Programming using an Object-Oriented Language

Main individual Assignment: The Code Breaker

Author: Elliott Bown

Email: elb95@aber.ac.uk

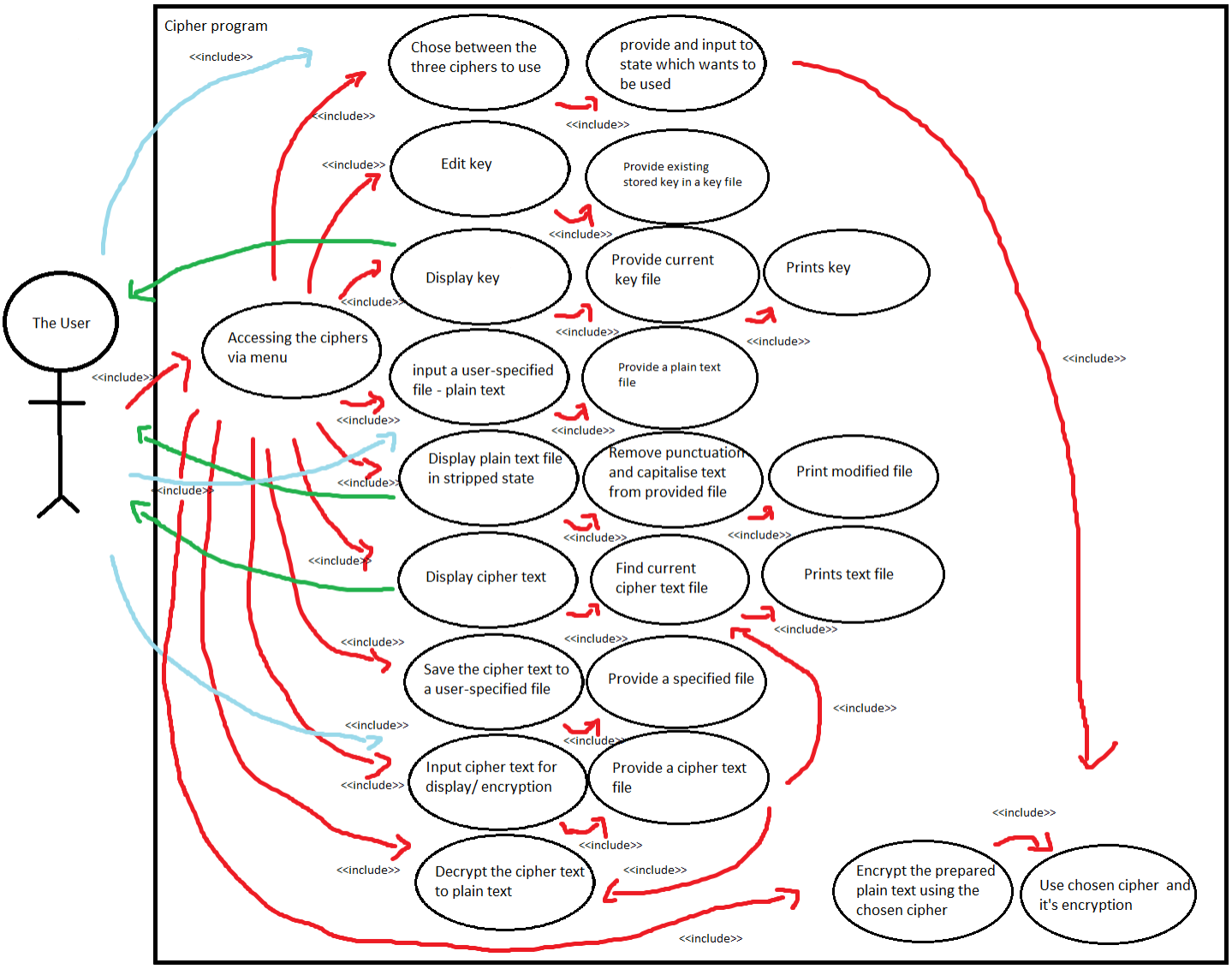
Date: 24/05/2023

Table of Contents

Introduction

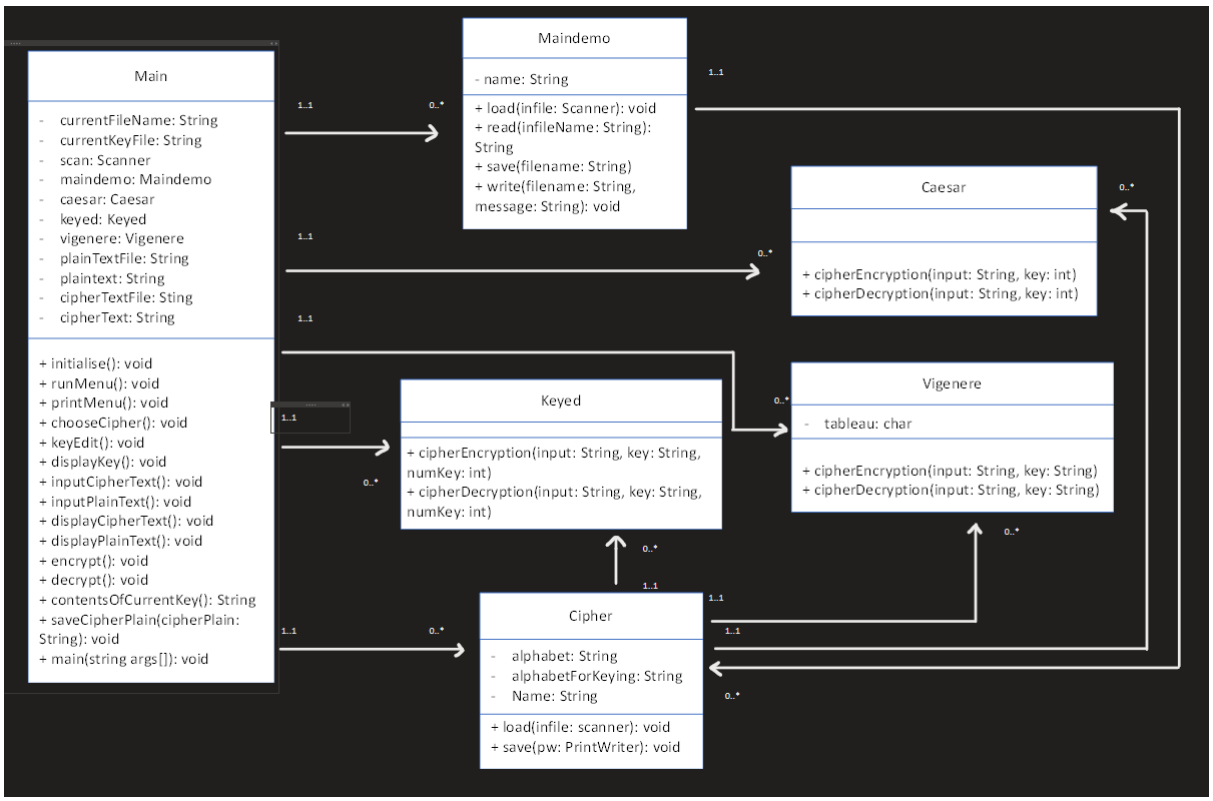
In this assignment, I was tasked with designing and coding a program that took files with messages in, and encrypted them using wither the Caesar cipher, Keyed Caesar cipher or the Vigenère cipher. This report follows the design process alongside some diagrams of the code designs, the testing of the algorithms, to make sure they followed what their intended use was, some screenshots depicting the working program and finally an evaluation of how the process went.
To summarise, I was able to achieve a working encryption program, following the principles stated in the brief within the time limit of the assignment without any major problems with development.

UML Use-case diagram



Design

UML Class Diagram



The smallest classes are “Caesar”, “Keyed” and “Vigenere” which only contain the code needed to encrypt and decrypt a string input with their cipher. These three classes are child classes to the parent “Cipher” class which contains some shared variables between the child classes like the variable alphabet.
The class “Maindemo” acts as a background running class to the “Main”, using the functions in cipher to include the saving, loading, and reading to files.
The “Main” Class runs all the functions made in “Maindemo” and runs the menu and interface to the program, adding extra programs to compile and run the extra functions that do not need to be included in any specific object-oriented way.

Pseudo-Code for the fuction “keyEdit()” in the class “Main()”

Function start

Call displayKey() to load file

Print if user would like to edit cipher key

Scan for input for a Y or N

Start Switch

    Case caesar

        Case Y

            Scan for an input for a new key

            Write the new key to current file

            Print the new key

        Case N

            Break

    Case keyed caesar

        Case Y

            Scan for an int number shift

            While the input > 26

                Scan for an int number shift

            Scan for a key word

            Change key word to upper case

            Remove spaces from key word

            Add the number shift and key word into one string

            Write the concatenated string into the current file

            Print the new key

        Case N

            Break

Case vigenere

Case Y

Scan for an input for a new key

While new key has numbers in it

Scan an input for a new key

Change key word to upper case

Remove spaces from key word

Write new key word to current file

Print new key word

Case N

Break

I think this function was the most complicated because of the amount of switch cases to accommodate the different instances in which this could occur, when the different ciphers are currently selected.
As well as this, the function handles loading the file to write to and displaying the current key and then writing to the file after changing the key.
This is also the function with the most “while loops” to catch errors and discrepancies when the user types incorrect inputs, such as a string when the input requires an integer.

