CS12320: Programming Using an Object-Orientated Language
Main Assignment

Zeyad S M Moustafa

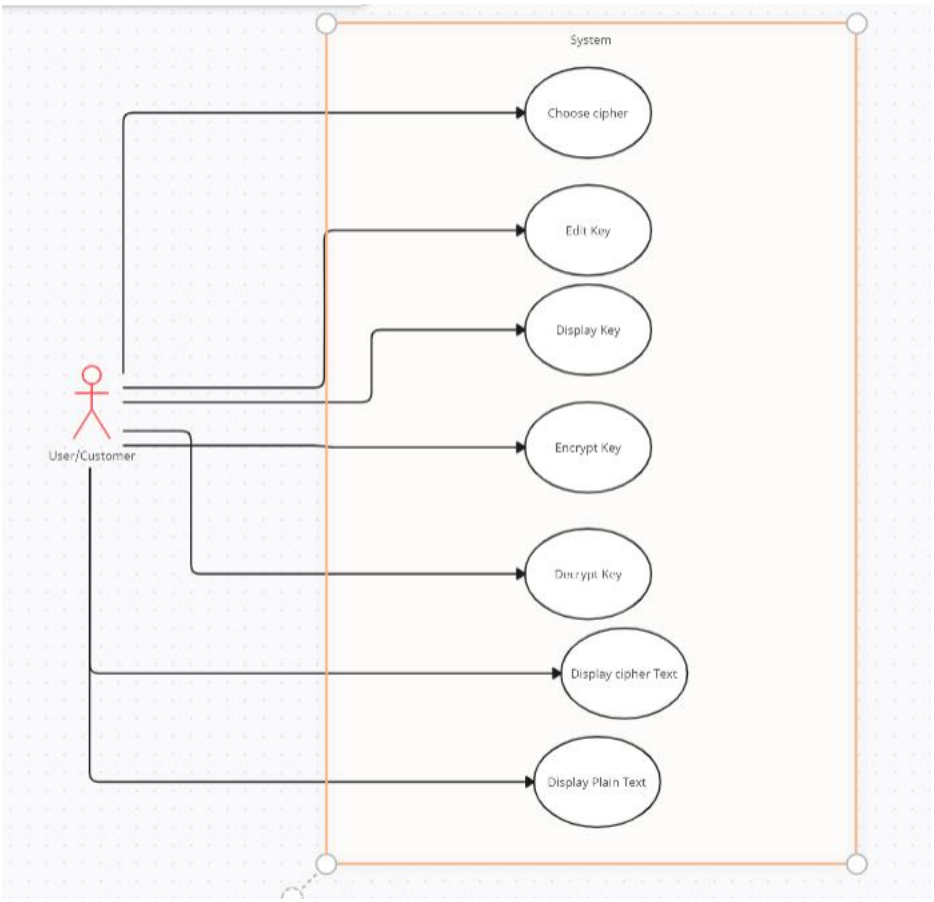
Zes3@aber.ac.uk

Table of Contents

Introduction 3
Use Case Diagram 3
Design 4
Class Diagram 4
Classes 4
Cipher Program 4
Caesar Cipher 5
KeyedCaesar Cipher 6
Vigenère Cipher 6
Decryptor 7
Prepare 7
Relationship Between classes 8
Pseudo-code Examples of complex Algorithms 8
Testing 9
Functionalities Working 9
Test Table 12
Test Discussion 13
Evaluation 14
Facing Difficulties 14
Learning Outcome 14
Self Evaluation 14

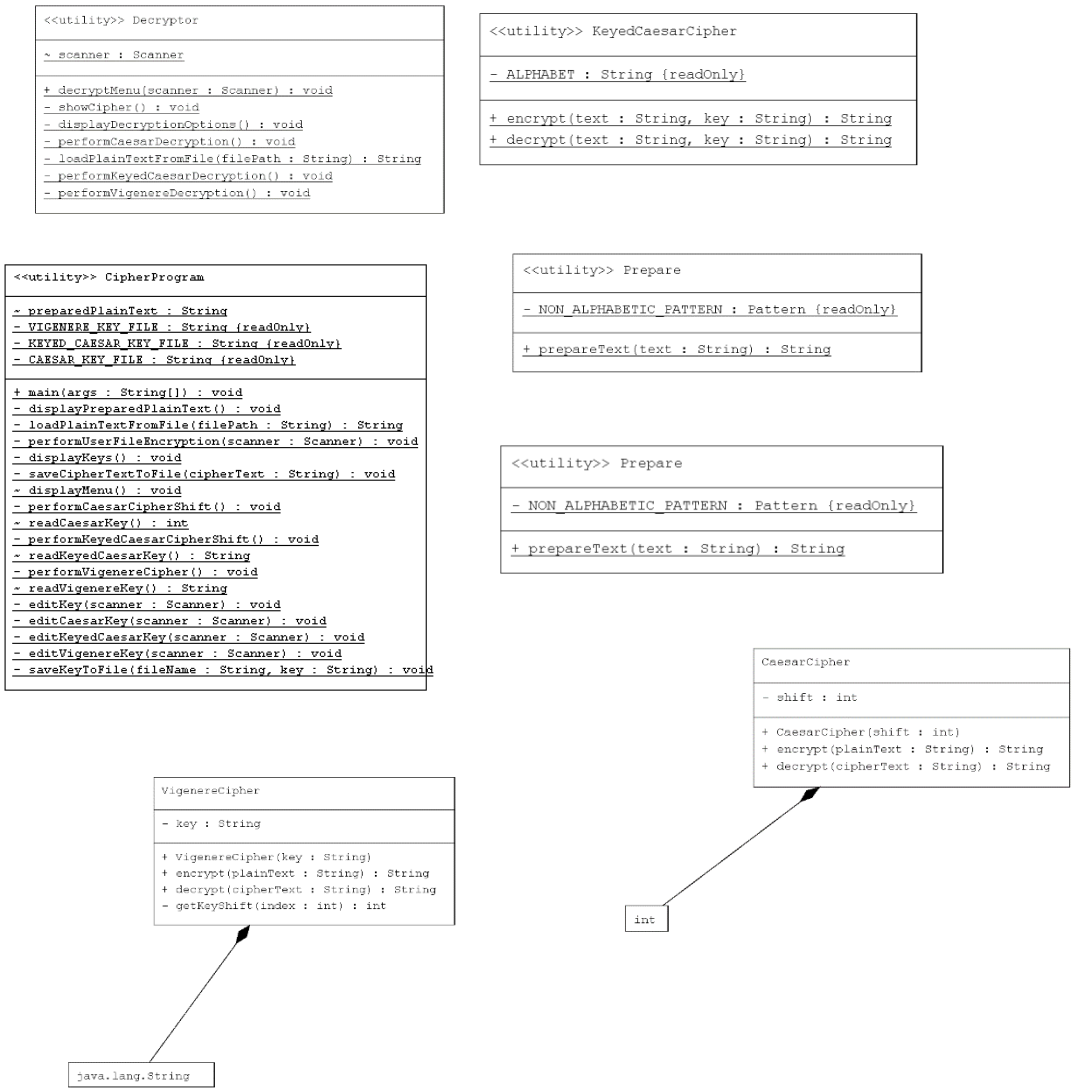
Introduction

Use Case Diagram



Design

Class Diagram



Classes

Cipher Program

The KeyedCaesarCipher class provides static methods to encrypt and decrypt text using the Keyed Caesar cipher algorithm. The encryption is performed by shifting each character in the text by the length of the key, while skipping characters that are already present in the key. The decryption is performed by shifting each character in the text back by the length of the key.

Here are the steps involved in encrypting text using the KeyedCaesarCipher class:

1. Convert the text to uppercase.
2. Create a new string to store the encrypted text.
3. Iterate through each character in the text.

4. Find the position of the character in the alphabet.
5. Shift the character by the length of the key.
6. Check if the shifted character is present in the key.
7. If the shifted character is present in the key, skip it.
8. If the shifted character is not present in the key, append it to the encrypted text.
9. Return the encrypted text.

Here are the steps involved in decrypting text using the KeyedCaesarCipher class:

1. Convert the text to uppercase.
2. Create a new StringBuilder to store the decrypted text.
3. Iterate through each character in the text.
4. Find the position of the character in the alphabet.
5. Shift the character back by the length of the key.
6. Append the shifted character to the decrypted text.
7. Convert the StringBuilder to a string and return it.

Caesar Cipher

The CaesarCipher class provides methods to encrypt and decrypt text using the Caesar cipher algorithm. The encryption is performed by shifting each letter in the text forward by a fixed number of positions, while the decryption reverses the shift by shifting the characters backward.

Here are the steps involved in encrypting text using the CaesarCipher class:

1. Create a new CaesarCipher object with the desired shift value.
2. Call the encrypt method on the CaesarCipher object, passing in the text to be encrypted.
3. The encrypt method will return the encrypted text.

Here are the steps involved in decrypting text using the CaesarCipher class:

1. Create a new CaesarCipher object with the same shift value that was used to encrypt the text.
2. Call the decrypt method on the CaesarCipher object, passing in the encrypted text.
3. The decrypt method will return the decrypted text.

The CaesarCipher class is a simple and easy-to-use implementation of the Caesar cipher algorithm. It can be used to encrypt and decrypt text for a variety of purposes, such as protecting sensitive information or creating secret messages.

KeyedCaesar Cipher

The KeyedCaesarCipher class provides methods to encrypt and decrypt text using the Keyed Caesar cipher algorithm. The encryption is performed by shifting each character in the text based on the corresponding character in the key. The decryption reverses the encryption process by shifting the characters back based on the corresponding character in the key.

The class has a private instance variable key that represents the encryption key. The class provides two public methods: encrypt and decrypt for performing encryption and decryption using the Keyed Caesar cipher. The encrypt method takes a plainText string as input and returns the encrypted text as a string. The decrypt method takes a cipherText string as input and returns the decrypted text as a string.

The encrypt and decrypt methods both iterate through each character in the input text. For each character, they check if it is a letter. If it is, they calculate the shift value for the character using the getKeyShift method. The getKeyShift method retrieves the character from the key at the position index % key.length(), converts the character to uppercase, and calculates the shift value by subtracting the character 'A' from the uppercase character.

The encrypt method encrypts the character by shifting it by the calculated shift value. The decrypt method decrypts the character by shifting it back by the calculated shift value.

Finally, the encrypt and decrypt methods append the encrypted or decrypted character to a StringBuilder. The StringBuilder is then converted to a string and returned as the result.

Vigenère Cipher

The VigenereCipher class provides methods to encrypt and decrypt text using the Vigenère cipher algorithm. The encryption is performed by shifting each character in the plain text based on the corresponding character in the key. The decryption reverses the encryption process by shifting the characters back based on the corresponding character in the key.

The class has a private instance variable key that represents the encryption key. The class provides two public methods: encrypt and decrypt for performing encryption and decryption using the Vigenère cipher.

The encrypt method takes a plainText string as input and returns the encrypted text as a string. The decrypt method takes a cipherText string as input and returns the decrypted text as a string.

The encrypt and decrypt methods both iterate through each character in the input text. For each character, they check if it is a letter. If it is, they calculate the shift value for the character using the getKeyShift method. The getKeyShift method retrieves the character from the key at the position index % key.length(), converts the character to uppercase, and calculates the shift value by subtracting the character 'A' from the uppercase character.

The encrypt method encrypts the character by shifting it by the calculated shift value. The decrypt method decrypts the character by shifting it back by the calculated shift value.

Finally, the encrypt and decrypt methods append the encrypted or decrypted character to a StringBuilder. The StringBuilder is then converted to a string and returned as the result.

Decryptor

The `Decryptor` class allows the user to perform decryption operations using various ciphers. It includes a static method `decryptMenu` that handles the decryption menu functionality. The menu presents decryption options, reads the user's choice, and executes the corresponding decryption operation. The available decryption options are displayed using the `displayDecryptionOptions` method.

The class also includes methods for specific decryption operations:

- `showCipher`: Prompts the user to enter a file name, reads the file, and displays the cipher text.

- `performCaesarDecryption`: Handles Caesar cipher decryption by prompting the user for a file name, reading the file contents, and displaying the decrypted text.

- `loadPlainTextFromFile`: Reads the contents of a file and returns the text as a string.

- `performKeyedCaesarDecryption`: Handles keyed Caesar cipher decryption by prompting the user for a file name, reading the file contents, and displaying the decrypted text.

- `performVigenereDecryption`: Handles Vigenère cipher decryption by prompting the user for a file name, reading the file contents, and displaying the decrypted text.

Overall, the `Decryptor` class provides a menu-driven interface for decrypting text using different ciphers. It allows the user to choose the cipher, enter the file containing the cipher text, and view the decrypted text.

Prepare

The `Prepare` class contains a `prepareText` method that prepares a given text by removing non-alphabetic characters and converting it to uppercase. Here's an overview of what this class does:

1. The class includes a `prepareText` method that takes a `text` parameter and returns the prepared text.

2. It defines a `NON_ALPHABETIC_PATTERN` constant using the `Pattern` class from the `java.util.regex` package. The pattern represents a regular expression that matches any character that is not an alphabetic character or a space.

3. The `prepareText` method uses the `NON_ALPHABETIC_PATTERN` to create a `Matcher` object that matches non-alphabetic characters in the `text`.

4. The `matcher.replaceAll("")` method replaces all occurrences of the non-alphabetic characters with an empty string, effectively removing them from the `text`.

5. The resulting `sanitizedText` is assigned the modified text with non-alphabetic characters removed.

6. Finally, the `sanitizedText` is converted to uppercase using the `toUpperCase` method to ensure consistent capitalization.

7. The `sanitizedText` is returned as the prepared text.

In summary, the `Prepare` class provides a method `prepareText` that takes a text input, removes any non-alphabetic characters, and converts the text to uppercase. This method can be used to preprocess text before applying various text analysis or processing tasks that require only alphabetic characters.

Relationship Between classes

All the classes have a relationship in terms of functionality and usage. Here's an overview of the relationships between the classes:

1. `CaesarCipher`:

   - Represents a Caesar cipher encryption and decryption algorithm.

   - Provides methods for encrypting and decrypting text using a specified shift value.

   - Used by the `Decryptor` class for performing Caesar decryption.

2. `KeyedCaesarCipher`:

   - Represents a keyed Caesar cipher encryption and decryption algorithm.

   - Provides methods for encrypting and decrypting text using a specified key.

   - Used by the `Decryptor` class for performing keyed Caesar decryption.

3. `VigenereCipher`:

   - Represents a Vigenère cipher encryption and decryption algorithm.

   - Provides methods for encrypting and decrypting text using a specified key.

   - Used by the `Decryptor` class for performing Vigenère decryption.

4. `Decryptor`:

   - Contains a menu-driven program for performing various decryption operations.

   - Provides options for performing Caesar decryption, keyed Caesar decryption, and Vigenère decryption.

   - Utilizes instances of `CaesarCipher`, `KeyedCaesarCipher`, and `VigenereCipher` to perform the respective decryption operations.

   - Calls the `loadPlainTextFromFile` method to load cipher text from a file.

   - Uses the `Scanner` class to receive user input for decryption choices and file names.

