Name	**Prakriti Sharma**
Student ID	**A20575259**
Course ID	**CS458**
Title	**Coding Assignment I**

*** *The code is pasted at the end of the document.*

The following code has been implemented in **JAVA** language. The basic functionalities covered are as follows.

1. It prints a menu for the user to select among Encryption, Decryption, and Brute Force attack.

- It enters an infinite loop (until the user inputs 0, viz the exit code).

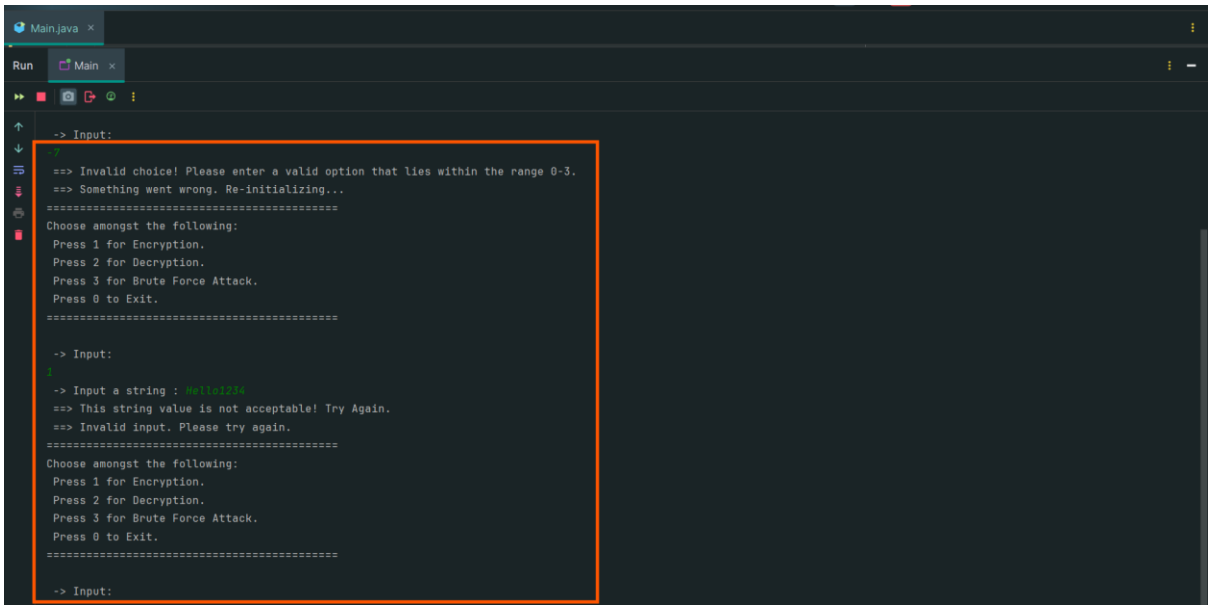
EXIT CODE:

- If the user enters an invalid value, it re-initializes the menu and asks for an input again.

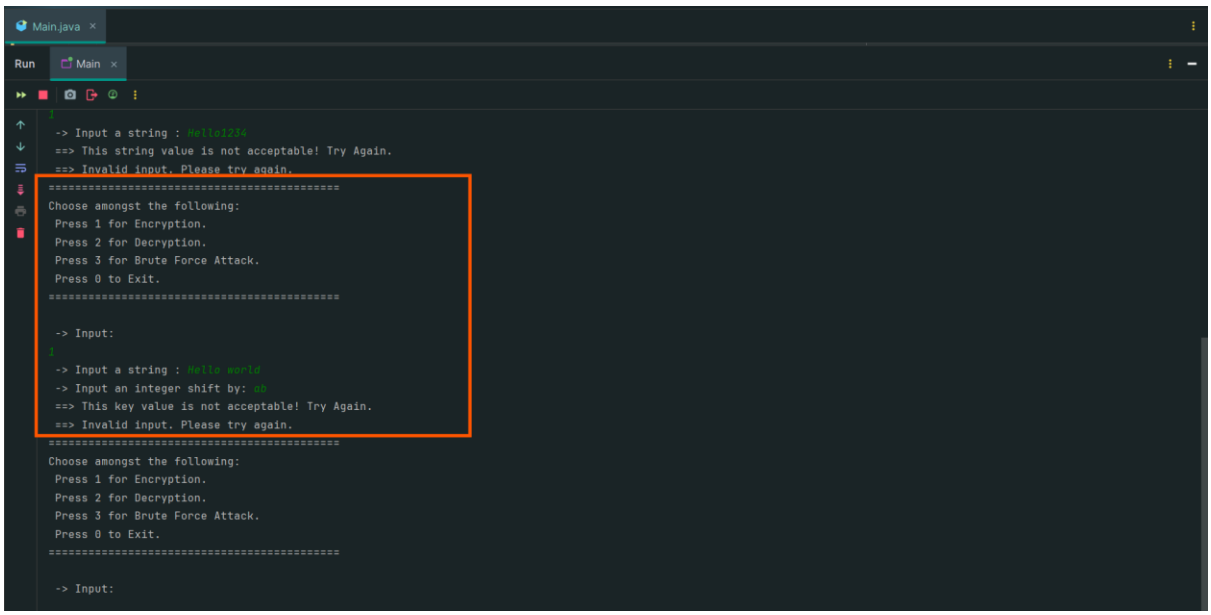
- If the user enters a value between 0-3, the cases are as follows:

  ✓ For ENCRYPTION:
  The function does not allow an invalid input in the String field and the key field.

STRING FIELD VALIDATION:

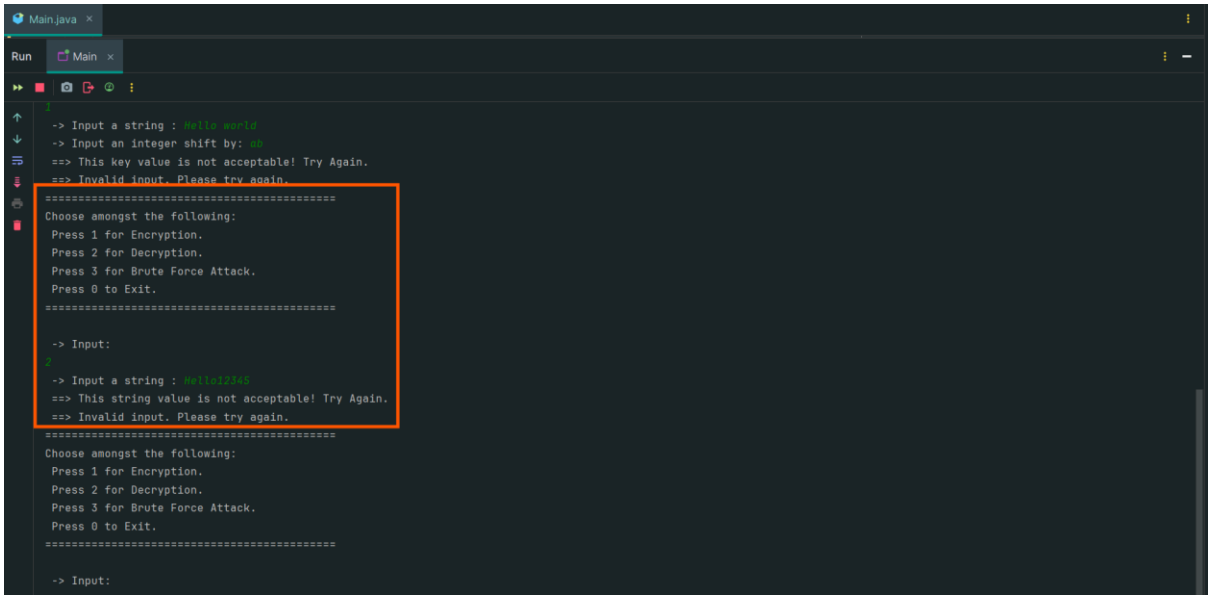


KEY FIELD VALIDATION:



✓ For DECRYPTION:
The function does not allow an invalid input in the String field and the key field.

