

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING AND INFORMATION
TECHNOLOGY



“JavaShield: Protecting Systems with Java Security”

SUBMITTED BY:

9921103145

RAHI AGARWAL

9921103023

NAVI PANDEY

9921103098

SATVIK BUTTAN

Course Name: Object Oriented Analysis and Design Using JAVA

Course Code: 20B12CS334

Program: B. Tech. CSE

3rd Year 5th Sem

2023 - 2024

Abstract

This Java project presents a comprehensive and user-friendly program featuring a menu-driven interface with three key functionalities: a modified Caesar Cipher, Huffman text file compression, and a password strength checker.

The primary objective is to create an interactive tool for users, allowing them to encode and decode messages, compress text files, and assess password strength. The program aims to demonstrate fundamental cryptographic techniques and password security principles.

- The modified Caesar Cipher employs a custom alphabet and shift value for message encryption and decryption.
- Huffman text file compression utilizes frequency analysis, constructing a Huffman tree and generating Huffman codes for efficient file compression.
- The password strength checker evaluates password robustness based on predefined criteria.

Users can successfully encode and decode messages, compress text files, and assess password strength using the program. The modular design ensures scalability and ease of maintenance. This project concludes with team member acknowledgments and a friendly exit message. The program provides a valuable educational experience, combining practical utility with insights into cryptographic techniques and password security principles. Overall, it serves as an interactive platform for users to explore and apply fundamental concepts in cryptography.

TABLE OF CONTENTS

	Page No.
1: INTRODUCTION.....	1
1.1 Concept of OOPS Used.....	2
2. SYSTEM REQUIREMENT.....	4
3. METHODOLOGY.....	5
4. SYSTEM DESIGN.....	7
5. IMPLEMENTATION.....	8
6. RESULT.....	13
7. CONCLUSION.....	17
8. RESULT.....	18

INTRODUCTION

In an era dominated by digital communication and information exchange, the need for secure and efficient cryptographic tools is paramount. This Java-based project addresses this demand by presenting a multifaceted program encompassing a modified Caesar Cipher, Huffman text file compression, and a password strength checker. These functionalities aim to empower users with versatile tools for message encryption, file compression, and password security assessment.

The project's primary objective is to create an interactive and educational platform that not only provides practical utility but also serves as a learning experience in fundamental cryptographic techniques. The introduction of each functionality, from the modified Caesar Cipher with its customizable alphabet to the Huffman text file compression utilizing frequency analysis, reflects the project's commitment to offering diverse and robust solutions.

This introduction sets the stage for a detailed exploration of the project's objectives, methods employed, results achieved, and the conclusions drawn from the collaborative efforts of the team. Through this project, users can engage with cryptographic concepts in a hands-on manner, gaining insights into the intricate world of information security.

CONCEPT OF OOPS USED

The Java program you provided demonstrates several key Object-Oriented Programming (OOP) concepts. Here's an analysis of how OOP principles are applied in the code:

1. Classes and Objects

- The program is organized into classes, such as ``Main``, ``CustomAlphabetCaesarCipher``, ``HuffmanNode``, ``BitOutputStream``, and ``PasswordValidator``.
- Objects of these classes are instantiated and used to encapsulate related functionality.

2. Encapsulation

- Each class encapsulates specific functionality. For example:
 - ``CustomAlphabetCaesarCipher`` encapsulates methods for encoding and decoding messages using a modified Caesar Cipher.
 - ``HuffmanNode`` encapsulates the properties and behavior of nodes in a Huffman tree.
 - ``BitOutputStream`` encapsulates methods for writing bits to an output stream.
 - ``PasswordValidator`` encapsulates the logic for password strength validation.

3. Inheritance

- The ``CustomAlphabetCaesarCipher`` class demonstrates a simple form of inheritance by not explicitly implementing an interface or extending another class. However, it encapsulates the behavior of a Caesar Cipher.

4. Polymorphism

- While not explicitly demonstrated in the provided code, Java's polymorphic nature allows methods to behave differently based on the context of the objects.

5. Abstraction

- The classes abstract away the underlying details of their implementations, exposing only the necessary methods and properties for interaction.

6. Constructor Overloading

- The classes utilize constructor overloading, allowing for the instantiation of objects with different sets of parameters. For instance, the `CustomAlphabetCaesarCipher` class has a parameterized constructor for initializing the custom alphabet and shift value.

