypting the cipher text result and displaying the plain text output.

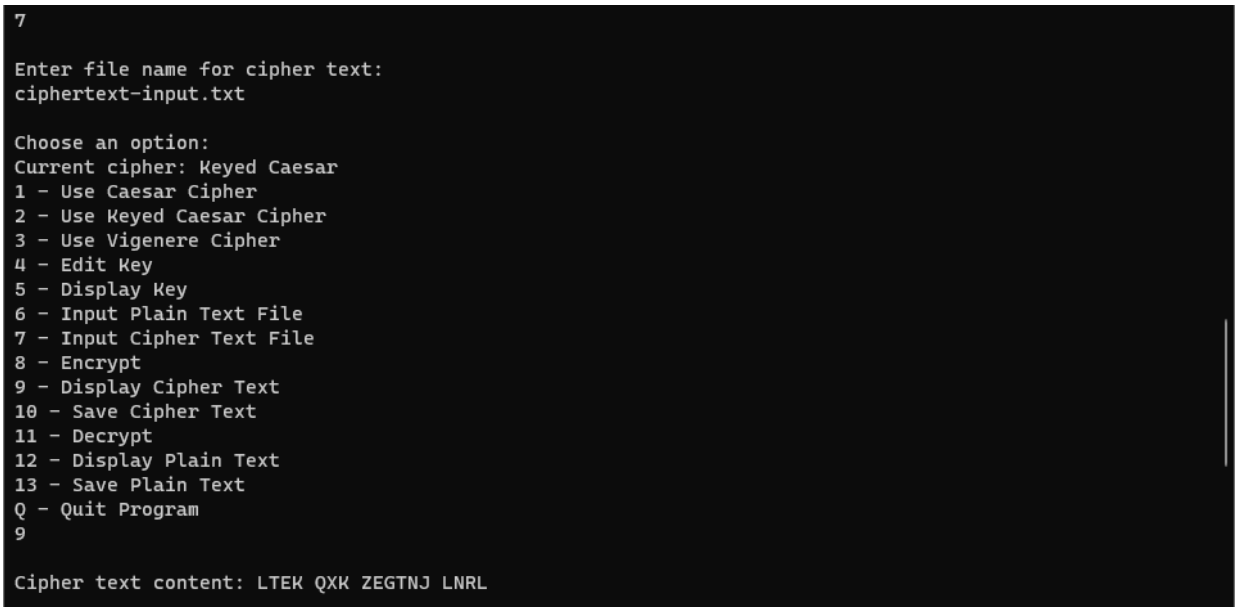

Figure 6.4.1: Inputting a cipher text file and displaying the contents.

Figure 6.4.2: Decrypting the contents of the cipher text file and displaying the plain text output.

Figure 7.1: Changing the chosen cipher to Vigenere Cipher.

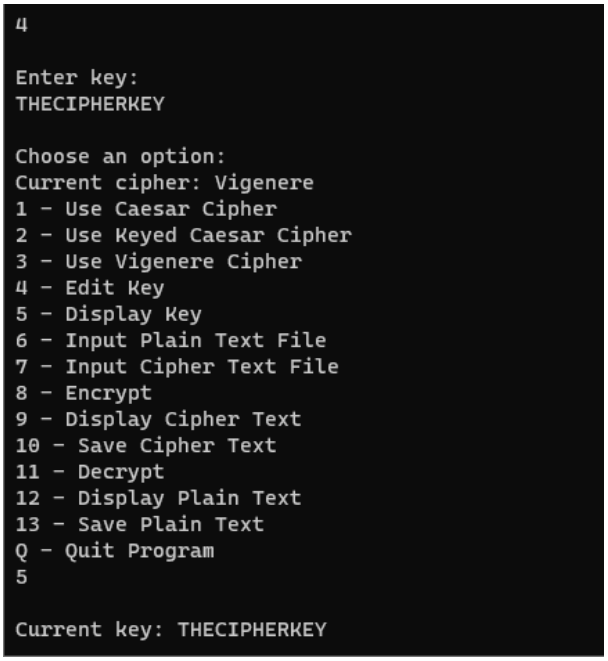

Figure 7.2.1: Setting the key to be used and displaying it.

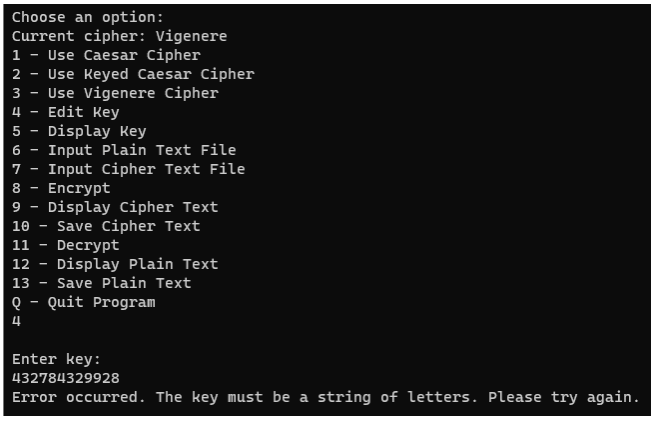

Figure 7.2.2: Entering an invalid key. Error output.