Testing

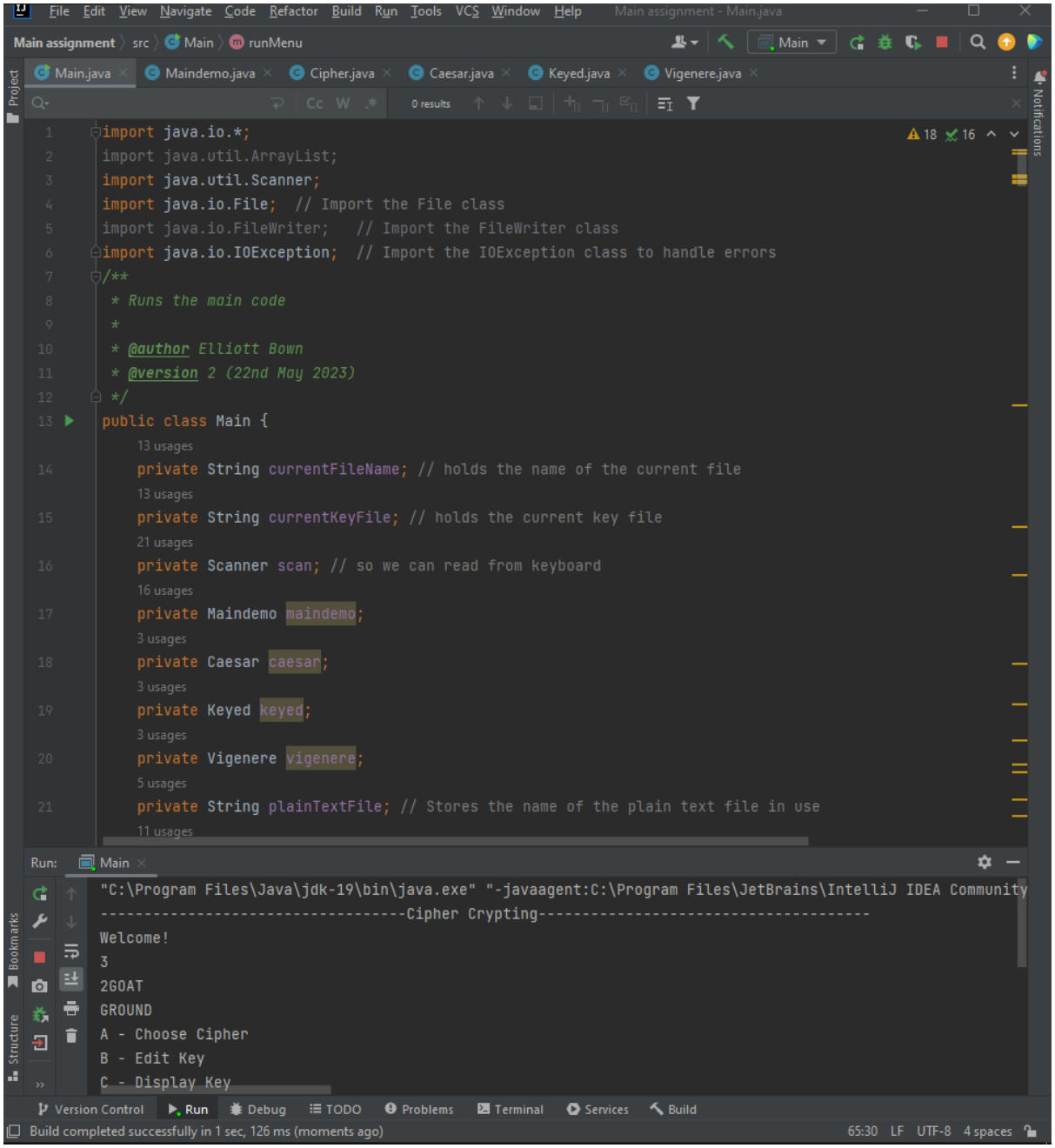
ID	Requirement	Description	Inputs	Expected Outputs	Pass/fail	Comments
A1	FR6	Encrypt plain text with chosen cipher	The plain text encrypted with Caesar cipher	The plain text correctly encrypted with a correctly shifted alphabet	P	
			The plain text encrypted with keyed cipher	The plain text correctly encrypted with a correctly keyed alphabet	P	
			The plain text encrypted with no cipher	N/A – always have a cipher chosen	P	
A2	FR1	Choose between the three ciphers	Enter “a” – the option for Caesar cipher	The code to make that capital and chose Caesar cipher as the current cipher	P	
			Enter “C” – the option for the Vigenère cipher	The code to match that to the option for Vigenère cipher and make that the current cipher	P	
			Enter “;” – not an option or even a letter	“Wrong input, try again” to come up on the screen	P	
A3	FR3/FR7	Display the Key/cipher	Enter “c” – the input for accessing the requirement	Capitalises the c and correctly load up the right function, displaying the current key for the current cipher/ cipher text	P	
A4	FR2	Edit the key – starts on a yes/ no question that the user must input y or n to continue,	Enter “;” – at the yes/ no question stage	“Wrong input, try again” to come up on the screen and asks you to try again	F	Does the display bit, but takes you back to the main menu
			Enter a non-integer key	“Wrong input, try again” to come up	F	Number format

		then allows you to enter a key	for the Caesar cipher	on the screen and asks you to try again		exception was thrown
			Enter a correct value key	“New key for Caesar cipher is: ‘input’”	P	
A5	FR4	Input a user-specified, plain text file	Enter a valid text file	Shows current contents of file and what it looks like formatted for encrypting	P	
			Enter a non-valid text file	“That's not a valid file.” Appears on the screen	F	Show’s that text, but then also shows “The formatted version of the file is:”
A6	FR6	Encrypt the prepared plain text using the chosen cipher	Entering “f” – the input to access the function – while a plaintext is selected	Tells you the encrypted version of the message	P	
			Entering “f” – the input to access the function – while there is no plaintext is selected	It takes you back to the original menu	P	
A7	FR8	Save the cipher text to a user specified file	Entering “h” – the input to access the function – while there is no encrypted messages	“Wrong input, try again” is displayed and takes you back to the menu	F	Throws a Null pointer exception
			Specifying a non-file when you are asked where to save cipher text to	“Wrong input, try again” is displayed and takes you back to the menu	F	Throws a Null pointer exception
			Specifying a non-file	Throws a Null pointer exception	P	

			while there is no encrypted messages			
A8	FR9	Input cipher text file so it can be displayed or decrypted	Specifying a non-file	"That's not a valid file" appears	P	
A9	FR10	Decrypt the cipher text to prepared plain text	Entering "i" – the input to access the function – while a cipher text is selected	Tells you the Decrypted version of the message	P	
			Entering "I" – the input to access the function – while there is no cipher text is selected	It takes you back to the original menu	P	
A10	FR11	Exit the program	Enter "Q" t exit the program	The program finishes	P	
			Enter anything else to exit the program	Either "Wrong input, try again" and takes you back to the menu, or you access another function	P	

Running the tests was simple and productive considering the majority passed their respective ones.
From observation, the functions which did not pass seem to throw exceptions to notify that it is a wrong input besides, although not continuing the program afterwards would be a detriment to the design if this were for any useful task.
Some of the functions could not be tested as there were no inputs to test on, such as displaying the cipher text, which only takes the already given cipher file and displays the text within.

The working program (with screenshots and descriptions)



```
----------------------------------Cipher Crypting-------------------------------------
Welcome!
5
3LEMON
YEET
A - Choose Cipher
B - Edit Key
C - Display Key
D - Input plain text file
E - Input cipher text file
F - Encrypt
G - Display cipher text
H - Save cipher text
I - Decrypt
J - Display plain text
K - Save plain text
Q - Quit
Current cipher: Caesar
Choose an option
```

You enter the program and it takes you to a menu with each of the other options on. Enter a letter corresponding to the function you want to continue.

```
Current cipher: Caesar
Choose an option
a
Current cipher:
Which cipher would you like to use?
A - Caesar
B - Keyed Caesar
C - Viginere
Q - quit
```

Typing “a” or “A”, the code capitalises the text if it isn’t already, will allow you to choose a cipher to encrypt test with.

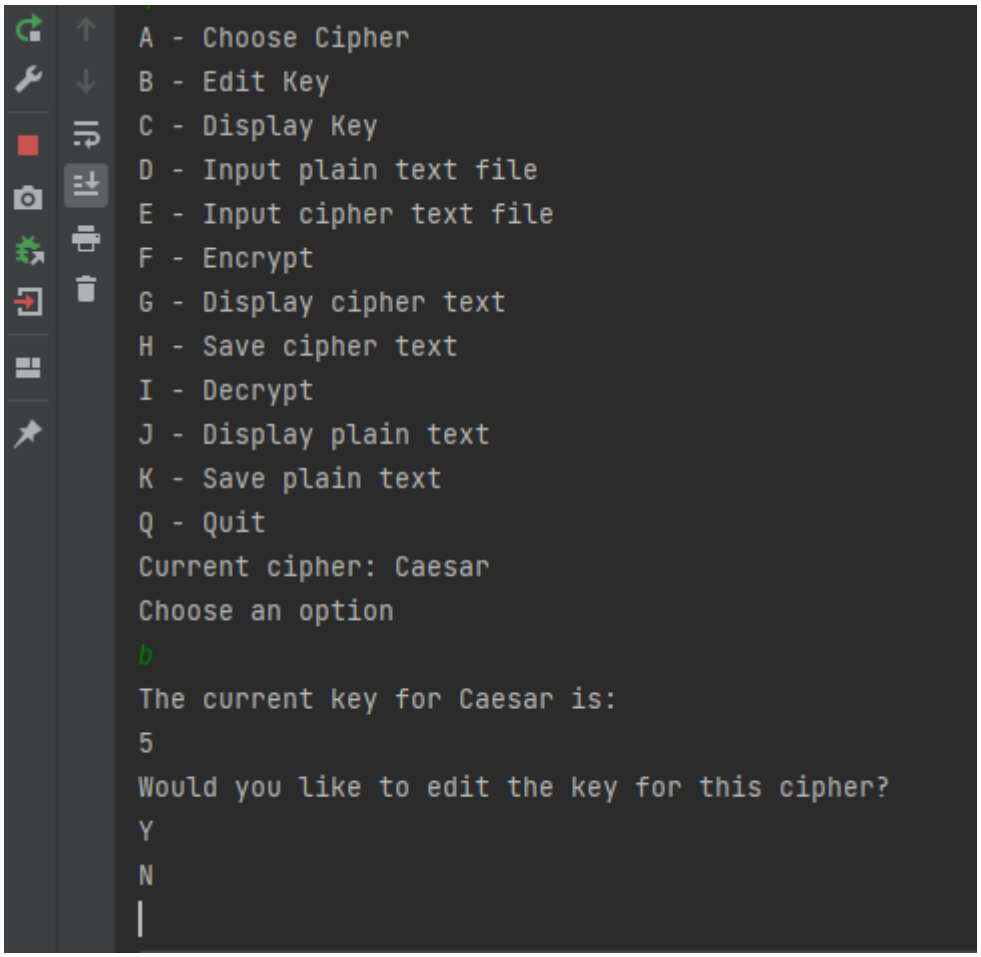
```
a
The current cipher is now: Caesar
Type Q to quit
```

```
b
The current cipher is now: Keyed Caesar
Type Q to quit
```

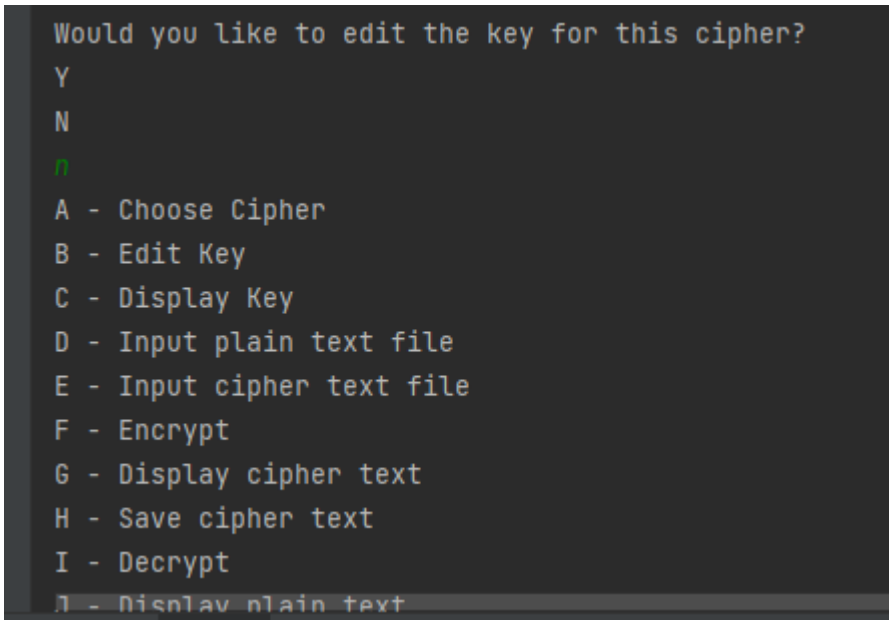
```
c
The current cipher is now: Vigenere
Type Q to quit
```

```
g
Wrong input, try again
```

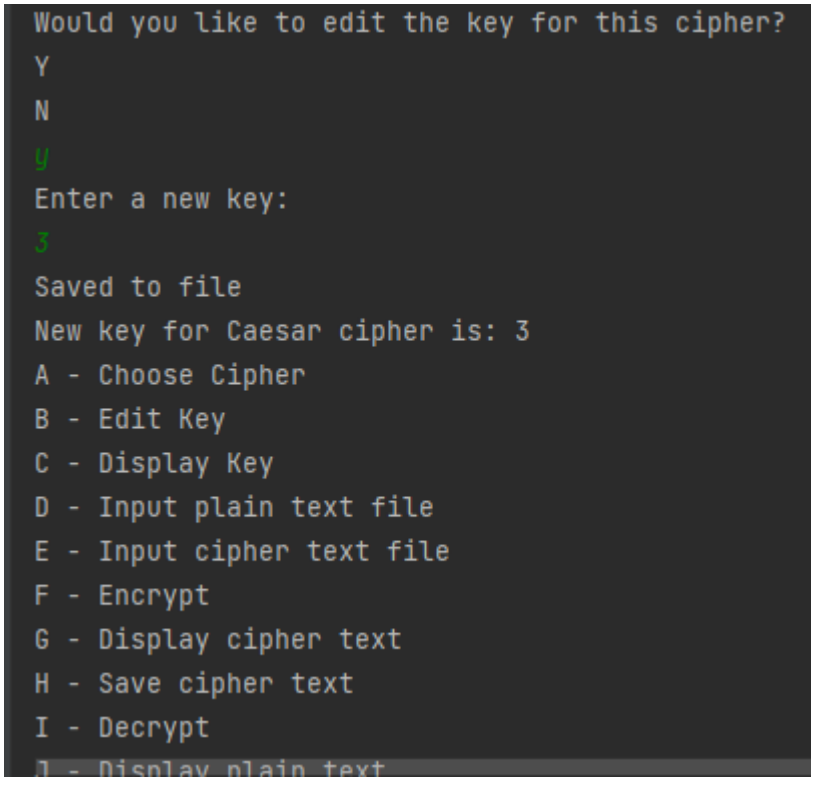
Typing in an unassigned input will come u with this message and let you input another letter.



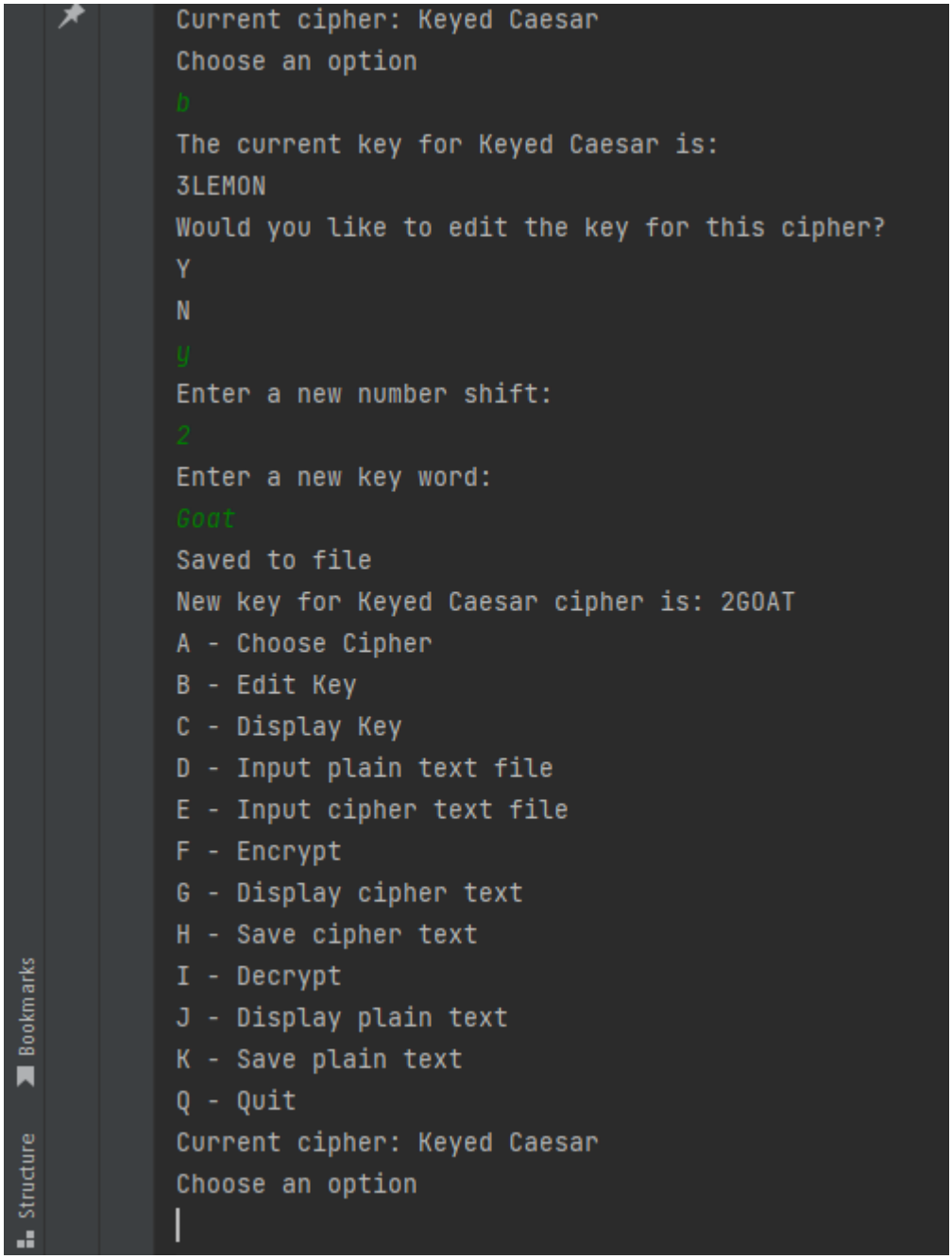
After choosing a cipher, it takes you back to this menu, by inputting “b”, you come to the edit key function which allows you to edit a key for the different ciphers.