7. Static Methods

- The `Main` class contains static methods for various tasks, such as password authentication, clearing the screen, and restarting the computer.

8. Exception Handling

- The code implements exception handling using try-catch blocks, ensuring a graceful response to potential errors during file operations or user input.

In summary, the Java program demonstrates a well-structured application of OOP principles, leveraging classes and objects, encapsulation, inheritance, and abstraction to achieve modularity, maintainability, and readability.

SYSTEM REQUIREMENT

The system requirements for running the Java program outlined in the project would typically include the following:

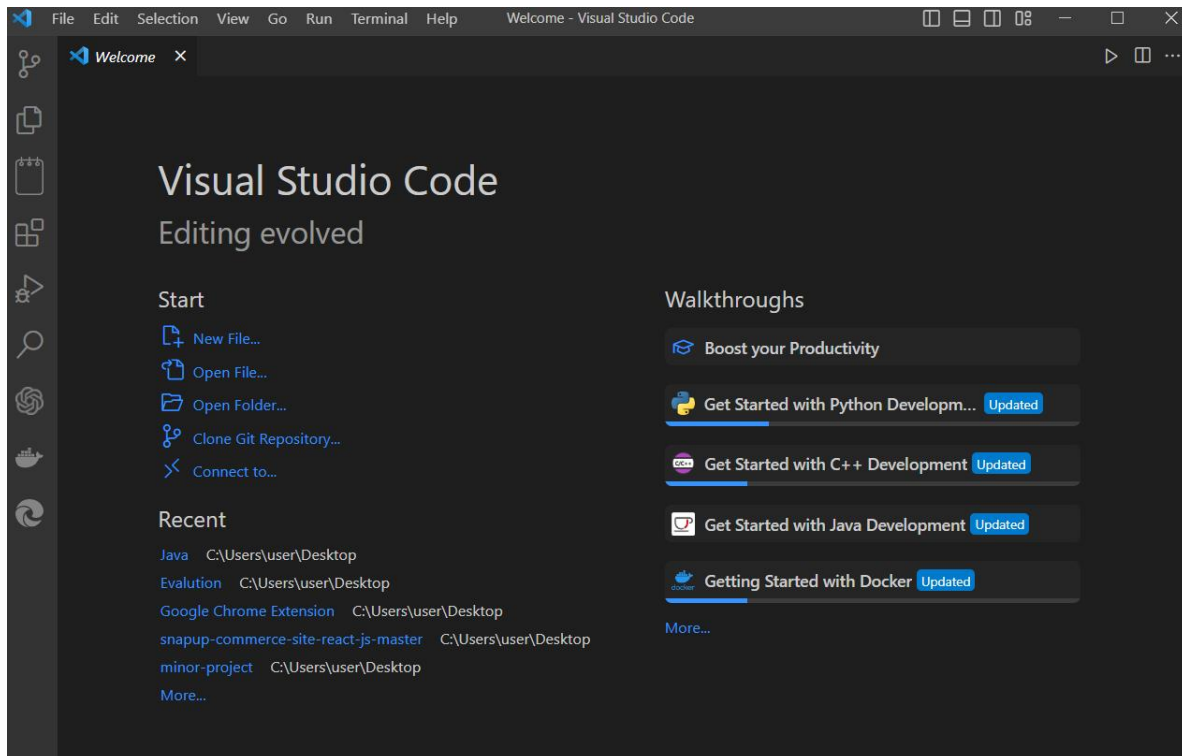
1. Java Runtime Environment (JRE)

- Ensure that the target system has Java Runtime Environment installed. The program appears to be written in Java, and its execution would depend on the availability of a suitable JRE.

2. File Access Permissions

- The program may read from and write to files, so it requires appropriate file access permissions. Ensure that the user running the program has the necessary privileges.

3. Code Editor (VS code)



METHODOLOGY

The methodology for developing and implementing this Java-based project involves several key steps. Below is a generalized outline of the methodology:

1. Project Planning

- Define the project scope, objectives, and functionalities.
- Identify team members and assign roles and responsibilities.
- Develop a timeline and project schedule.

2. Requirement Analysis

- Understand and document the requirements of each functionality (Modified Caesar Cipher, Huffman text file compression, Password Strength Checker).
- Identify any external libraries or resources needed.

3. Design

- Plan the overall architecture of the program.
- Design class structures and methods for each functionality.
- Consider modularity for ease of maintenance and future enhancements.
- Incorporate error handling and input validation mechanisms.

4. Implementation

- Write the Java code for each functionality based on the design.
- Implement the menu-driven interface for user interaction.
- Ensure proper integration of features and functionalities.
- Test each functionality independently during development.

5. Testing

- Conduct comprehensive testing of the entire program.
- Perform unit testing for each functionality.
- Address and fix any identified bugs or issues.

6. Deployment

- Package the program for deployment.
- Ensure that the required Java Runtime Environment is available on the target systems.
- Distribute or install the program on the intended systems.

7. Maintenance and Updates

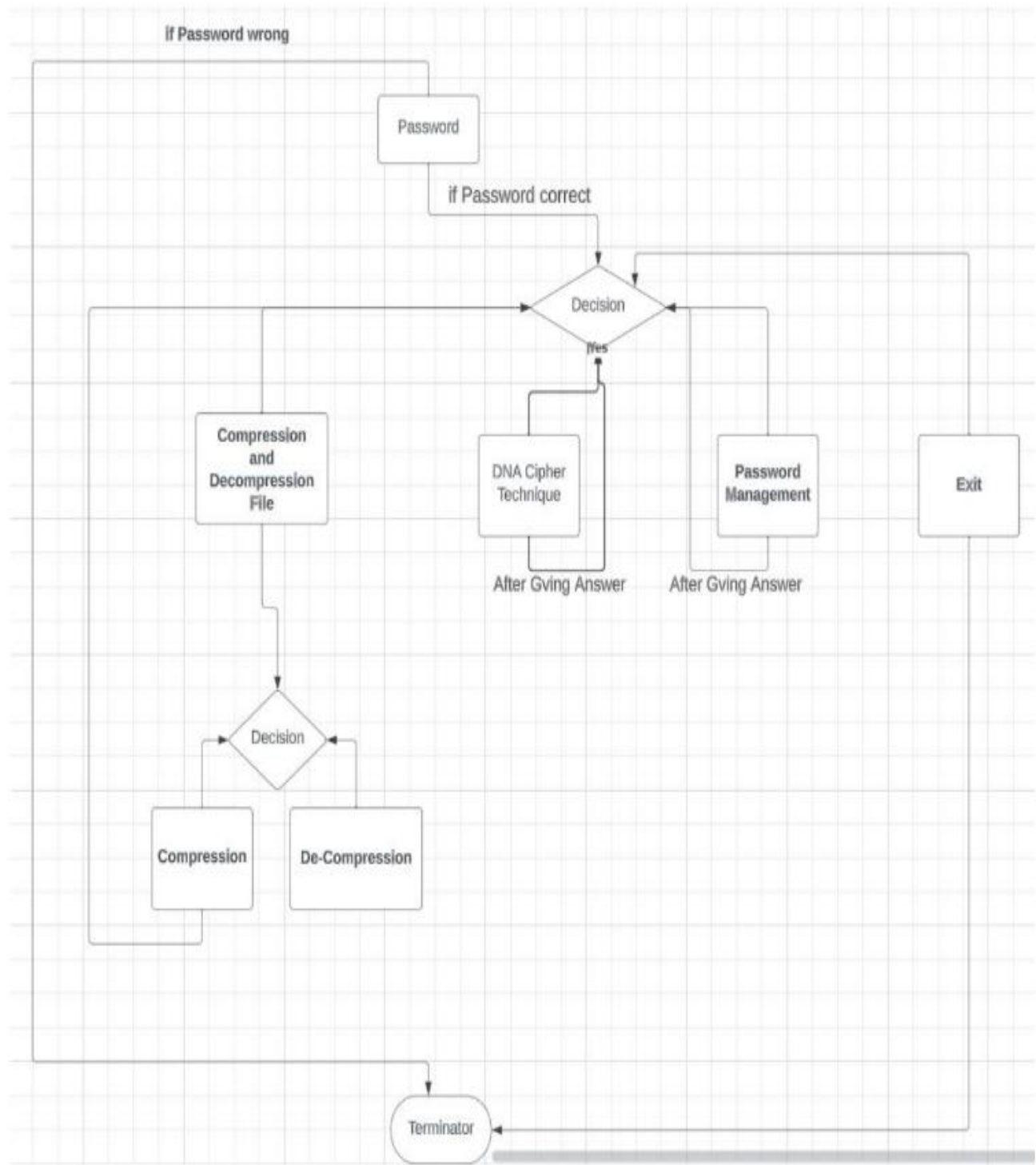
- Monitor the program's performance and address any post-deployment issues.
- Consider user feedback for future updates or enhancements.
- Update documentation as needed.