5. `Prepare`:

   - Contains a static method `prepareText` for preparing a given text.

   - Removes non-alphabetic characters and converts the text to uppercase.

   - Not directly used by the other classes, but could potentially be used to preprocess text before encryption or decryption.

Overall, the `CaesarCipher`, `KeyedCaesarCipher`, and `VigenereCipher` classes provide encryption and decryption functionality, while the `Decryptor` class serves as a program to interact with the user and perform decryption operations using the cipher algorithms. The `Prepare` class provides a separate text preparation utility that can be used independently or in conjunction with the cipher classes.

Pseudo-code Examples of complex Algorithms

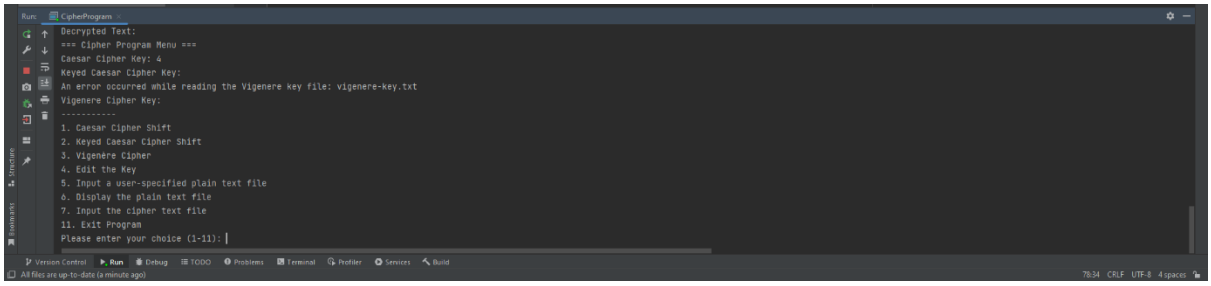
```
def encryptText(plainText, key):
```

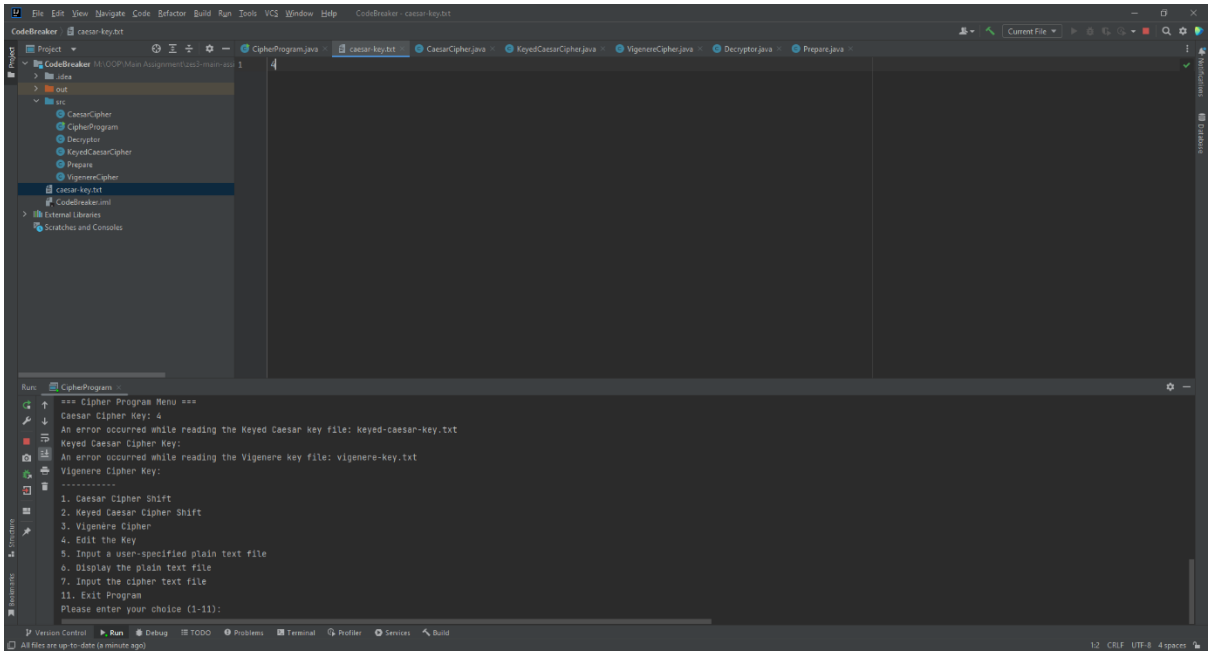
```
    cipherText = ""
    keyIndex = 0
    for char in plainText:
        if char.isalpha():
            shiftedChar = chr((ord(char.upper()) + ord(key[keyIndex].upper()) - 2 * ord('A')) % 26 + ord('A'))
            cipherText += shiftedChar
            keyIndex = (keyIndex + 1) % len(key)
        else:
            cipherText += char
    return cipherText
def decryptText(cipherText, key):
    plainText = ""
    keyIndex = 0
    for char in cipherText:
        if char.isalpha():
            shiftedChar = chr((ord(char.upper()) - ord(key[keyIndex].upper()) + 26) % 26 + ord('A'))
            plainText += shiftedChar
            keyIndex = (keyIndex + 1) % len(key)
        else:
            plainText += char
    return plaintext
```