Figure 7.3.1: Encrypting the prepared plain text (shown in Figure 5.2.2) and displaying the cipher text output.

Figure 7.3.2: Decrypting the cipher text result and displaying the plain text output.

Figure 7.4.1: Inputting a cipher text file and displaying the contents.

Figure 7.4.2: Decrypting the contents of the cipher text file and displaying the plain text output.

Figure 8.1: Entering an invalid option in the menu.

```
Choose an option:
Current cipher: Vigenere
1 - Use Caesar Cipher
2 - Use Keyed Caesar Cipher
3 - Use Vigenere Cipher
4 - Edit Key
5 - Display Key
6 - Input Plain Text File
7 - Input Cipher Text File
8 - Encrypt
9 - Display Cipher Text
10 - Save Cipher Text
11 - Decrypt
12 - Display Plain Text
13 - Save Plain Text
Q - Quit Program
q

C:\Users\Ethan\Documents\java-projects\main-assignment\src>
```

Figure 9.1.1: Quitting the program.

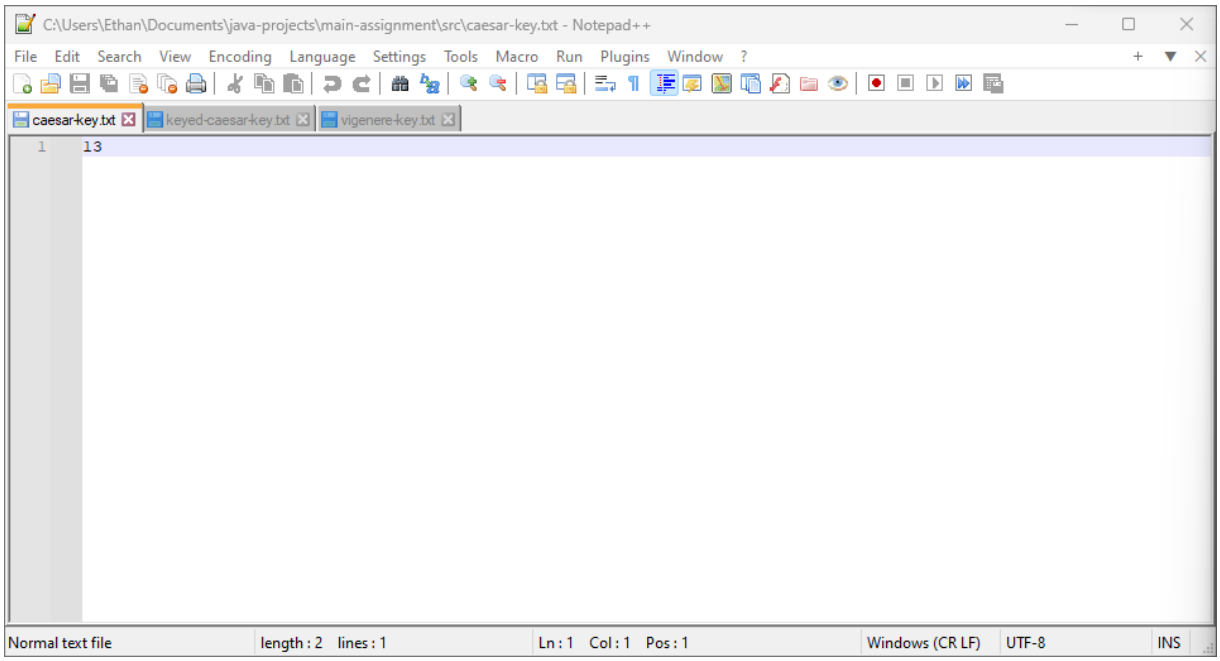

Figure 9.2.1: Caesar Cipher key saved to "caesar-key.txt" after program exits.

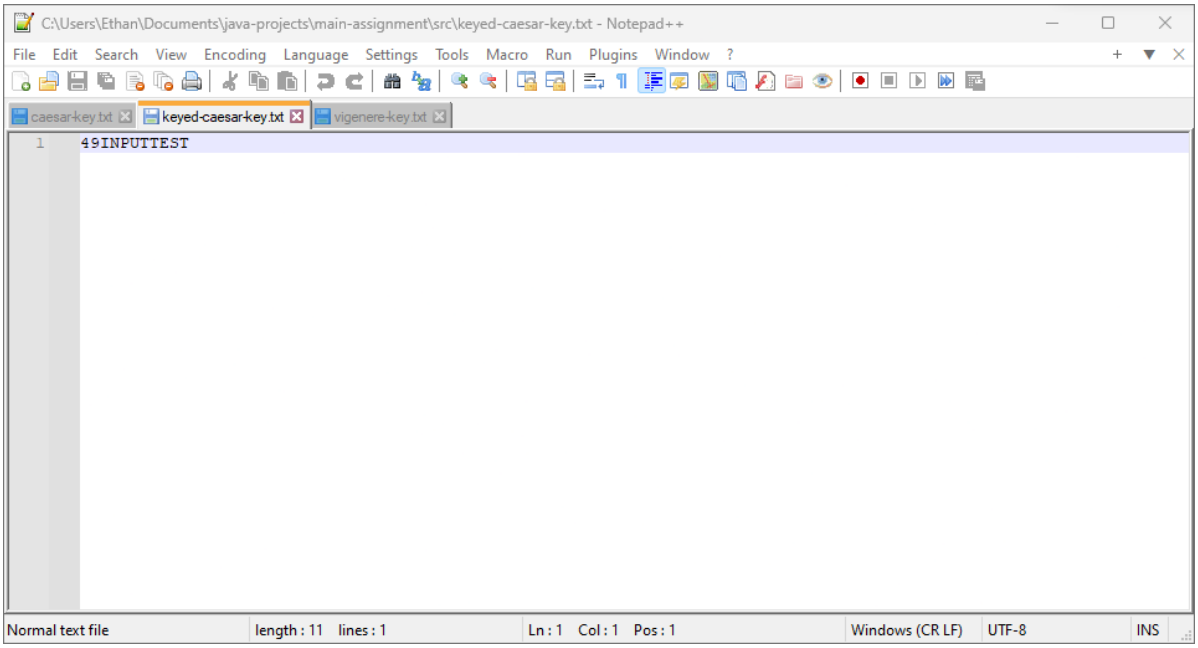

Figure 9.2.2: Keyed Caesar Cipher key saved to "keyed-caesar-key.txt" after program exits.

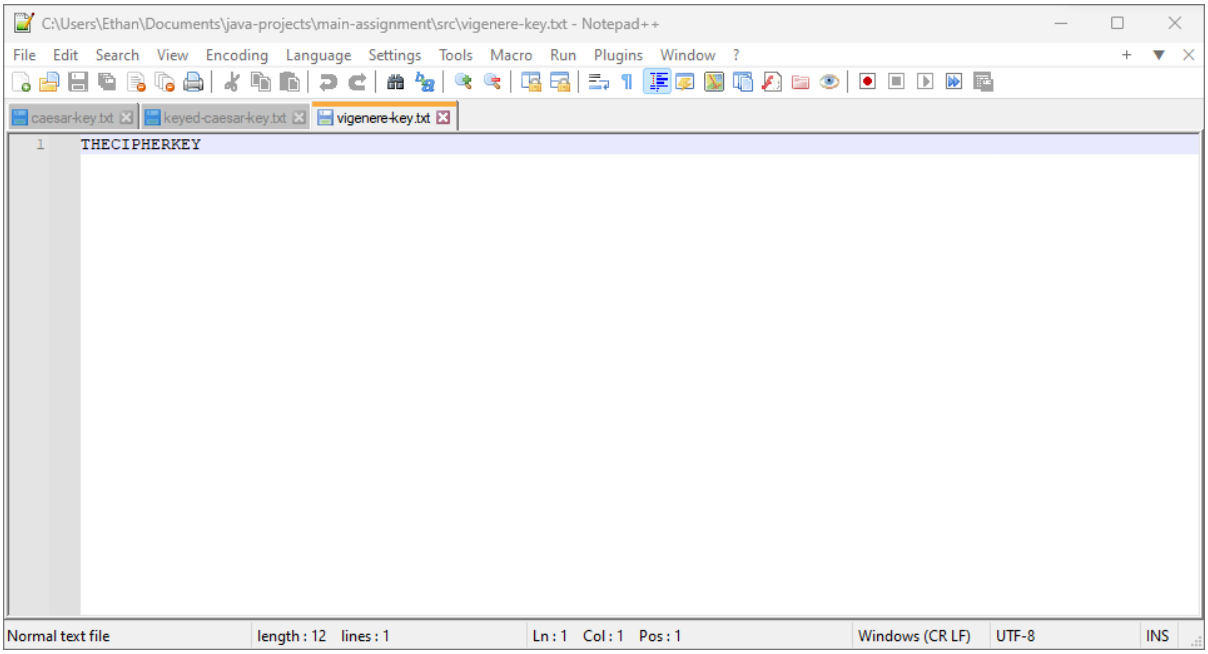

Figure 9.2.3: Vigenere Cipher key saved to "vigenere-key.txt" after program exits.