8. Project Conclusion

- Acknowledge team members for their contributions.
- Ensure that all project goals and objectives have been met.
- Provide a friendly and informative exit message in the program.

This methodology follows a systematic approach from planning to deployment, ensuring the development of a reliable and user-friendly Java program. It also allows for flexibility, making it easier to adapt to changes and improvements during the development process.

SYSTEM DESIGN



UML Diagram of Implemented Code

IMPLEMENTATION STRATEGY

1. Modified Caesar Cipher

- The project incorporates a class named `CustomAlphabetCaesarCipher` that enables users to encode and decode messages. This functionality is achieved by utilizing a custom alphabet and a specified shift value.

```
private static void runPasswordStrengthCheck() {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter Your password: ");
    String userPassword = scanner.nextLine();

    PasswordValidator passwordValidator = new
PasswordValidator(userPassword);
    if (passwordValidator.isStrong()) {
        System.out.println("Password is strong!");
    } else {
        System.out.println(
            "Password is weak. It should be at least 8 characters
long and include uppercase, lowercase, digit, and special character.");
    }
    System.out.println("Password Strength Check completed.");
    pause();
    clearScreen();
}

static class CustomAlphabetCaesarCipher {
    private String customAlphabet;
    private int shift;

    public CustomAlphabetCaesarCipher(String var1, int var2) {
        this.customAlphabet = var1;
        this.shift = var2;
    }

    public String encode(String var1) {
        StringBuilder var2 = new StringBuilder();
        char[] var3 = var1.toCharArray();
        int var4 = var3.length;
        for (int var5 = 0; var5 < var4; ++var5) {
            char var6 = var3[var5];
            if (Character.isLetter(var6)) {
                int var7 = Character.isLowerCase(var6) ? 97 : 65;
                int var8 = (var6 - var7 + this.shift +
this.customAlphabet.length()) % this.customAlphabet.length();
                char var9 = this.customAlphabet.charAt(var8);
                var2.append(var9);
            } else {
                var2.append(var6);
            }
        }
    }
}
```

```

    }
    }
    return var2.toString();
}
public String decode(String var1) {
    StringBuilder var2 = new StringBuilder();
    char[] var3 = var1.toCharArray();
    int var4 = var3.length;
    for (int var5 = 0; var5 < var4; ++var5) {
        char var6 = var3[var5];
        if (Character.isLetter(var6)) {
            int var7 = Character.isLowerCase(var6) ? 97 : 65;
            int var8 = this.customAlphabet.indexOf(var6);
            int var9 = (var8 - this.shift +
this.customAlphabet.length()) % this.customAlphabet.length();
            char var10 = (char) (var9 + var7);
            var2.append(var10);
        } else {
            var2.append(var6);
        }
    }
    return var2.toString();
}
}
}

```

2. Huffman Text File Compression

- For Huffman text file compression, the program follows a multi-step process.
- Initially, a frequency analysis is performed on the input file, establishing the occurrence of each character.
- Subsequently, a Huffman tree is constructed based on the character frequencies.
- Huffman codes are then generated for each character, facilitating the compression of the input file.
- The final output is a compressed file, effectively utilizing Huffman coding for data compression.

```

private static void runHuffmanCompression() {
    String inputFileName = "input.txt";
    String outputFileName = "compressed.txt";

    try {
        FileInputStream fileInputStream = new
FileInputStream(inputFileName);