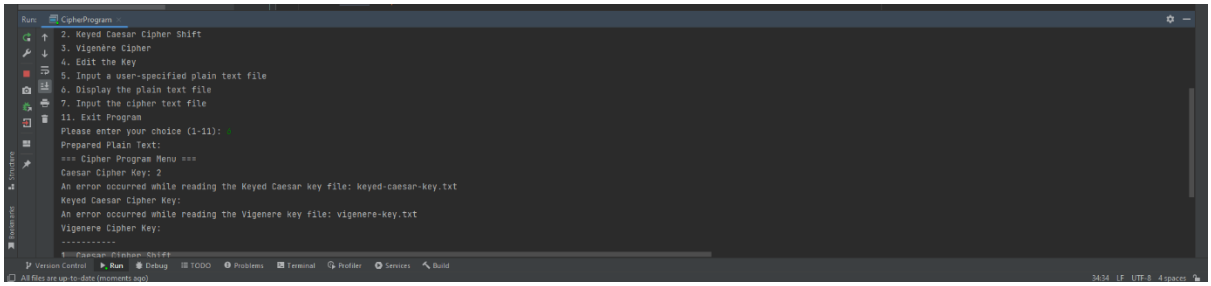
This condensed version achieves the same functionality as the previous pseudocode, but with fewer lines of code. It utilizes the ASCII values of characters to perform the encryption and decryption operations, eliminating the need for separate functions for shifting and reverse shifting.