“N” takes the user back to the menu.



“Y” allows the user to input a new key, which is then saved and returns the user to the main menu to continue. This is applicable to all ciphers.


Current cipher: Keyed Caesar
Choose an option
b
The current key for Keyed Caesar is:
3LEMON
Would you like to edit the key for this cipher?
Y
N
y
Enter a new number shift:
2
Enter a new key word:
Goat
Saved to file
New key for Keyed Caesar cipher is: 2GOAT
A - Choose Cipher
B - Edit Key
C - Display Key
D - Input plain text file
E - Input cipher text file
F - Encrypt
G - Display cipher text
H - Save cipher text
I - Decrypt
J - Display plain text
K - Save plain text
Q - Quit
Current cipher: Keyed Caesar
Choose an option
Bookmarks
Structure

```
Current cipher: Vigenere
Choose an option
b
The current key for Vigenere is:
YEET
Would you like to edit the key for this cipher?
Y
N
y
Enter a new key:
Ground
Saved to file
New key for Keyed Caesar cipher is: GROUND
A - Choose Cipher
B - Edit Key
C - Display Key
D - Input plain text file
E - Input cipher text file
F - Encrypt
G - Display cipher text
H - Save cipher text
I - Decrypt
J - Display plain text
K - Save plain text
Q - Quit
Current cipher: Vigenere
Choose an option
```

```
Q - Quit
Current cipher: Vigenere
Choose an option
c
The current key for Vigenere is:
GROUND
A - Choose Cipher
```

Inputting “C” on the main menu displays the key for the current cipher.

```
Q - Quit
Current cipher: Caesar
Choose an option
c
The current key for Caesar is:
3
A - Choose Cipher
```

```
Q - Quit
Current cipher: Keyed Caesar
Choose an option
c
The current key for Keyed Caesar is:
2GOAT
A - Choose Cipher
```

```
Choose an option
d
Input the file name:
plain.txt
Current contents of file:
yo sup howdy
The formatted version of the file is:
YOSUPHOWDY
Saved to file
And is stored in prep.txt
A - Choose Cipher
```

“D” allows the user to specify a text file and shows the contents as well as how the contents will be formatted for coding. Inputting a non-valid file will result in a message notifying the user as much.

```
Input the file name:
d
That's not a valid file.
```

```
Q - Quit
Current cipher: Caesar
Choose an option
e
Input the cipher text file name:
cipher.txt
Current contents of the cipher text file:
HXJJXSF
The formatted version of the file is:
HXJJXSF
Saved to file
And is stored in prep.txt
A - Choose Cipher
```

“E” will do the same thing but with a cipher text file.

```
Q - Quit
Current cipher: Caesar
Choose an option
f
The Encryption of this message becomes:
BRVXSKRZGB
A - Choose Cipher
```

The input of “F” will encrypt the current plain text file with the current cipher and display the result. Here the encryption used is the Caesar cipher.

```
Q - Quit
Current cipher: Keyed Caesar
Choose an option
f
The Encryption of this message becomes:
GPUWQHPYCG
```

Encryption with Keyed Cipher.

```
Current cipher: Vigenere
Choose an option
f
The Encryption of this message becomes:
GHIQEMWPTU
```

Encryption with Vigenère cipher.

```
Q - Quit
Current cipher: Caesar
Choose an option
g
The Formatted contents of the file prep.txt is:
HXJJXSF
A - Choose Cipher
```

After inputting a cipher text file using the E function, “G” will display the text within the cipher text file.

```
Choose an option
h
Specify the file you want to save thisciphertext to:
Hello.txt
File created with the name: Hello.txt
A - Choose Cipher
```

```
Choose an option
h
Specify the file you want to save thisciphertext to:
cipher2.txt
File already exists
```