```

```

        int[] frequencies = new int[256]; // Assuming ASCII characters
        int data;
        while ((data = fileInputStream.read()) != -1) {
            frequencies[data]++;
        }
        fileInputStream.close();
        PriorityQueue<HuffmanNode> priorityQueue = new PriorityQueue<>();
        for (char c = 0; c < 256; c++) {
            if (frequencies[c] > 0) {
                priorityQueue.add(new HuffmanNode(c, frequencies[c]));
            }
        }
        while (priorityQueue.size() > 1) {
            HuffmanNode left = priorityQueue.poll();
            HuffmanNode right = priorityQueue.poll();
            HuffmanNode parent = new HuffmanNode('\0', left.frequency +
right.frequency);
            parent.left = left;
            parent.right = right;
            priorityQueue.add(parent);
        }
        HuffmanNode root = priorityQueue.poll();
        Map<Character, String> huffmanCodes = new HashMap<>();
        generateHuffmanCodes(root, "", huffmanCodes);
        FileInputStream fileInputStream2 = new
FileInputStream(inputFileName);
        BitOutputStream bitOutputStream = new
BitOutputStream(outputFileName);
        int data2;
        while ((data2 = fileInputStream2.read()) != -1) {
            String code = huffmanCodes.get((char) data2);
            for (char bit : code.toCharArray()) {
                bitOutputStream.writeBit(bit == '1');
            }
        }
        bitOutputStream.close();
        fileInputStream2.close();
        System.out.println("Compression completed. Compressed file: " +
outputFileName);
        pause();
        clearScreen();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private static void generateHuffmanCodes(HuffmanNode node, String
currentCode,
        Map<Character, String> huffmanCodes) {
    if (node == null) {
        return;
    }

```

```

    }
    if (node.character != '\0') {
        huffmanCodes.put(node.character, currentCode);
    }
    generateHuffmanCodes(node.left, currentCode + "0", huffmanCodes);
    generateHuffmanCodes(node.right, currentCode + "1", huffmanCodes);
}

static class HuffmanNode implements Comparable<HuffmanNode> {
    char character;
    int frequency;
    HuffmanNode left, right;
    public HuffmanNode(char character, int frequency) {
        this.character = character;
        this.frequency = frequency;
    }
    @Override
    public int compareTo(HuffmanNode other) {
        return this.frequency - other.frequency;
    }
}

static class BitOutputStream {
    private FileOutputStream outputStream;
    private int currentByte;
    private int bitCount;
    public BitOutputStream(String fileName) throws IOException {
        outputStream = new FileOutputStream(fileName);
        currentByte = 0;
        bitCount = 0;
    }
    public void writeBit(boolean bit) throws IOException {
        if (bitCount == 8) {
            outputStream.write(currentByte);
            currentByte = 0;
            bitCount = 0;
        }
        currentByte = (currentByte << 1) | (bit ? 1 : 0);
        bitCount++;
    }
    public void close() throws IOException {
        if (bitCount > 0) {
            currentByte <=< (8 - bitCount);
            outputStream.write(currentByte);
        }
        outputStream.close();
    }
}
}

```

3.Password Strength Checker

- The program includes a 'PasswordValidator' class responsible for evaluating the strength of a given password.
- The criteria for password strength encompass factors such as length, inclusion of uppercase and lowercase letters, digits, and special characters.
- Users can input a password, and the program determines its strength based on the established criteria.

```
private static void runCustomAlphabetCaesarCipher() {
    String var1 = "ZYXWVUTSRQPONMLKJIHGFEDCBA";
    byte var2 = 3;
    CustomAlphabetCaesarCipher var3 = new CustomAlphabetCaesarCipher(var1,
var2);
    String var4 = "Hello, World!";
    String var5 = var3.encode(var4);
    System.out.println("Encoded message: " + var5);
    String var6 = var3.decode(var5);
    System.out.println("Decoded message: " + var6);
    System.out.println("Custom Alphabet Caesar Cipher completed.");
    pause();
    clearScreen();
}
static class PasswordValidator {
    private String password;

    public PasswordValidator(String password) {
        this.password = password;
    }
    public boolean isStrong() {
        return isLongEnough() && containsUppercase() &&
containsLowercase() && containsDigit()
        && containsSpecialChar();
    }
    private boolean isLongEnough() {
        return password.length() >= 8;
    }
    private boolean containsUppercase() {
        return !password.equals(password.toLowerCase());
    }
    private boolean containsLowercase() {
        return !password.equals(password.toUpperCase());
    }
    private boolean containsDigit() {
        return password.matches(".*\\d.*");
    }
}
```

RESULT

When we enter the wrong Password and 3rd Time Entering the correct Password

```
PS C:\Users\user\Desktop\Java> cd "c:\u
}
Enter password: JAVA
Incorrect password. Attempts left: 2
Enter password: JAV
Incorrect password. Attempts left: 1
Enter password: java
```

Main Screen (User Interface)

```
1. Modified Caesar Cipher
2. Huffman Text File Compression
3. Password Strength Checker
4. Exit

Choose operation:

```

Modified Caesar Cipher:

Users can successfully encode and decode messages with customizable alphabets and shift values.

