Transposition Techniques

1)Rail Fence Cipher-:

- In the rail fence cipher, the plain-text is written downwards and diagonally on successive rails of an imaginary fence. When we reach the bottom rail, we traverse upwards moving diagonally, after reaching the top rail, the direction is changed again. Thus the alphabets of the message are written in a zig-zag manner.

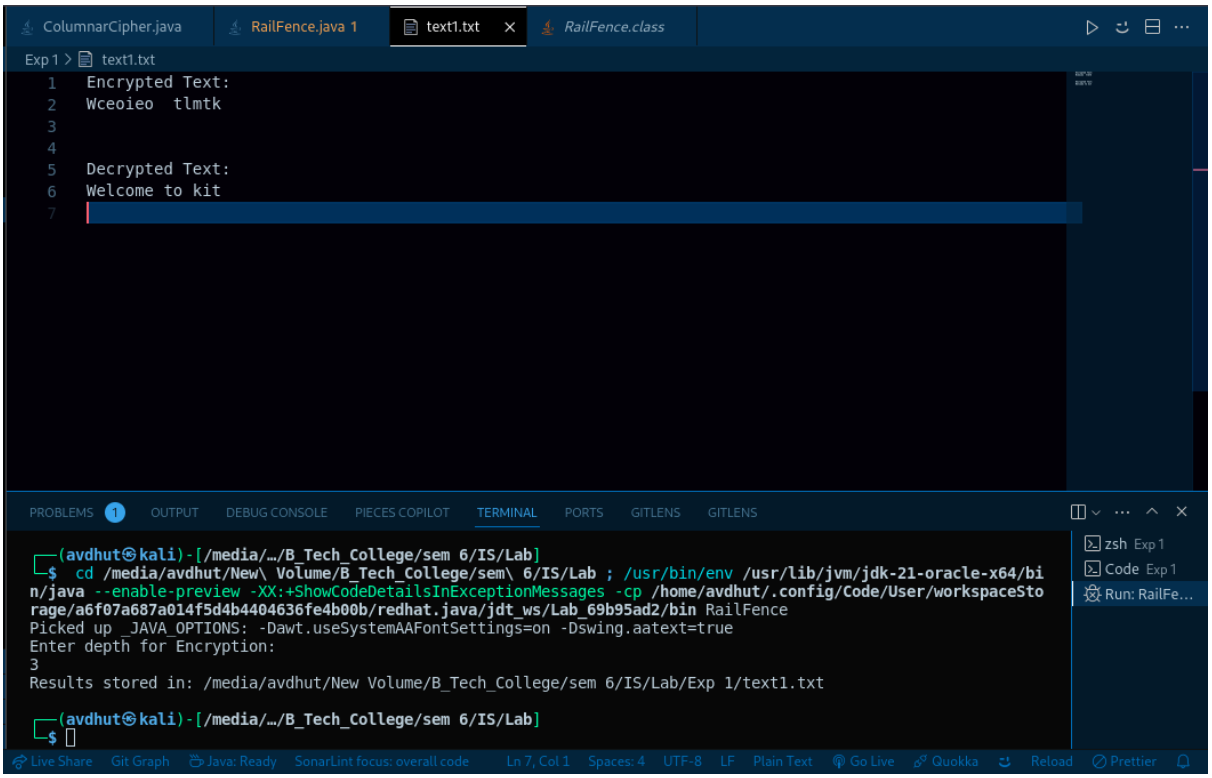
Input:-

```
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

class RailFenceCipher {
   int depth;

   String encryption(String plainText, int depth){
       int r = depth;
       int len = plainText.length();
       int c = len / depth;
       char[][] mat = new char[r][c];
       int k = 0;

       StringBuilder cipherText = new StringBuilder();
       for (int i = 0; i < c; i++) {
           for (int j = 0; j < r; j++) {
               if (k != len)
                   mat[j][i] = plainText.charAt(k++);
               else
                   mat[j][i] = 'X';
           }
       }
       for (int i = 0; i < r; i++) {
           for (int j = 0; j < c; j++) {
               cipherText.append(mat[i][j]);
           }
       }
       return cipherText.toString();
```

```
    }

    String decryption(String cipherText, int depth) {
        int r = depth;
        int len = cipherText.length();
        int c = len / depth;
        char[][] mat = new char[r][c];
        int k = 0;

        StringBuilder plainText = new StringBuilder();

        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c; j++) {
                mat[i][j] = cipherText.charAt(k++);
            }
        }
        for (int i = 0; i < c; i++) {
            for (int j = 0; j < r; j++) {
                plainText.append(mat[j][i]);
            }
        }

        return plainText.toString();
    }
}

class RailFence {

    public static void main(String[] args) throws IOException{
        RailFenceCipher rf = new RailFenceCipher();
        int depth;

        String plainText;
        String cipherText;
        String decryptedText;
        String filePath;
        // Hardcoded file path
        filePath = System.getProperty("user.dir") + "/Exp 1/text1.txt";

        plainText = readTextFromFile(filePath);

        System.out.println("Enter depth for Encryption:");
        try (Scanner scn = new Scanner(System.in)) {
```

```
            depth = scn.nextInt();
        }

        cipherText = rf.encryption(plainText, depth);

        decryptedText = rf.decryption(cipherText, depth);
        // Write results back to the same file
        writeTextToFile(filePath, "Encrypted Text:\n" + cipherText +
                "\n\nDecrypted Text:\n" + decryptedText);
        System.out.println("Results stored in: " + filePath);
    }

    private static String readTextFromFile(String filePath) throws
FileNotFoundException {
        StringBuilder content = new StringBuilder();
        try (Scanner scanner = new Scanner(new FileReader(filePath))) {
            while (scanner.hasNextLine()) {
                content.append(scanner.nextLine()).append("\n");
            }
        }
        return content.toString();
    }

    private static void writeTextToFile(String filePath, String content)
throws IOException {
        try (BufferedWriter bw = new BufferedWriter(new
FileWriter(filePath))) {
            bw.write(content);
        }
    }
}
```