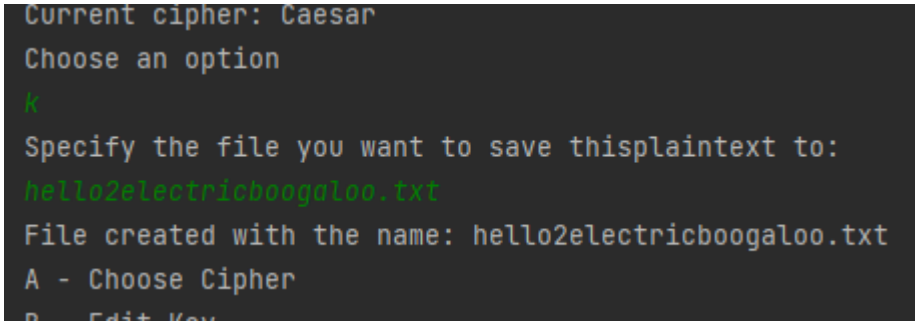
“H” will save cipher text created with the encrypt function to a user-created file, if that file already exists, there will be a message to notify the user and take the user back to the menu.

```
Q - Quit
Current cipher: Caesar
Choose an option
i
The Decryption of this message becomes:
YOSUPHOWDY
A - Choose Cipher
```

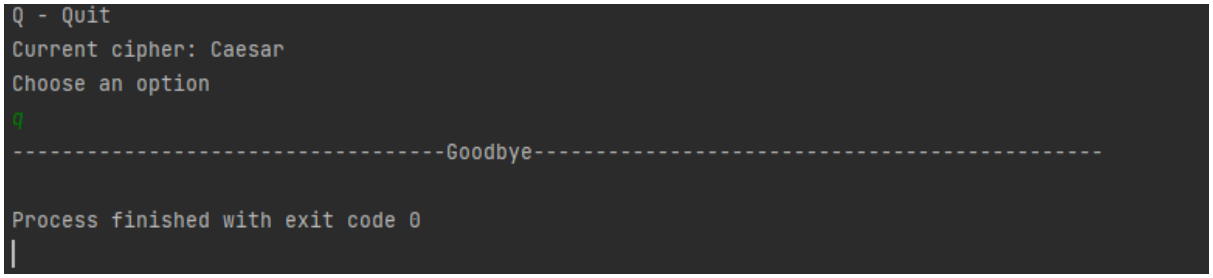
The input “I” will decrypt any encrypted code and display it.

```
Q - Quit
Current cipher: Caesar
Choose an option
j
The Formatted contents of the file prep.txt is:
YOSUPHOWDY
A - Choose Cipher
```

“J” will display the formatted text in the file prep.txt.



“K” will save the decrypted text to a user specified file much like saving the plain text.



And inputting “Q” will quit the menu and therefore the program as well.

Evaluation

I started by researching each of the ciphers using the specified workings, stated in the brief, and began to design and code each individually in their own classes. I started with the Caesar cipher as it was the simplest and continued to the Keyed Caesar Cipher and then the Vigenère cipher last, deciding upon creating the tableau by hand to completely avoid any mistakes in the code later on.
Once I had completed that task, I tested each with manual inputs to make sure they were working and started coding the interactive menu and the following functions to allow the user to interact with the menu.
First I completed the choose between ciphers function to allow the user to choose between the ciphers to encrypt their code with, followed by the "inputPlaintext()" and "inputCipherText()" functions to be able to have text to encrypt, the latter because it was so similar to the prior. Similarly, the "displayCipherText()" and "displayPlainText()" were easy enough to code in conjunction with each other due to the similarities in the shared code and function to display text from a file. It was at this point that I started to concern myself with more inheritance-based classes and made the Cipher class, of which the other three cipher classes extended to reuse variables like "alphabet". As well as this, in each of the functions, I attempted to include code to catch inputs with non-valid types, for example, in the edit key function, I added "while loops" to make sure an integer couldn't be over 26, the number of letter in the alphabet, as well as this, I included try and catches in loading and writing to files to make sure the code did not break if there was an error in loading or writing to a file.

Difficult tasks

Making the menu run in the first place was what I found hardest. After Designing and coding the ciphers, it took me a while to understand the workings of java again, including the inheritance of classes and the object-oriented design, despite having coded in it for the module. It was only after I realised that I could write the functions in the main class and reference them from other classes did I managed to continue my work flow and complete the assignment.
From an actual coding point of view, I would say that figuring out how to save and read from files correctly was the hardest task, as well as adding the try and

catch's so the code would not break if there was ever and error with the files. On the positive side, I am more confident in coding such things now that I have spent time working on them.

Throughout the coding process, I tested my work by running the program multiple times, making sure each element was working before moving onto the next one.

What remains to be done

If I spent more time on this assignment, I would code the program in a way that you could access any one of the functions at any time instead of having to access them to make the code not break, even if the functions do not function, as long as they'd print and error message and continue the code for example.

Awarded marks

I believe I should be awarded 65% of the marks available: My cover page is neat and informative, I am less confident in the detail and how the professionalism of my diagrams would affect the marks so I would dock a few for those, my descriptions of the implementation of the code are fairly decent so I would allow myself most of the points there, the implementation using inheritance went slightly well, although I concede the robustness of the program may not be as well coded as the rest of the program.