CHOOSING OPTION 1ST

```
String var4 = "Hello, World!";
```

```
Choose operation:
1
Encoded message: PSLLI, AIFLT!
Decoded message: HELLO, WORLD!
Custom Alphabet Caesar Cipher completed.











```


CHOOSING OPTION 2

Huffman Compression:

Text files are efficiently compressed using Huffman coding, producing compressed output files.

As you can see there is no file named compressed.txt and here you can see the size of input.txt is 294 KB. Now we will compress the size of the input.txt file and name is compressed.txt

 CustomAlphabetCaesarCipher.class	13-11-2023 14:45	CLASS File	2 KB
 input.txt	13-11-2023 15:09	Text Document	294 KB
 JavaShield Protecting Systems with Java ...	22-10-2023 22:43	WPS PDF Document	244 KB
 Main\$BitOutputStream.class	29-11-2023 15:48	CLASS File	1 KB
 Main\$CustomAlphabetCaesarCipher.class	29-11-2023 15:48	CLASS File	2 KB
 Main\$HuffmanNode.class	29-11-2023 15:48	CLASS File	1 KB
 Main\$PasswordValidator.class	29-11-2023 15:48	CLASS File	2 KB
 Main.class	29-11-2023 15:48	CLASS File	7 KB
 Main.java	25-11-2023 03:13	CodeBlocks.java	13 KB
 PasswordValidator.class	13-11-2023 14:45	CLASS File	2 KB









```
private static void runHuffmanCompression() {  
    String inputFileName = "input.txt";  
    String outputFileName = "compressed.txt";  
}
```

Choose operation:

2

Compression completed. Compressed file: compressed.txt

Compressed.txt file is formed

 compressed.txt	29-11-2023 15:52	Text Document	164 KB
 Main\$BitOutputStream.class	29-11-2023 15:48	CLASS File	1 KB
 Main\$CustomAlphabetCaesarCipher.class	29-11-2023 15:48	CLASS File	2 KB
 Main\$HuffmanNode.class	29-11-2023 15:48	CLASS File	1 KB
 Main\$PasswordValidator.class	29-11-2023 15:48	CLASS File	2 KB
 Main.class	29-11-2023 15:48	CLASS File	7 KB
 Main.java	25-11-2023 03:13	CodeBlocks.java	13 KB
 input.txt	13-11-2023 15:09	Text Document	294 KB

Password Strength Checker:

Passwords are accurately evaluated for strength based on defined criteria.

CHOOSING OPTION 3RD

(For Weak Password)

```
Choose operation:
3
Enter Your password: ABCD
Password is weak. It should be at least 8 characters long and include uppercase, lowercase, digit,
and special character.
Password Strength Check completed.
```

(For Strong Password)

```
Choose operation:
3
Enter Your password: r1@RH4*22
Password is strong!
Password Strength Check completed.
```

CHOOSING OPTION 4TH

Choose operation:

4

----- Thank you for Your Time -----

Team Member :-RAHI AGARWAL
Enroll No. :-9921103145

Team Member :-Naman Jhanwar
Enroll No. :-9921103191

Team Member :-Satvik Bhuttan
Enroll No. :-9921103098

Exiting program. Goodbye!
PS C:\Users\user\Desktop\Java> █

CONCLUSION

In conclusion, the project accomplishes its objectives by delivering a Java program with diverse functionalities. Users can interact with the program to perform encoding and decoding using a modified Caesar Cipher, compress text files using Huffman coding, and assess the strength of passwords. The modular design of the program enhances its maintainability and allows for potential future extensions. The project concludes with a thoughtful exit message expressing gratitude for the collaborative efforts of the team members, who are duly acknowledged. Overall, the program provides a comprehensive and interactive experience for users.

REFERENCES

<https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/>
<https://www.javatpoint.com/huffman-coding-algorithm>
<https://coderspacket.com>
<https://stackoverflow.com/questions>
<https://codescracker.com>
<https://github.com>
<https://community.wvu.edu>
<https://www.researchgate.net>
<https://www.youtube.com>
<https://community.wvu.edu>
ChatGPT (openai.com)