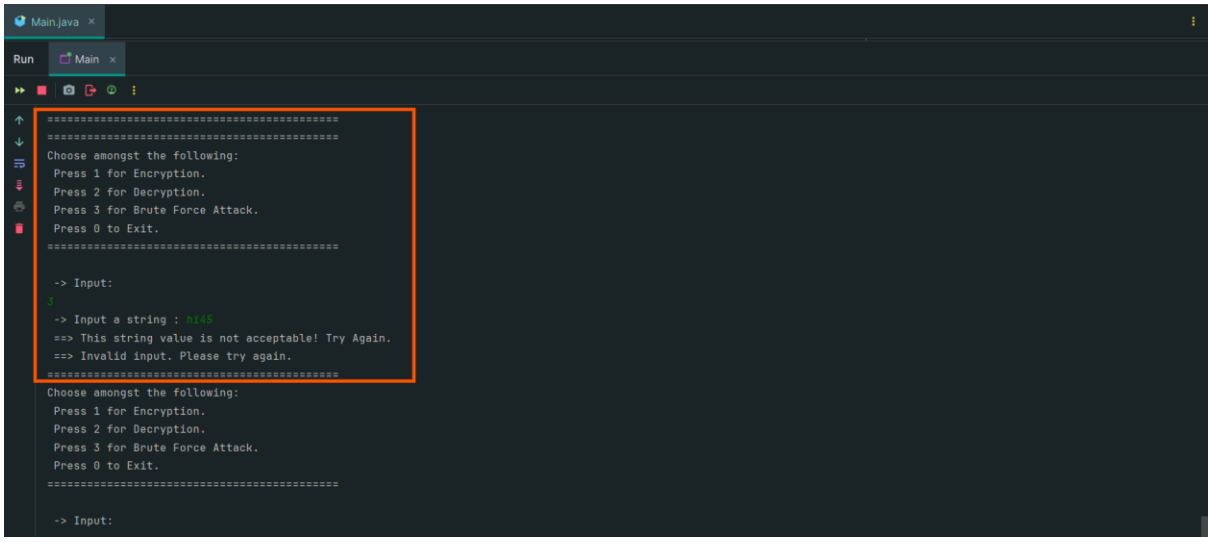
STRING FIELD VALIDATION:



KEY FIELD VALIDATION:

✓ For BRUTE FORCE:
The function does not allow an invalid input in the String field.

VALIDATION:

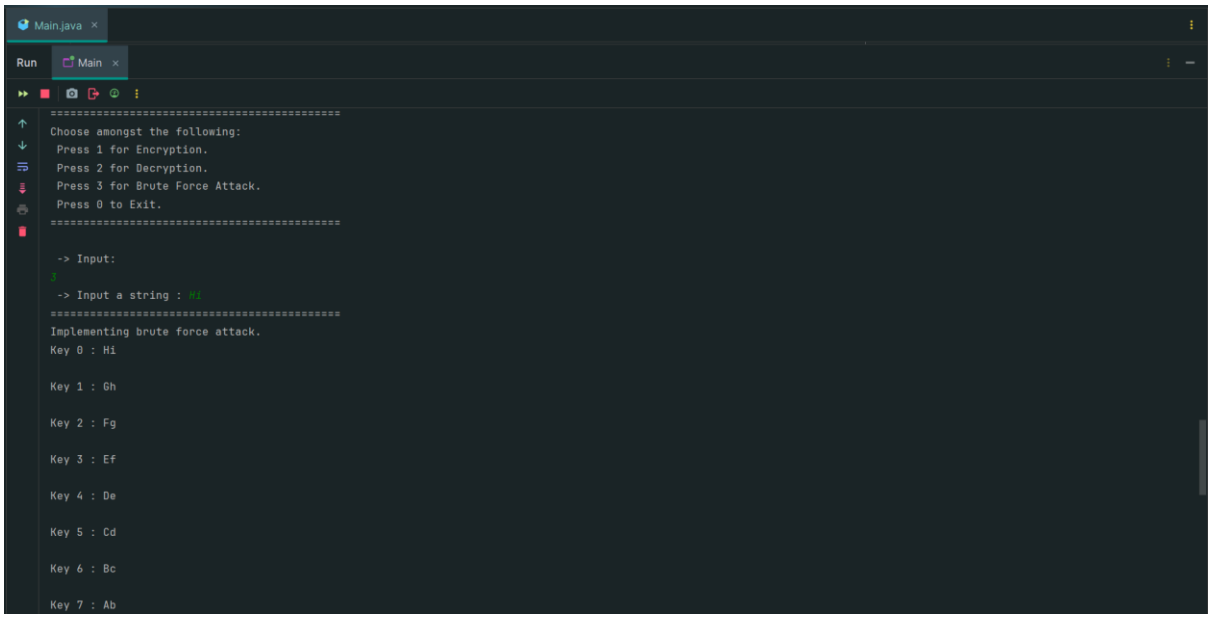


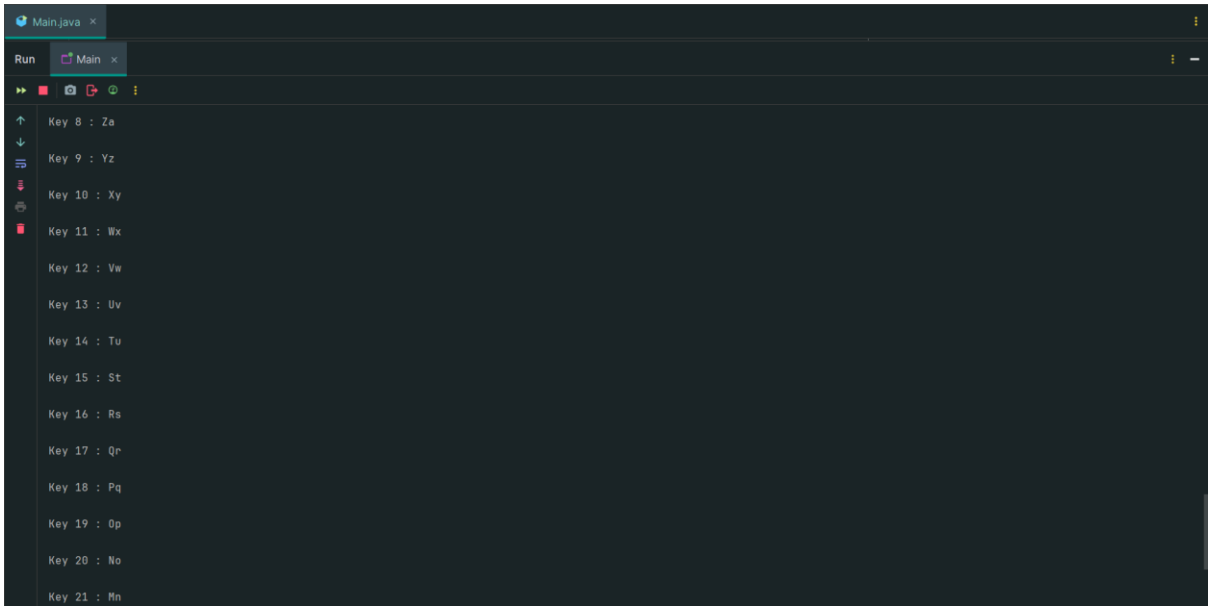
Functional Testcases:

1. Encryption

2. Decryption

3. Brute Force




Main.java
Run
Main
Key 8 : Za
Key 9 : Yz
Key 10 : Xy
Key 11 : Wx
Key 12 : Vw
Key 13 : Uv
Key 14 : Tu
Key 15 : St
Key 16 : Rs
Key 17 : Qr
Key 18 : Pq
Key 19 : Op
Key 20 : No
Key 21 : Mn


Main.java
Run
Main
Key 18 : Pq
Key 19 : Op
Key 20 : No
Key 21 : Mn
Key 22 : Lm
Key 23 : Kl
Key 24 : Jk
Key 25 : Ij
==========
==========
Choose amongst the following:
Press 1 for Encryption.
Press 2 for Decryption.
Press 3 for Brute Force Attack.
Press 0 to Exit.
==========
-> Input:

CODE

```
import java.util.Scanner;

public class Main {

    // Encryption function applying Substitution Cipher
    public static String encryptionFunction(String input, int shift) {
        // Initialize a StringBuilder to store the encrypted output
        StringBuilder encryptionOutput = new StringBuilder();
        // Loop through each character of the input string
        for(int i = 0; i < input.length(); i++) {
            // Get the current character at index 'i'
            char ch = input.charAt(i);

            // Check if the character is an uppercase letter
            if (Character.isUpperCase(ch)) {
                // Apply the substitution cipher shift to uppercase characters
                // The algorithm is first removing the ASCII value of 'A' from the character, then the shift is applied, after which the result is modded by 26 to ensure it wraps around within the alphabet. After that, the ASCII value of 'A' is added back to get the cipher text of the character.
                char currentEncryptedChar = (char) (((ch - 'A' + shift) % 26) + 'A');
                // Append the encrypted character to the result
                encryptionOutput.append(currentEncryptedChar);
            } else if (Character.isLowerCase(ch)) { // Check if the character is a lowercase letter
                // Apply the substitution cipher shift to lowercase characters
                // The algorithm is first removing the ASCII value of 'a' from the character, then the shift is applied, after which the result is modded by 26 to ensure it wraps around within the alphabet. After that, the ASCII value of 'a' is added back to get the cipher text of the character.
                char currentEncryptedChar = (char) (((ch - 'a' + shift) % 26) + 'a');
                // Append the encrypted character to the result
                encryptionOutput.append(currentEncryptedChar);
            } else { // If the character is neither uppercase nor lowercase (like spaces, punctuation, etc.)
                // Leave the character unchanged and append it to the result
                encryptionOutput.append(ch);
            }
        }
        // Convert the StringBuilder to a String and return the final encrypted result
        return encryptionOutput.toString();
    }

    // Decryption function applying Substitution Cipher
    public static String decryptionFunction(String input, int shift) {
        // Initialize a StringBuilder to store the decrypted output
        StringBuilder decryptionOutput = new StringBuilder();
        // Loop through each character of the input string
        for (int i = 0; i < input.length(); i++) {
            // Get the current character at index 'i'
            char ch = input.charAt(i);
```

```
            // Check if the character is an uppercase letter
            if (Character.isUpperCase(ch)) {
                // Decrypt the substitution cipher shift for uppercase letters
                // The algorithm is first removing the ASCII value of 'A' from the character, then the shift is applied, after which the result is modded by 26 to ensure it wraps around within the alphabet. After that, the ASCII value of 'A' is added back to get the cipher text of the character.
                // 'A' is added back to convert it to the corresponding ASCII value of the decrypted character
                char currentEncryptedChar = (char) (((ch - 'A' - shift + 26) % 26) + 'A');
                // Append the decrypted character to the result
                decryptionOutput.append(currentEncryptedChar);
            } else if (Character.isLowerCase(ch)) { // Check if the character is a lowercase letter
                // Decrypt the substitution cipher shift for lowercase letters
                // The algorithm is first removing the ASCII value of 'a' from the character, then the shift is applied, after which the result is modded by 26 to ensure it wraps around within the alphabet. After that, the ASCII value of 'a' is added back to get the cipher text of the character.
                // 'a' is added back to convert it to the corresponding ASCII value of the decrypted character
                char currentEncryptedChar = (char) (((ch - 'a' - shift + 26) % 26) + 'a');
                // Append the decrypted character to the result
                decryptionOutput.append(currentEncryptedChar);
            } else { // If the character is neither uppercase nor lowercase (e.g., spaces, punctuation, etc.)
                // Leave the character unchanged and append it to the result
                decryptionOutput.append(ch);
            }
        }
        // Convert the StringBuilder to a String and return the final decrypted result
        return decryptionOutput.toString();
    }

    // Function for brute-force decryption attack
    public static void bruteForceFunction(String input) {
        // Stimulate the brute force attack by running a loop for 26 values of key(all possible values(after 26, the output starts to repeat)) and decrypting the ciphertext value inputted.
        for (int i = 0; i < 26; i++) {
            System.out.println("Key " + i + " : " + decryptionFunction(input, i) + "\n");
        }
    }

    // Function to check if a key is a valid integer
    public static boolean isInteger(String str) {
        try {
            Integer.parseInt(str);
            return true;
        } catch (NumberFormatException e) {
            return false;
        }
    }

    // Function to fetch the user's operation choice
    public static void fetchChoice(StringBuilder num) {
        Scanner scanner = new Scanner(System.in);
        // Menu Display
```

```
        System.out.println("==========================================");
        System.out.println("Choose amongst the following:\n " +
                "Press 1 for Encryption.\n " +