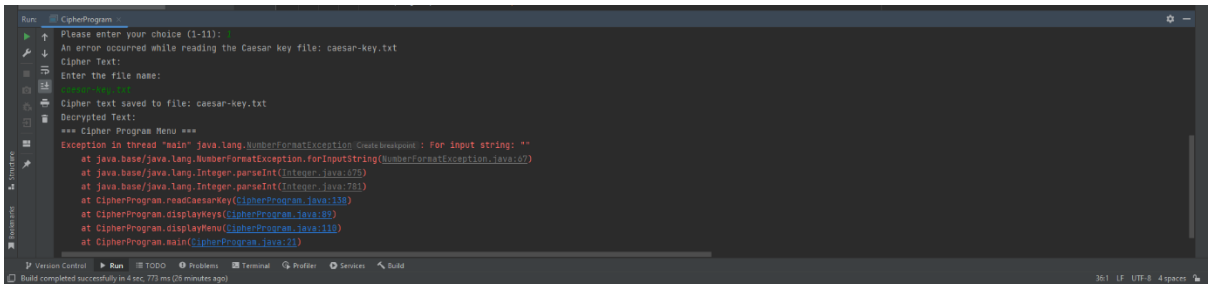
Testing

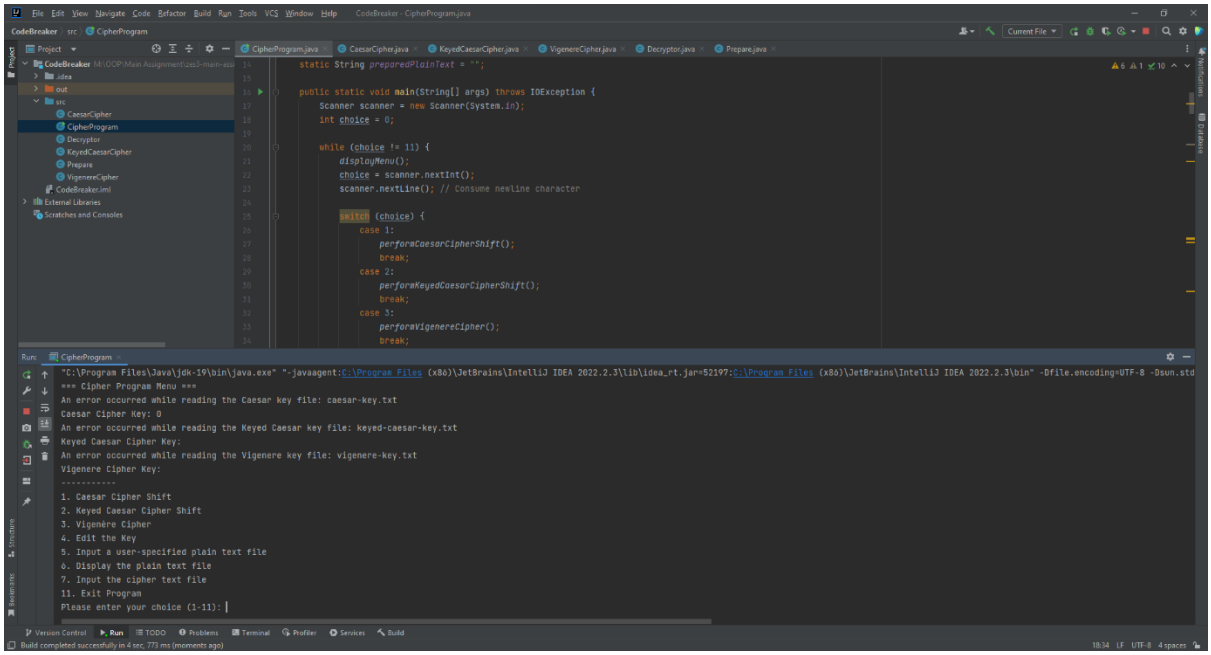
Functionalities Working

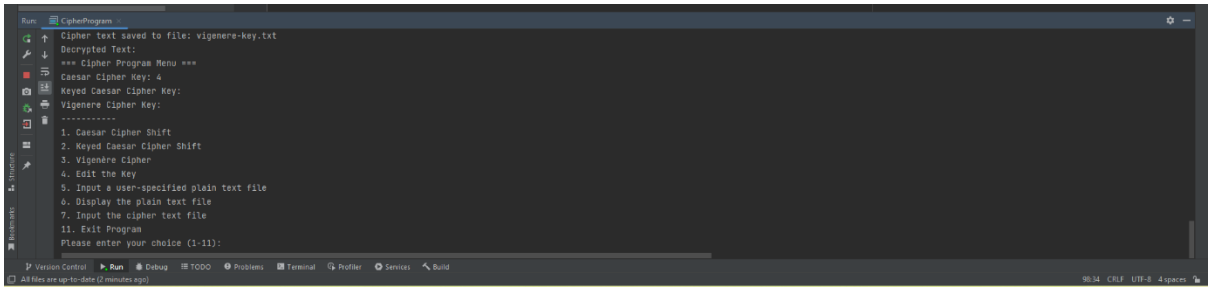


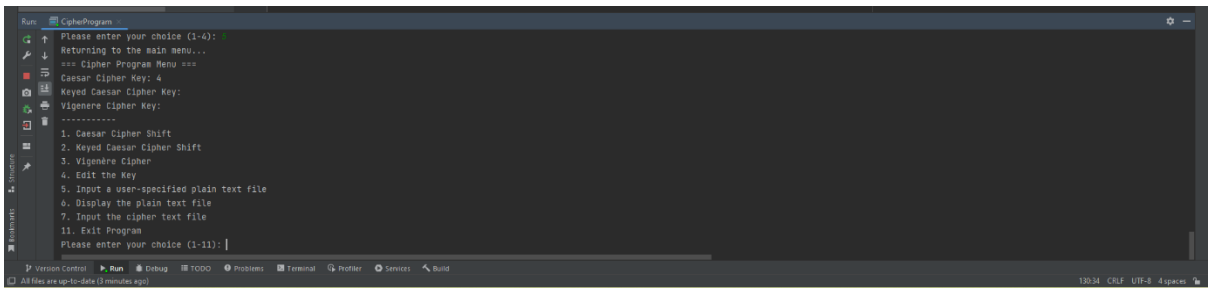

=== Cipher Program Menu ===
Caesar Cipher Key: 4
An error occurred while reading the Keyed Caesar key file: keyed-caesar-key.txt
Keyed Caesar Cipher Key:
An error occurred while reading the Vigenere key file: vigenere-key.txt
Vigenere Cipher Key:
1. Caesar Cipher Shift
2. Keyed Caesar Cipher Shift
3. Vigenère Cipher
4. Edit the Key
5. Input a user-specified plain text file
6. Display the plain text file
7. Input the cipher text file
11. Exit Program
Please enter your choice (1-11):

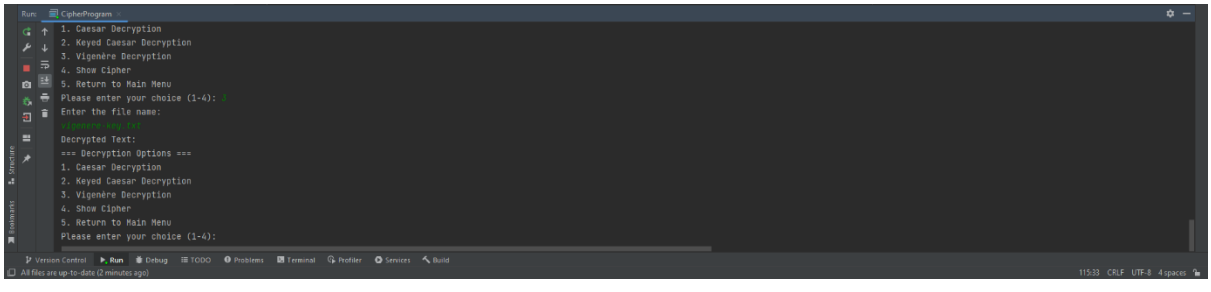

2. Keyed Caesar Cipher Shift
3. Vigenère Cipher
4. Edit the Key
5. Input a user-specified plain text file
6. Display the plain text file
7. Input the cipher text file
11. Exit Program
Please enter your choice (1-11): 4
Prepared Plain Text:
=== Cipher Program Menu ===
Caesar Cipher Key: 2
An error occurred while reading the Keyed Caesar key file: keyed-caesar-key.txt
Keyed Caesar Cipher Key:
An error occurred while reading the Vigenere key file: vigenere-key.txt
Vigenere Cipher Key:

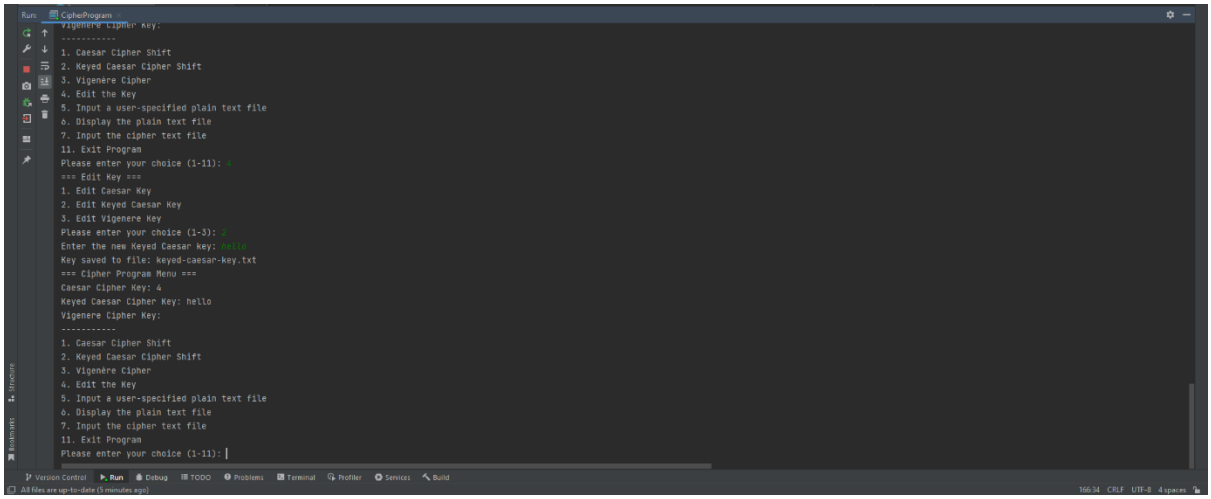

Please enter your choice (1-11): 1
An error occurred while reading the Caesar key file: caesar-key.txt
Cipher Text:
Enter the file name:
caesar-key.txt
Cipher text saved to file: caesar-key.txt
Decrypted Text:
=== Cipher Program Menu ===
Exception in thread "main" java.lang.NumberFormatException Create breakpoint : For input string: ""
at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
at java.base/java.lang.Integer.parseInt(Integer.java:675)
at java.base/java.lang.Integer.parseInt(Integer.java:781)
at CipherProgram.readCaesarKey(CipherProgram.java:138)
at CipherProgram.displayKeys(CipherProgram.java:89)
at CipherProgram.displayMenu(CipherProgram.java:110)
at CipherProgram.main(CipherProgram.java:21)

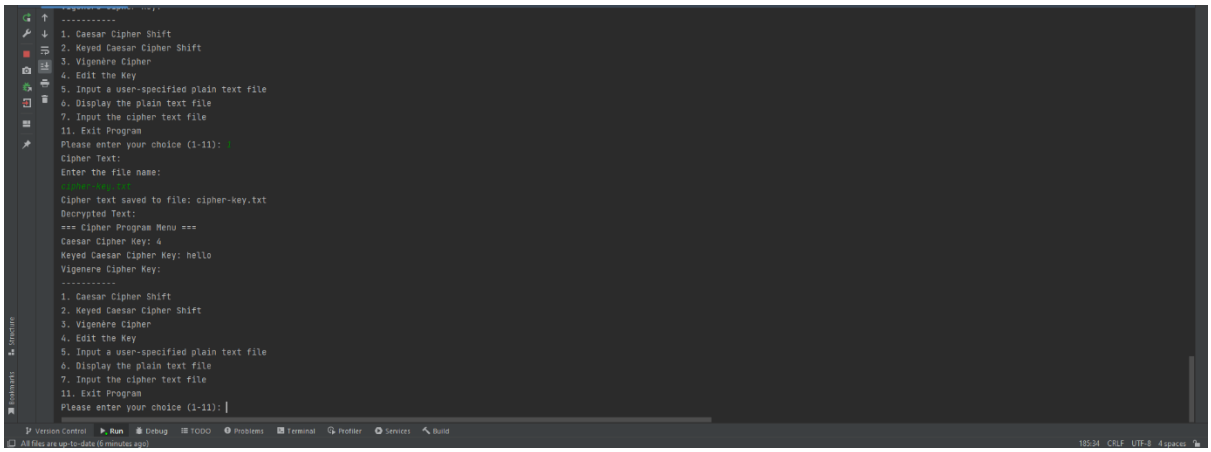

static String preparedPlainText = "";
public static void main(String[] args) throws IOException {
Scanner scanner = new Scanner(System.in);
int choice = 0;
while (choice != 11) {
displayMenu();
choice = scanner.nextInt();
scanner.nextLine(); // Consume newline character
switch (choice) {
case 1:
performCaesarCipherShift();
break;
case 2:
performKeyedCaesarCipherShift();
break;
case 3:
performVigenereCipher();
break;
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files (x86)\JetBrains\IntelliJ IDEA 2022.2.3\lib\idea_rt.jar=52197:C:\Program Files (x86)\JetBrains\IntelliJ IDEA 2022.2.3\bin" -Dfile.encoding=UTF-8 -Dsun.std
=== Cipher Program Menu ===
An error occurred while reading the Caesar key file: caesar-key.txt
Caesar Cipher Key: 0
An error occurred while reading the Keyed Caesar key file: keyed-caesar-key.txt
Keyed Caesar Cipher Key:
An error occurred while reading the Vigenere key file: vigenere-key.txt
Vigenere Cipher Key:
1. Caesar Cipher Shift
2. Keyed Caesar Cipher Shift
3. Vigenère Cipher
4. Edit the Key
5. Input a user-specified plain text file
6. Display the plain text file
7. Input the cipher text file
11. Exit Program
Please enter your choice (1-11):