3.3 Discussion

To ensure my program functioned in accordance with the guidelines set in the assignment brief, I executed a range of different inputs into my code, to guarantee that no erroneous or invalid data was produced.

For every individual functional requirement, I input a range of valid, as well as invalid, data into the program. One way I easily checked the current stored inputs was to output the contents stored in the Cipher class. This displayed the prepared plain text, cipher text, and the key used for a range of different operations in the program.

Given that each cipher algorithm requires a specific key format, I ensured that the code was able to detect any incorrect key format issues. This was pivotal, as if this error checking didn't exist, then erroneous data would be output.

Furthermore, since loading and saving from files is a large aspect of this assignment, I tested an extensive range of different inputs for entering filenames. This included invalid file names, and pre-existing file names. Any errors generated were displayed in the command-line to the user. I also attempted to enter in files that contained invalid data. For example, I entered the "caesar-key.txt" file with a string instead of an integer inside, which allowed the file to load but produced an error message when the encryption or decryption algorithm was called. This prompted me to implement more specific error-checking statements.

One issue that I came across whilst testing file inputs was when I entered a plain text file containing abnormal characters. Depending on the cipher, when I called the encryption or decryption operations, some characters would bypass the error checking statements I had initially put in place. This caused the alphabet used in the algorithm to be inaccurate, thus producing the incorrect cipher text output. I resolved this issue by refining the error-checking statements to allow only a very specific range of characters, instead of disallowing set punctuation characters. To ensure the correct output was produced from the encryption and decryption algorithms, I entered the example test data as shown in the supplementary section of the assignment brief [1]. As well as this, I read through other details and examples of each cipher, explained in CS11110: Information Security [2]. My test table details the full list of test inputs I trialled in my program and how I resolved all the issues I encountered during the development of the program.

4 Evaluation

4.1 Approach to Solving the Assignment

Prior to implementing the features of the assignment brief into Java, I carefully read through the assignment brief, with the objective of creating an efficient and structured approach. This included the creation of a comprehensive plan, including pseudo-code of the required features. During the development of my code, I frequently tested the program to ensure that it was working optimally, making use of breakpoints to help diagnose any issues I encountered.

4.1.1 Difficulties Encountered

One of the primary difficulties I faced during development was incorporating the use of inheritance, which proved to be a major obstacle as initially, I had stored each key for each cipher as a different data type. This presented a significant challenge, as it was difficult to store similar properties in the parent class. The first version of the program met all the basic functional requirements, however, there was limited error checking and a lack of correctly used inheritance. To improve the code, I refined my initial plans and rewrote each cipher algorithm to inherit three private instance variables that could be stored in a parent class.

Another difficulty I encountered was during the implementation of the Keyed Caesar Cipher algorithm. This was mainly due to misunderstanding the details of the cipher function, as it utilised three different alphabets. Additionally, implementing this algorithm proved to be challenging due to the key being composed of both an integer and a string. As I was inheriting the key, I had to split it into an integer and a string in the algorithm.

4.1.2 Remaining Work

Although I completed all functional requirements detailed in the assignment brief, there is the potential for showing further creativity and innovation. To increase usability of my program, I could implement a graphical user interface, which would be more user-friendly than a command-line interface for many users. Furthermore, I could also add additional cryptographic algorithms, such as AES, MD5, DES, etc.

4.1.3 What has been Learnt

Completing this assignment has further enhanced my skills in Java programming, as well as increasing my understanding of the principles of inheritance. I have also gained a realisation of the importance of code optimisation, particularly with reducing code duplication. Through careful planning of my development process, I was able to create my code and encounter minimal errors. I believe that I have improved since my Mini Assignment, as I have built on the principles I learnt from that.

4.2 Self Evaluation

4.2.1 What Mark I Should be Awarded

I have invested a considerable amount of time and effort into this assignment, ensuring that all the functional requirements outlined in the assignment brief were completed with code of high quality, featuring high cohesion and low coupling. While the code meets all the functional requirements specified, I am certain that there is potential for further improvement, which could show further creativity and innovation. In terms of the mark I think I should be awarded, I feel that I deserve a mark of 75% due to the high quality of my program and the level of effort put in.

References

[1] *Main Assignment: The Code Breaker (Online)*
https://blackboard.aber.ac.uk/bbcswebdav/pid-2465274-dt-content-rid-8596383_1/xid-8596383_1
Accessed: 20th March 2023
Note: Restricted Access (Aberystwyth University Blackboard)

[2] *Codes and Ciphers: An Introduction (Online)*
Hannah Dee
https://blackboard.aber.ac.uk/bbcswebdav/pid-2144989-dt-content-rid-7286937_1/xid-7286937_1
Accessed: 6th April 2023
Note: Restricted Access (Aberystwyth University Blackboard)