                "Press 2 for Decryption. \n " +
                "Press 3 for Brute Force Attack.\n " +
                "Press 0 to Exit.");
        System.out.println("==========================================\n");
        num.setLength(0);
        // Take user input for choice of operation
        System.out.println(" -> Input: ");
        num.append(scanner.nextLine());

        // Validate the input for choice
        if (!num.toString().matches("[0-3]")) {
            // Take input again if the entered value is invalid, continue otherwise
            System.out.println(" ==> Invalid choice! Please enter a valid option that lies within the range
0-3.");
        }
    }

    // Fetch input for encryption/decryption (string and key)
    public static void fetchInputForEncryptionDecryption(StringBuilder stringInput, StringBuilder
shift) {
        Scanner scanner = new Scanner(System.in);

        // Take the string input
        System.out.print(" -> Input a string : ");
        stringInput.setLength(0); // Reset the StringBuilder
        stringInput.append(scanner.nextLine());

        if (!stringInput.toString().matches("[a-zA-Z ]+")) {
            // Take input again if the entered value is invalid, continue otherwise
            System.out.println(" ==> This string value is not acceptable! Try Again.");
            return;
        }

        // Take the key input
        System.out.print(" -> Input an integer shift by: ");
        shift.setLength(0);
        shift.append(scanner.nextLine());

        // Take input again if the entered value is invalid, continue otherwise
        if (!isInteger(shift.toString())) {
            System.out.println(" ==> This key value is not acceptable! Try Again.");
        }
    }

    // Fetch input for brute-force (only string)
    public static void fetchInputForBruteForce(StringBuilder stringInput) {
        Scanner scanner = new Scanner(System.in);
```

```
        // Take the string input
        System.out.print(" -> Input a string : ");
        stringInput.setLength(0);
        stringInput.append(scanner.nextLine());

        // Take input again if the entered value is invalid, continue otherwise
        if (!stringInput.toString().matches("[a-zA-Z ]+")) {
            System.out.println(" ==> This string value is not acceptable! Try Again.");
        }
    }

    public static void main(String[] args) {
        StringBuilder stringInput = new StringBuilder();
        StringBuilder shift = new StringBuilder();
        StringBuilder num = new StringBuilder();

        while (true) {
            // Opens Menu for the user
            fetchChoice(num);

            // If user enters 0, exit the program
            if (num.toString().equals("0")) {
                System.out.println(" ==> Exiting the program. Goodbye!");
                break;
            }

            // Value of user's input for the choice
            int numChoice;
            try {
                numChoice = Integer.parseInt(num.toString());
            } catch (NumberFormatException e) {
                continue;
            }

            String result;

            switch (numChoice) {
                case 1:
                    // 1 for Encryption
                    fetchInputForEncryptionDecryption(stringInput, shift);
                    // Ensure inputs are valid before proceeding
                    if (stringInput.isEmpty() || shift.isEmpty() || !isInteger(shift.toString()) ||
!stringInput.toString().matches("[a-zA-Z ]+")) {
                        System.out.println(" ==> Invalid input. Please try again.");
                    } else {
                        result = encryptionFunction(stringInput.toString(), Integer.parseInt(shift.toString()));
                        System.out.println("============================================");
                        System.out.println("The Encrypted Output is : " + result);
                        System.out.println("============================================");
                    }
                    break;
```

```
            case 2:
                // 2 for Decryption
                fetchInputForEncryptionDecryption(stringInput, shift);
                // Ensure inputs are valid before proceeding
                if (stringInput.isEmpty() || shift.isEmpty() || !isInteger(shift.toString()) ||
!stringInput.toString().matches("[a-zA-Z ]+")) {
                    System.out.println(" ==> Invalid input. Please try again.");
                } else {
                    result = decryptionFunction(stringInput.toString(), Integer.parseInt(shift.toString()));
                    System.out.println("============================================");
                    System.out.println("The Decrypted Output is : " + result);
                    System.out.println("============================================");
                }
                break;

            case 3:
                // 3 for Brute Force Attack
                fetchInputForBruteForce(stringInput);
                // Ensure inputs are valid before proceeding
                if (stringInput.isEmpty() || !stringInput.toString().matches("[a-zA-Z ]+")) {
                    System.out.println(" ==> Invalid input. Please try again.");
                } else {
                    System.out.println("============================================");
                    System.out.println("Implementing brute force attack.");
                    bruteForceFunction(stringInput.toString());
                    System.out.println("============================================");
                }
                break;

            default:
                System.out.println(" ==> Something went wrong. Re-initializing...");
                break;
        }
    }
  }
}
```

**** END OF DOCUMENT ****