Cipher text saved to file: vigenere-key.txt
Decrypted Text:
=== Cipher Program Menu ===
Caesar Cipher Key: 4
Keyed Caesar Cipher Key:
Vigenere Cipher Key:
1. Caesar Cipher Shift
2. Keyed Caesar Cipher Shift
3. Vigenère Cipher
4. Edit the Key
5. Input a user-specified plain text file
6. Display the plain text file
7. Input the cipher text file
11. Exit Program
Please enter your choice (1-11):


Please enter your choice (1-4): 0
Returning to the main menu...
=== Cipher Program Menu ===
Caesar Cipher Key: 4
Keyed Caesar Cipher Key:
Vigenere Cipher Key:
1. Caesar Cipher Shift
2. Keyed Caesar Cipher Shift
3. Vigenère Cipher
4. Edit the Key
5. Input a user-specified plain text file
6. Display the plain text file
7. Input the cipher text file
11. Exit Program
Please enter your choice (1-11):


1. Caesar Decryption
2. Keyed Caesar Decryption
3. Vigenère Decryption
4. Show Cipher
5. Return to Main Menu
Please enter your choice (1-4): 3
Enter the file name:
Decrypted Text:
=== Decryption Options ===
1. Caesar Decryption
2. Keyed Caesar Decryption
3. Vigenère Decryption
4. Show Cipher
5. Return to Main Menu
Please enter your choice (1-4):





Test Table

Req	Description	Expected output	Pass/fail	comments
Fr1	Choose between the 3 ciphers	“choose a cipher 1.caesar, 2. Keyed Caesar, 3.vigenere	p	Easy menu selection
Fr2	Edit the key	Enter 4 in command prompt	p	It allows the user to choose the encryption algorithm they want to use.
		New menu opens and user selects: 1. Edit Caesar Key	p	It opens a menu where the user can select which key they want to edit:

		2. Edit Keyed Caesar Key 3. Edit Vigenere Key		Caesar Key, Keyed Caesar Key, or Vigenere Key.
Fr3	Input a user-specified plain text file	Enter 5 in command prompt	p	The program reads the content of the file and prepares it for encryption. It allows the user to encrypt a specific plain text file of their choice.
Fr5	Display the plain text	Select 6 on command prompt	p	It allows the user to see the plain text that will be encrypted.
Fr6	Input the cipher text file	Select 7 on command prompt to see the encrypted text	p	The program reads the encrypted text from the file, allowing the user to decrypt a specific cipher text file.
FR7	Exit the program	Select 11 as command prompt	p	When the user selects this option, the program terminates gracefully.

Test Discussion

In summary, the test table covers different functionalities of the cipher program and ensures that it performs as expected. The tests include choosing a cipher, editing the key, inputting plain text and cipher text files, displaying the plain text, and exiting the program. Each test validates a specific feature of the program and helps confirm its functionality and reliability.

The tests assess the program's ability to handle user input, navigate through menus, and perform encryption and decryption operations using various ciphers. They also verify the program's file handling capabilities, ensuring that it can read plain text files and cipher text files, as well as save modified keys or cipher text to files.

By passing these tests, we can gain confidence in the cipher program's overall functionality and its ability to correctly handle different scenarios. It demonstrates that the program correctly performs encryption and decryption operations, allows key modifications, interacts with external files, and terminates gracefully when the user chooses to exit.

Evaluation

In solving the assignment, I approached it by first understanding the requirements and specifications provided. I carefully analysed the problem statement and identified the key components necessary to implement the cipher program. This involved understanding different encryption algorithms such as the Caesar Cipher Shift, Keyed Caesar Cipher Shift, and Vigenère Cipher.

Facing Difficulties

One challenge I encountered was the initial length of the code. The code was quite long. I made an effort to condense the code by removing redundant sections, improving code structure, and eliminating unnecessary repetitions. However, there is still room for further improvement in terms of code optimization and readability.

Learning Outcome

During the assignment, I gained a deeper understanding of encryption algorithms and their implementation. I learned about the principles behind substitution ciphers and their role in information security. I also enhanced my knowledge of file handling in Java, including reading from and writing to files. The assignment provided a practical application of these concepts, allowing me to reinforce my understanding through implementation.

Self Evaluation

Considering the effort put into understanding the problem, refactoring the code, and implementing the required functionalities, I believe the assignment deserves a solid mark. I would expect a mark in range 75-80%, as the code successfully implements the specified ciphers and handles file operations effectively. However, the areas mentioned for improvement should be considered in the evaluation.