Output-:



2)Columnar Cipher-:

- <u>Encryption</u>
  We first pick a keyword for our encryption. We write the plaintext out in a grid where the number of columns is the number of letters in the keyword. We then title each column with the respective letter from the keyword. We take the letters in the keyword in alphabetical order, and read down the columns in this order. If a letter is repeated, we do the one that appears first, then the next and so on.

```
import java.io.*; import
java.util.*;
 public class ColumnarTransposition {
// Key for Columnar Transposition
static final String key = "4312567";
    static Map<Character, Integer> keyMap = new HashMap<>();

    static void setPermutationOrder() {
        // Add the permutation order into the map
for (int i = 0; i < key.length(); i++) {
keyMap.put(key.charAt(i), i);
        }
    }


     // Encryption
    static String encryptMessage(String msg) {
int row, col;
        StringBuilder cipher = new StringBuilder();

        col = key.length();

        row = (int) Math.ceil((double) msg.length() / col);

        char[][] matrix = new char[row][col];

        for (int i = 0, k = 0; i < row; i++) {
for (int j = 0; j < col; ) {
if (k < msg.length()) {
char ch = msg.charAt(k);
                    if (Character.isLetter(ch) || ch == ' ') {
matrix[i][j] = ch;                          j++;
}                     k++;                  } else {
                    matrix[i][j] = ' ';
j++;
                }
            }
        }
          for (Map.Entry<Character, Integer> entry : keyMap.entrySet())
{             int columnIndex = entry.getValue();

            for (int i = 0; i < row; i++) {
```

```
                if (Character.isLetter(matrix[i][columnIndex]) ||
matrix[i][columnIndex] == ' ' || matrix[i][columnIndex] == '_') {
cipher.append(matrix[i][columnIndex]);
                }
            }
        }

        return cipher.toString();
    }

    // Decryption
    static String decryptMessage(String cipher) {
int col = key.length();

        int row = (int) Math.ceil((double) cipher.length() / col);
char[][] cipherMat = new char[row][col];         int k = 0;
        for (int j = 0; j < col; j++) {
for (int i = 0; i < row; i++) {
cipherMat[i][j] = cipher.charAt(k);
k++;
            }
        }

        int index = 0;
        for (Map.Entry<Character, Integer> entry : keyMap.entrySet()) {
entry.setValue(index++);
        }
         char[][] decCipher = new char[row][col];
for (int l = 0; l < key.length(); l++) {             int
columnIndex = keyMap.get(key.charAt(l));             for
(int i = 0; i < row; i++) {
                decCipher[i][l] = cipherMat[i][columnIndex];
            }
        }

        StringBuilder msg = new StringBuilder();
for (int i = 0; i < row; i++) {             for
(int j = 0; j < col; j++) {                 if
(decCipher[i][j] != '_') {
msg.append(decCipher[i][j]);
                }
            }
        }

        return msg.toString();
    }       public static void main(String[]
args) {         try {
            /* Read plaintext from a .txt file */
            String filePath = "D:\\IS\\text.txt";
            String msg = readTextFromFile(filePath);

            setPermutationOrder();
            String cipher = encryptMessage(msg);
            String decryptedMsg = decryptMessage(cipher);

            writeTextToFile(filePath, "Plaintext:\n" + msg + "\n\nEncrypted
```

```
Message:\n" + cipher + "\n\nDecrypted Message:\n" + decryptedMsg);
System.out.println("Results stored in: " + filePath);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
     private static String readTextFromFile(String filePath) throws
Exception {
        StringBuilder content = new StringBuilder();
        try (Scanner scanner = new Scanner(new FileReader(filePath))) {
while (scanner.hasNextLine()) {
                content.append(scanner.nextLine()).append("\n");
            }
}
        return content.toString();
    }
    private static void writeTextToFile(String filePath, String content)
throws Exception {
        try (BufferedWriter bw = new BufferedWriter(new
FileWriter(filePath))) {             bw.write(content);
        }
    }
}
```

output-: