**AMEYA SANJAY BHAGAT**

**A05/S.20.05**

**INFORMATION AND NETWORK SECURITY PRACTICALS**

**Practical 1**: *Write programs to implement the following Substitution Cipher Techniques*

1. Caesar Cipher

**Code**:

import java.util.Scanner;

public class CaesarCipher

{

public static final String ALPHABET = "abcdefghijklmnopqrstuvwxyz";

public static String encrypt(String plainText, int shiftKey)

{

plainText = plainText.toLowerCase();

String cipherText = "";

for (int i = 0; i < plainText.length(); i++)

{

int charPosition = ALPHABET.indexOf(plainText.charAt(i));

int keyVal = (shiftKey + charPosition) % 26;

char replaceVal = ALPHABET.charAt(keyVal);

cipherText += replaceVal;

}

return cipherText;

}

public static String decrypt(String cipherText, int shiftKey)

{

cipherText = cipherText.toLowerCase();

String plainText = "";

for (int i = 0; i < cipherText.length(); i++)

{

int charPosition = ALPHABET.indexOf(cipherText.charAt(i));

int keyVal = (charPosition - shiftKey) % 26;

if (keyVal < 0)

{

keyVal = ALPHABET.length() + keyVal;

}

char replaceVal = ALPHABET.charAt(keyVal);

plainText += replaceVal;

}

return plainText;

}

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter the String for Encryption: ");

String message = new String();

message = sc.next();

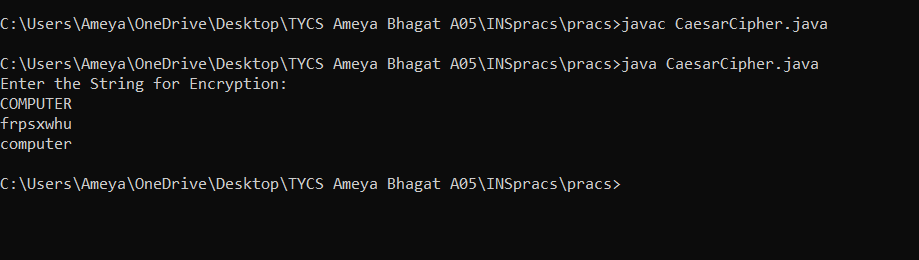
System.out.println(encrypt(message, 3));

System.out.println(decrypt(encrypt(message, 3), 3));

sc.close();

}

**Output** :



1. Monoalphabetic Cipher

**Code**:

import java.util.Scanner;

public class MonoalphabeticCipher

{

public static char p[] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',

'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',

'w', 'x', 'y', 'z' };

public static char ch[] = { 'Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O',

'P', 'A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', 'Z', 'X', 'C',

'V', 'B', 'N', 'M' };

public static String doEncryption(String s)

{

char c[] = new char[(s.length())];

for (int i = 0; i < s.length(); i++)

{

for (int j = 0; j < 26; j++)

{

if (p[j] == s.charAt(i))

{

c[i] = ch[j];

break;

}

}

}

return (new String(c));

}

public static String doDecryption(String s)

{

char p1[] = new char[(s.length())];

for (int i = 0; i < s.length(); i++)

{

for (int j = 0; j < 26; j++)

{

if (ch[j] == s.charAt(i))

{

p1[i] = p[j];

break;

}

}

}

return (new String(p1));

}

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter the message: ");

String en = doEncryption(sc.next().toLowerCase());

System.out.println("Encrypted message: " + en);

System.out.println("Decrypted message: " + doDecryption(en));

sc.close();

}

}

**Output** :

Text

Description automatically generated

**Practical 2**: *Write programs to implement the following Substitution Cipher Techniques*

1. Vernam Cipher

**Code**:

import java.io.\*;

public class insprac2 {

public static String stringEncryption(String text,

String key)

{

String cipherText = "";

int cipher[] = new int[key.length()];

for (int i = 0; i < key.length(); i++) {

cipher[i] = text.charAt(i) - 'A'

+ key.charAt(i)

- 'A';

}

for (int i = 0; i < key.length(); i++) {

if (cipher[i] > 25) {

cipher[i] = cipher[i] - 26;

}

}

for (int i = 0; i < key.length(); i++) {

int x = cipher[i] + 'A';

cipherText += (char)x;

}

return cipherText;

}

public static String stringDecryption(String s,

String key)

{

String plainText = "";

int plain[] = new int[key.length()];

for (int i = 0; i < key.length(); i++) {

plain[i]

= s.charAt(i) - 'A'

- (key.charAt(i) - 'A');

}

for (int i = 0; i < key.length(); i++) {

if (plain[i] < 0) {

plain[i] = plain[i] + 26;

}

}

for (int i = 0; i < key.length(); i++) {

int x = plain[i] + 'A';

plainText += (char)x;

}

return plainText;

}

public static void main(String[] args)

{

String plainText = "Hello";

String key = "MONEY";

String encryptedText = stringEncryption(

plainText.toUpperCase(), key.toUpperCase());

System.out.println("Cipher Text - "

+ encryptedText);

System.out.println(

"Message - "

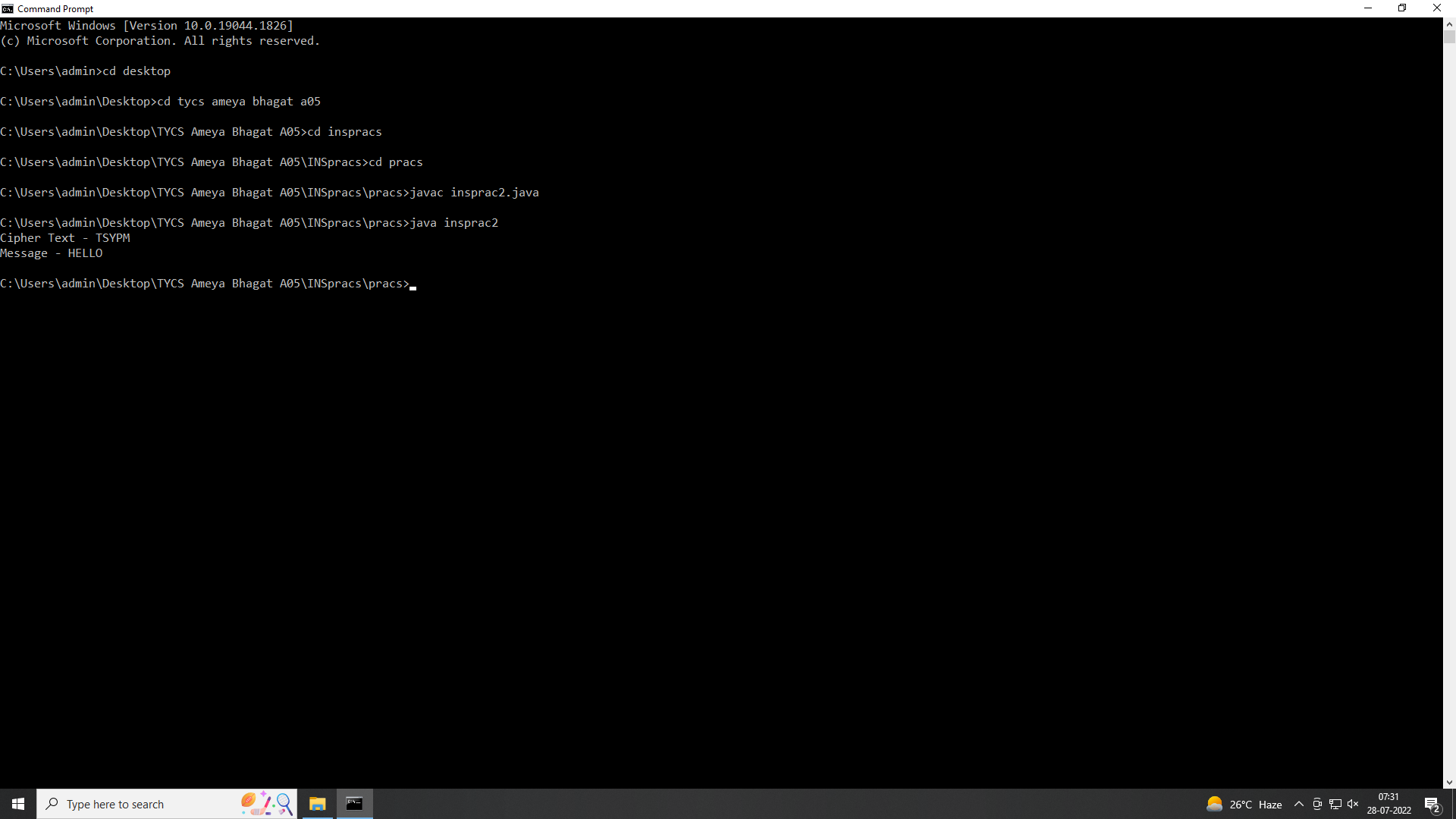
+ stringDecryption(encryptedText,

key.toUpperCase()));

}

}

**Output** :



1. Playfair Cipher

**Code**:

import java.io.\*;

import java.util.\*;

class Playfair {

String key;

String plainText;

char[][] matrix = new char[5][5];

public Playfair(String key, String plainText)

{

this.key = key.toLowerCase();

this.plainText = plainText.toLowerCase();

}

public void cleanPlayFairKey()

{

LinkedHashSet<Character> set

= new LinkedHashSet<Character>();

String newKey = "";

for (int i = 0; i < key.length(); i++)

set.add(key.charAt(i));

Iterator<Character> it = set.iterator();

while (it.hasNext())

newKey += (Character)it.next();

key = newKey;

}

public void generateCipherKey()

{

Set<Character> set = new HashSet<Character>();

for (int i = 0; i < key.length(); i++)

{

if (key.charAt(i) == 'j')

continue;

set.add(key.charAt(i));

}

String tempKey = new String(key);

for (int i = 0; i < 26; i++)

{

char ch = (char)(i + 97);

if (ch == 'j')

continue;

if (!set.contains(ch))

tempKey += ch;

}

for (int i = 0, idx = 0; i < 5; i++)

for (int j = 0; j < 5; j++)

matrix[i][j] = tempKey.charAt(idx++);

System.out.println("Playfair Cipher Key Matrix:");

for (int i = 0; i < 5; i++)

System.out.println(Arrays.toString(matrix[i]));

}

public String formatPlainText()

{

String message = "";

int len = plainText.length();

for (int i = 0; i < len; i++)

{

if (plainText.charAt(i) == 'j')

message += 'i';

else

message += plainText.charAt(i);

}

for (int i = 0; i < message.length(); i += 2)

{

if (message.charAt(i) == message.charAt(i + 1))

message = message.substring(0, i + 1) + 'x'

+ message.substring(i + 1);

}

if (len % 2 == 1)

message += 'x'; // dummy character

return message;

}

public String[] formPairs(String message)

{

int len = message.length();

String[] pairs = new String[len / 2];

for (int i = 0, cnt = 0; i < len / 2; i++)

pairs[i] = message.substring(cnt, cnt += 2);

return pairs;

}

public int[] getCharPos(char ch)

{

int[] keyPos = new int[2];

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

if (matrix[i][j] == ch)

{

keyPos[0] = i;

keyPos[1] = j;

break;

}

}

}

return keyPos;

}

public String encryptMessage()

{

String message = formatPlainText();

String[] msgPairs = formPairs(message);

String encText = "";

for (int i = 0; i < msgPairs.length; i++)

{

char ch1 = msgPairs[i].charAt(0);

char ch2 = msgPairs[i].charAt(1);

int[] ch1Pos = getCharPos(ch1);

int[] ch2Pos = getCharPos(ch2);

if (ch1Pos[0] == ch2Pos[0]) {

ch1Pos[1] = (ch1Pos[1] + 1) % 5;

ch2Pos[1] = (ch2Pos[1] + 1) % 5;

}

else if (ch1Pos[1] == ch2Pos[1])

{

ch1Pos[0] = (ch1Pos[0] + 1) % 5;

ch2Pos[0] = (ch2Pos[0] + 1) % 5;

}

else {

int temp = ch1Pos[1];

ch1Pos[1] = ch2Pos[1];

ch2Pos[1] = temp;

}

encText = encText + matrix[ch1Pos[0]][ch1Pos[1]]

+ matrix[ch2Pos[0]][ch2Pos[1]];

}

return encText;

}

}

public class Playfair1 {

public static void main(String[] args)

{

System.out.println("Example-1\n");

String key1 = "Problem";

String plainText1 = "Playfair";

System.out.println("Key: " + key1);

System.out.println("PlainText: " + plainText1);

Playfair pfc1 = new Playfair(key1, plainText1);

pfc1.cleanPlayFairKey();

pfc1.generateCipherKey();

String encText1 = pfc1.encryptMessage();

System.out.println("Cipher Text is: " + encText1);

System.out.println("\nExample-2\n");

String key2 = "Problem";

String plainText2 = "Hello";

System.out.println("Key: " + key2);

System.out.println("PlainText: " + plainText2);

Playfair pfc2 = new Playfair(key2, plainText2);

pfc2.cleanPlayFairKey();

pfc2.generateCipherKey();

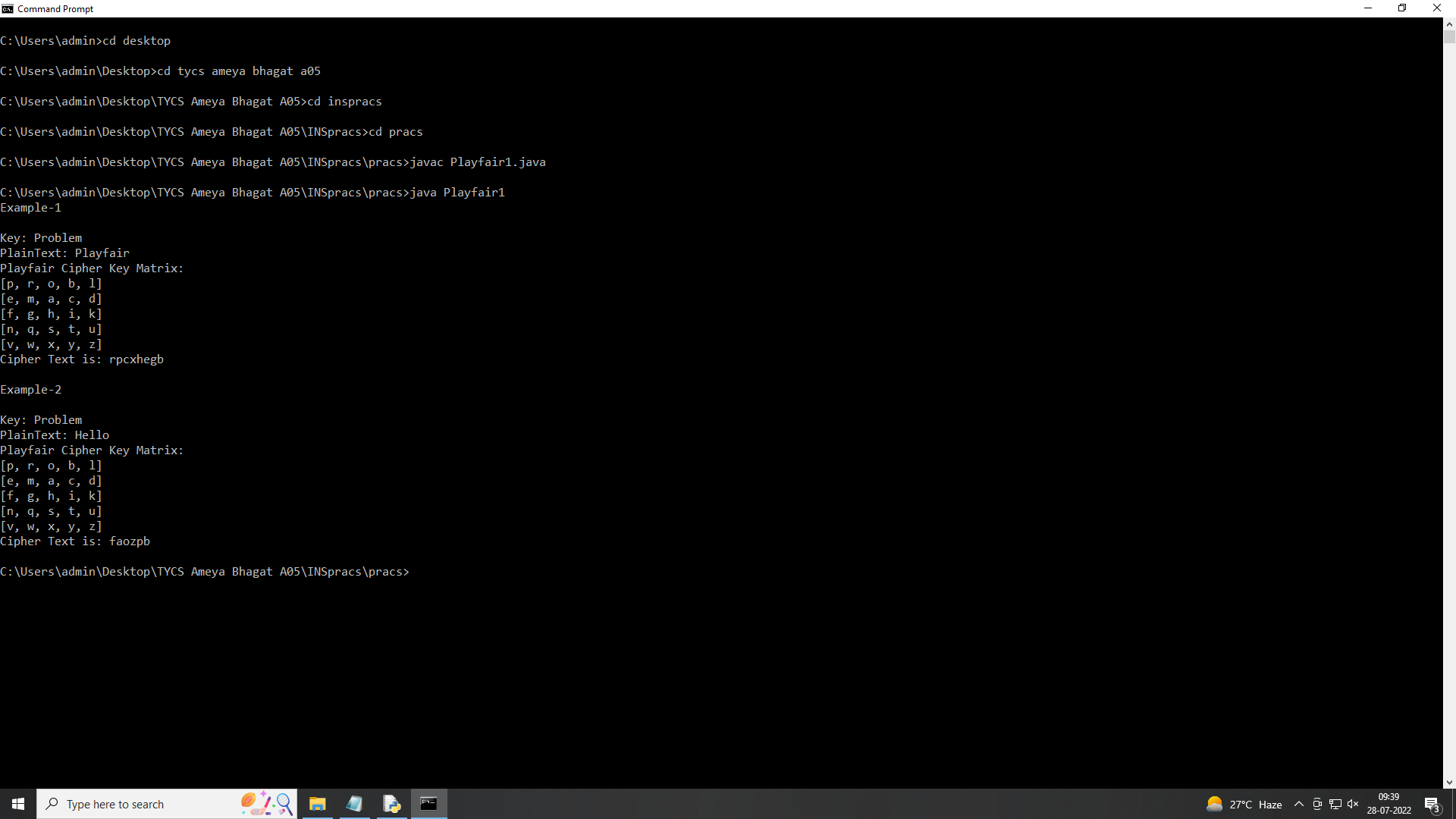
String encText2 = pfc2.encryptMessage();

System.out.println("Cipher Text is: " + encText2);

}

}

**Output** :



**Practical 3**: *Write programs to implement the following Transposition Cipher Techniques*

1. Simple Columnar Technique

**Code**:

import java.util.Scanner;

public class SCC {

private static Scanner in;

public static void main(String[] args){

System.out.println("Columnar Transposition Cipher");

in = new Scanner(System.in);

System.out.print("1. Encryption\n2. Decryption\nChoose(1,2): ");

int choice = in.nextInt();

in.nextLine();

if (choice == 1){

System.out.println("Encryption");

encryption();

} else if (choice == 2){

System.out.println("Decryption");

decryption();

} else {

System.out.println("Invalid Choice");

System.exit(0);

}

} // main method

private static void encryption(){

System.out.print("Enter Message: ");

String plainText = in.nextLine().toUpperCase().replace(" ", "");

StringBuilder msg = new StringBuilder(plainText);

System.out.print("Enter Keyword: ");

String keyword = in.nextLine().toUpperCase();

// assigning numbers to keywords

int[] kywrdNumList = keywordNumAssign(keyword);

// printing keyword

for (int i = 0, j = 1; i < keyword.length(); i++, j++) {

System.out.print(keyword.substring(i, j) + " ");

}

System.out.println();

for (int i: kywrdNumList){

System.out.print(i + " ");

}

System.out.println();

System.out.println("-------------------------");

// in case characters don't fit the entire grid perfectly.

int extraLetters = msg.length() % keyword.length();

//System.out.println(extraLetters);

int dummyCharacters = keyword.length() - extraLetters;

// System.out.println(dummyCharacters);

if (extraLetters != 0){

for (int i = 0; i < dummyCharacters; i++) {

msg.append(".");

}

}

int numOfRows = msg.length() / keyword.length();

// Converting message into a grid

char[][] arr = new char[numOfRows][keyword.length()];

int z = 0;

for (int i = 0; i < numOfRows; i++) {

for (int j = 0; j < keyword.length(); j++) {

arr[i][j] = msg.charAt(z);

z++;

}

}

for (int i = 0; i < numOfRows; i++) {

for (int j = 0; j < keyword.length(); j++) {

System.out.print(arr[i][j] + " ");

} // inner for loop

System.out.println();

} // for loop

// cipher text generation

StringBuilder cipherText = new StringBuilder();

System.out.println();

// getting locations of numbers

String numLoc = getNumberLocation(keyword, kywrdNumList);

System.out.println("Location of numbers: " + numLoc);

System.out.println();

for (int i = 0, k = 0; i < numOfRows; i++, k++) {

int d;

if (k == keyword.length()){

break;

} else {

d = Character.getNumericValue(numLoc.charAt(k));

}

for (int j = 0; j < numOfRows; j++) {

cipherText.append(arr[j][d]);

}

}

System.out.println("Cipher Text: " + cipherText);

} // encryption method

private static void decryption(){

System.out.print("Enter Message: ");

String msg = in.nextLine().toUpperCase().replace(" ", "");

System.out.print("Enter Keyword: ");

String keyword = in.nextLine().toUpperCase();

int numOfRows = msg.length() / keyword.length();

// array with dummy values

char[][] arr = new char[numOfRows][keyword.length()];

// assigning numbers to keywords

int[] kywrdNumList = keywordNumAssign(keyword);

String numLoc = getNumberLocation(keyword, kywrdNumList);

for (int i = 0, k = 0; i < msg.length(); i++, k++) {

int d = 0;

if (k == keyword.length()){

k = 0;

} else {

d = Character.getNumericValue(numLoc.charAt(k));

}

for (int j = 0; j < numOfRows; j++, i++) {

arr[j][d] = msg.charAt(i);

} // for loop

--i;

}

StringBuilder plainText = new StringBuilder();

for (int i = 0; i < numOfRows; i++) {

for (int j = 0; j < keyword.length(); j++) {

plainText.append(arr[i][j]);

} // inner for loop

} // for loop

System.out.println("Plain Text: " + plainText);

} // decryption function

private static String getNumberLocation(String keyword, int[]

kywrdNumList) {

String numLoc = "";

for (int i = 1; i < keyword.length() + 1; i++) {

for (int j = 0; j < keyword.length(); j++) {

if (kywrdNumList[j] == i){

numLoc += j;

}

}

}

return numLoc;

} // getting number location function

private static int[] keywordNumAssign(String keyword) {

String alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

int[] kywrdNumList = new int[keyword.length()];

int init = 0;

for (int i = 0; i < alpha.length(); i ++){

for (int j = 0; j < keyword.length(); j++) {

if (alpha.charAt(i) == keyword.charAt(j)){

init++;

kywrdNumList[j] = init;

}

} // inner for

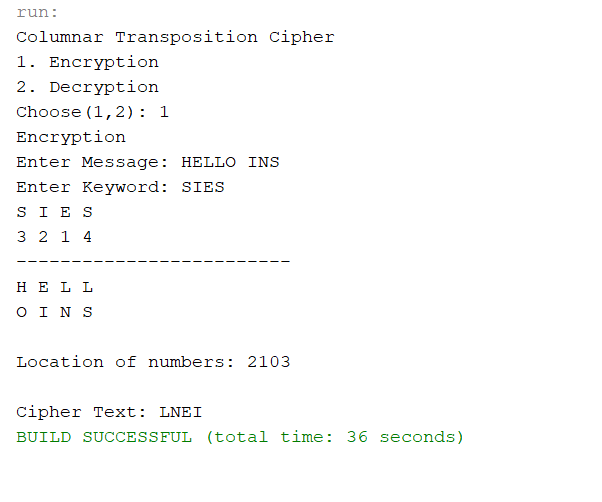
} // for

return kywrdNumList;

}

}

**Output**:



B) Rail Fence Cipher

**Code**:

import java.util.\*;

public class RailFence {

public static void main(String arg[]){

System.out.println("Enter the number of rails:");

Scanner in=new Scanner (System.in);

int rails=in.nextInt();

System.out.println("Enter the plaintext for encryption");

Scanner inn=new Scanner (System.in);

String plaintext=inn.next();

encryption(plaintext,rails);

System.out.println("------------------Decryption process start----------");

System.out.println("Enter the number of rails:");

rails=in.nextInt();

System.out.println("Enter the ciphertext for decryption:");

String ciphertext=in.next();

decryption(ciphertext,rails);

}

public static void encryption(String str,int rails){

boolean checkdown=false;

int j=0;

int row=rails;

int col=str.length();

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

for(int i=0;i<col;i++){

if(j==0||j==row-1)

checkdown=!checkdown;

a[j][i]=str.charAt(i);

if(checkdown){

j++;

}

else

j--;

}

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

for(int k=0;k<col;k++){

System.out.print(a[i][k]+" ");

}

System.out.println();

}

String en="";

System.out.println("----------------------");

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

for(int k=0;k<col;k++){

if(a[i][k]!=0)

en=en+a[i][k];

}

}

System.out.println(en);//printing the ciphertext

}

public static void decryption(String str,int rails){

boolean checkdown=false;

int j=0;

int row=rails;

int col=str.length();

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

//first of all mark the rails position by \* in the matrix

for(int i=0;i<col;i++){

if(j==0||j==row-1)

checkdown=!checkdown;

a[j][i]='\*';

if(checkdown)j++;

else j--;

}

int index=0;

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

for(int k=0;k<col;k++){

if(a[i][k]=='\*'&&index<str.length()){

a[i][k]=str.charAt(index++);

}

}

}

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

for (int k=0;k<col;k++){

System.out.print(a[i][k]+ "\t");

}

System.out.println();

}

checkdown=false;

String s="";

j=0;

for(int i=0;i<col;i++){

if( j==0||j==row-1)

checkdown=!checkdown;

s+=a[j][i];

if(checkdown)j++;

else j--;

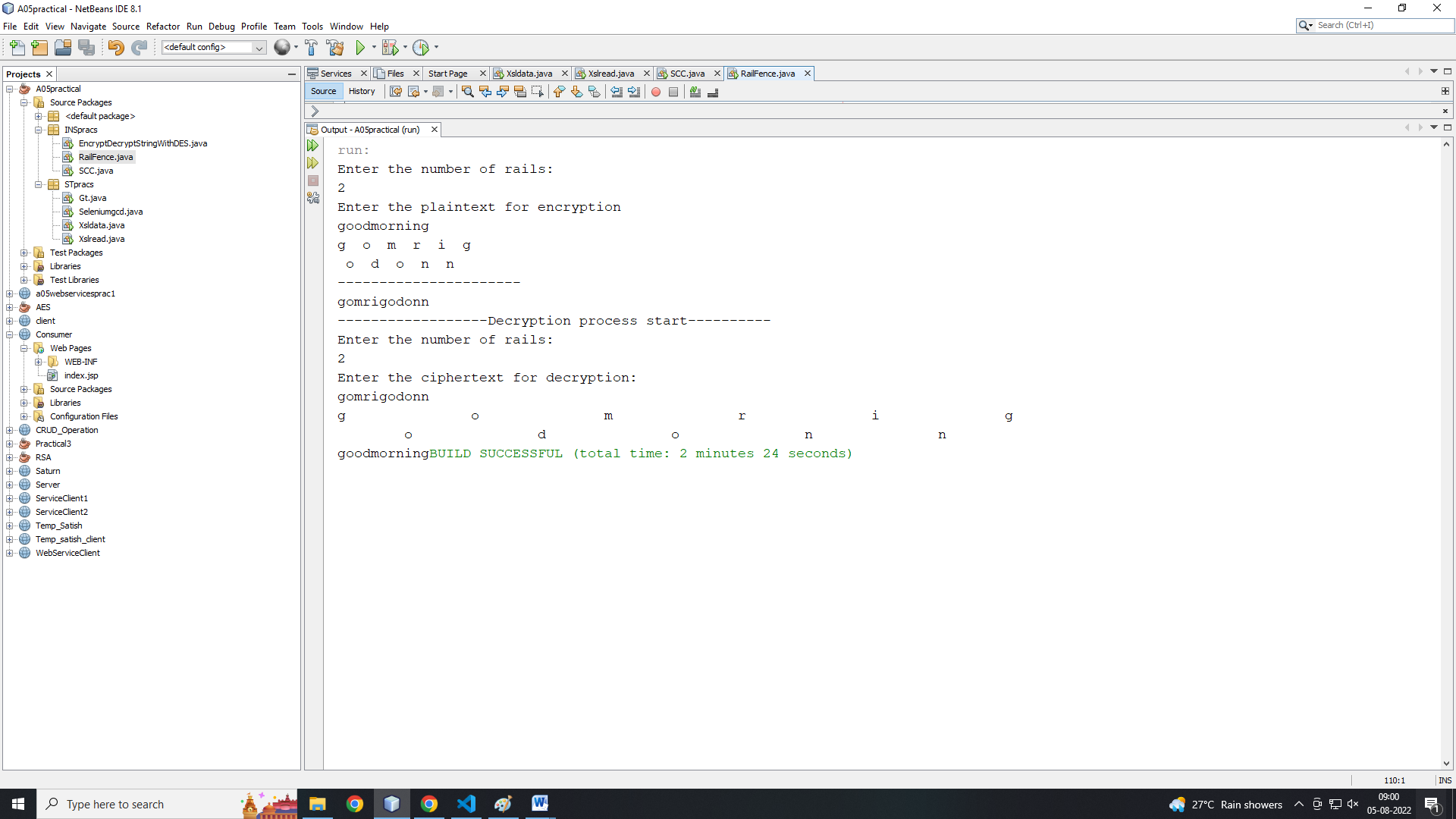
}

System.out.print(s);

}

}

**OUTPUT**:



**Practical 4**: *Write program to encrypt and decrypt strings using*

1. DES Algorithm

**Code**:

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.NoSuchPaddingException;

import javax.crypto.SecretKey;

import com.sun.mail.util.BASE64DecoderStream;

import com.sun.mail.util.BASE64EncoderStream;

public class EncryptDecryptStringWithDES {

private static Cipher ecipher;

private static Cipher dcipher;

private static SecretKey key;

public static void main(String[] args) {

try {

key = KeyGenerator.getInstance("DES").generateKey();

ecipher = Cipher.getInstance("DES");

dcipher = Cipher.getInstance("DES");

ecipher.init(Cipher.ENCRYPT\_MODE, key);

dcipher.init(Cipher.DECRYPT\_MODE, key);

String encrypted = encrypt("This is a classified message!");

String decrypted = decrypt(encrypted);

System.out.println("Decrypted: " + decrypted);

} catch (NoSuchAlgorithmException e) {

System.out.println("No Such Algorithm:" + e.getMessage());

return;

} catch (NoSuchPaddingException e) {

System.out.println("No Such Padding:" + e.getMessage());

return;

} catch (InvalidKeyException e) {

System.out.println("Invalid Key:" + e.getMessage());

return;

}

}

public static String encrypt(String str) {

try {

byte[] utf8 = str.getBytes("UTF8");

byte[] enc = ecipher.doFinal(utf8);

enc = BASE64EncoderStream.encode(enc);

return new String(enc);

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

public static String decrypt(String str) {

try {

byte[] dec = BASE64DecoderStream.decode(str.getBytes());

byte[] utf8 = dcipher.doFinal(dec);

return new String(utf8, "UTF8");

} catch (Exception e) {

e.printStackTrace();

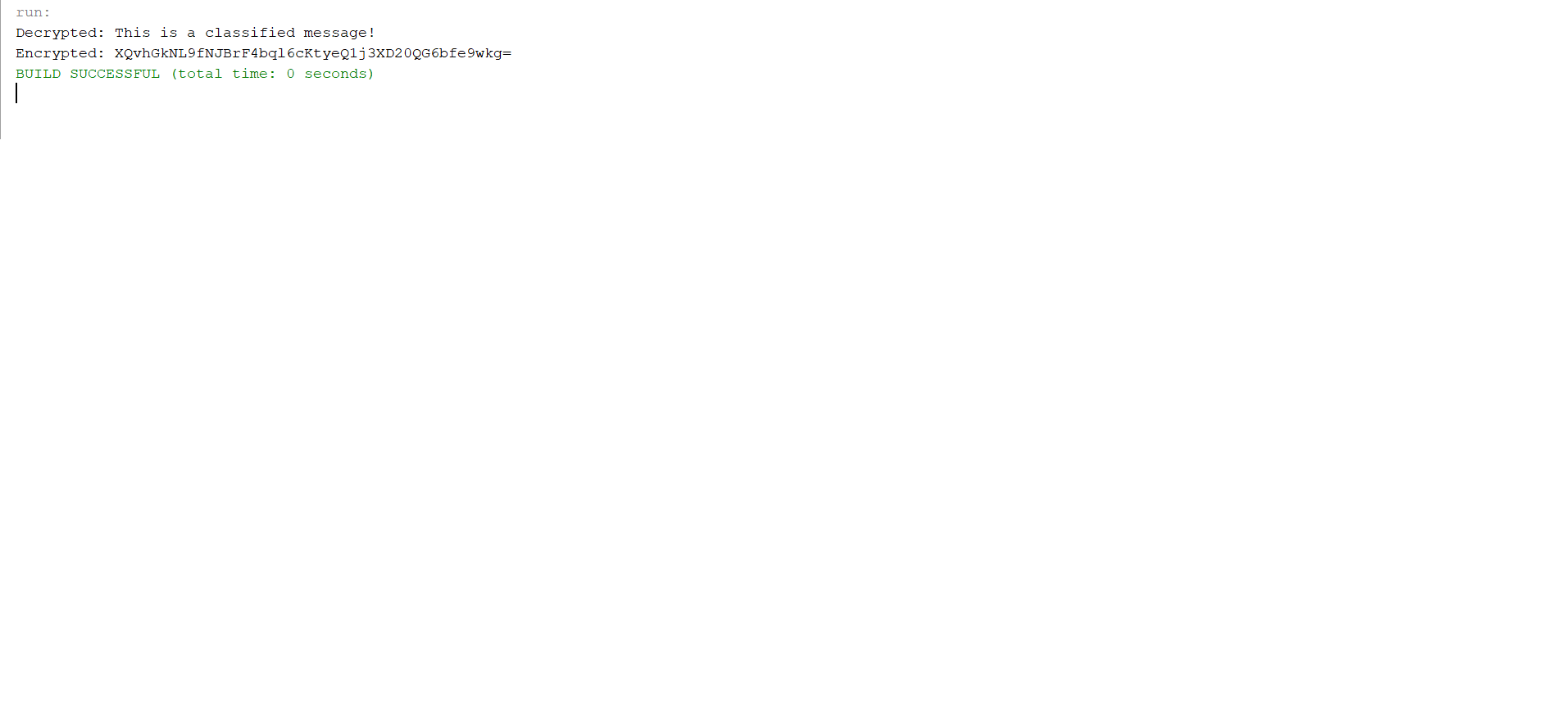
}

return null;

}

}

**Output**:



1. AES Algorithm

**Code** :

import javax.crypto.Cipher;

import javax.crypto.SecretKey;

import javax.crypto.SecretKeyFactory;

import javax.crypto.spec.IvParameterSpec;

import javax.crypto.spec.PBEKeySpec;

import javax.crypto.spec.SecretKeySpec;

import java.nio.charset.StandardCharsets;

import java.security.InvalidAlgorithmParameterException;

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import java.security.spec.InvalidKeySpecException;

import java.security.spec.KeySpec;

import java.util.Base64;

import javax.crypto.BadPaddingException;

import javax.crypto.IllegalBlockSizeException;

import javax.crypto.NoSuchPaddingException;

public class AESExample

{

private static final String SECRET\_KEY = "123456789";

private static final String SALTVALUE = "abcdefg";

public static String encrypt(String strToEncrypt)

{

try

{

byte[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

IvParameterSpec ivspec = new IvParameterSpec(iv);

SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");

KeySpec spec = new PBEKeySpec(SECRET\_KEY.toCharArray(), SALTVALUE.getBytes(), 65536, 128);

SecretKey tmp = factory.generateSecret(spec);

SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES");

Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

cipher.init(Cipher.ENCRYPT\_MODE, secretKey, ivspec);

return Base64.getEncoder()

.encodeToString(cipher.doFinal(strToEncrypt.getBytes(StandardCharsets.UTF\_8)));

}

catch (InvalidAlgorithmParameterException | InvalidKeyException | NoSuchAlgorithmException | InvalidKeySpecException | BadPaddingException | IllegalBlockSizeException | NoSuchPaddingException e)

{

System.out.println("Error occured during encryption: " + e.toString());

}

return null;

}

public static String decrypt(String strToDecrypt)

{

try

{

byte[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

IvParameterSpec ivspec = new IvParameterSpec(iv);

SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");

KeySpec spec = new PBEKeySpec(SECRET\_KEY.toCharArray(), SALTVALUE.getBytes(), 65536, 128);

SecretKey tmp = factory.generateSecret(spec);

SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES");

Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");

cipher.init(Cipher.DECRYPT\_MODE, secretKey, ivspec);

return new String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));

}

catch (InvalidAlgorithmParameterException | InvalidKeyException | NoSuchAlgorithmException | InvalidKeySpecException | BadPaddingException | IllegalBlockSizeException | NoSuchPaddingException e)

{

System.out.println("Error occured during decryption: " + e.toString());

}

return null;

}

public static void main(String[] args)

{

String originalval = "AES Encryption";

String encryptedval = encrypt(originalval);

String decryptedval = decrypt(encryptedval);

System.out.println("Original value: " + originalval);

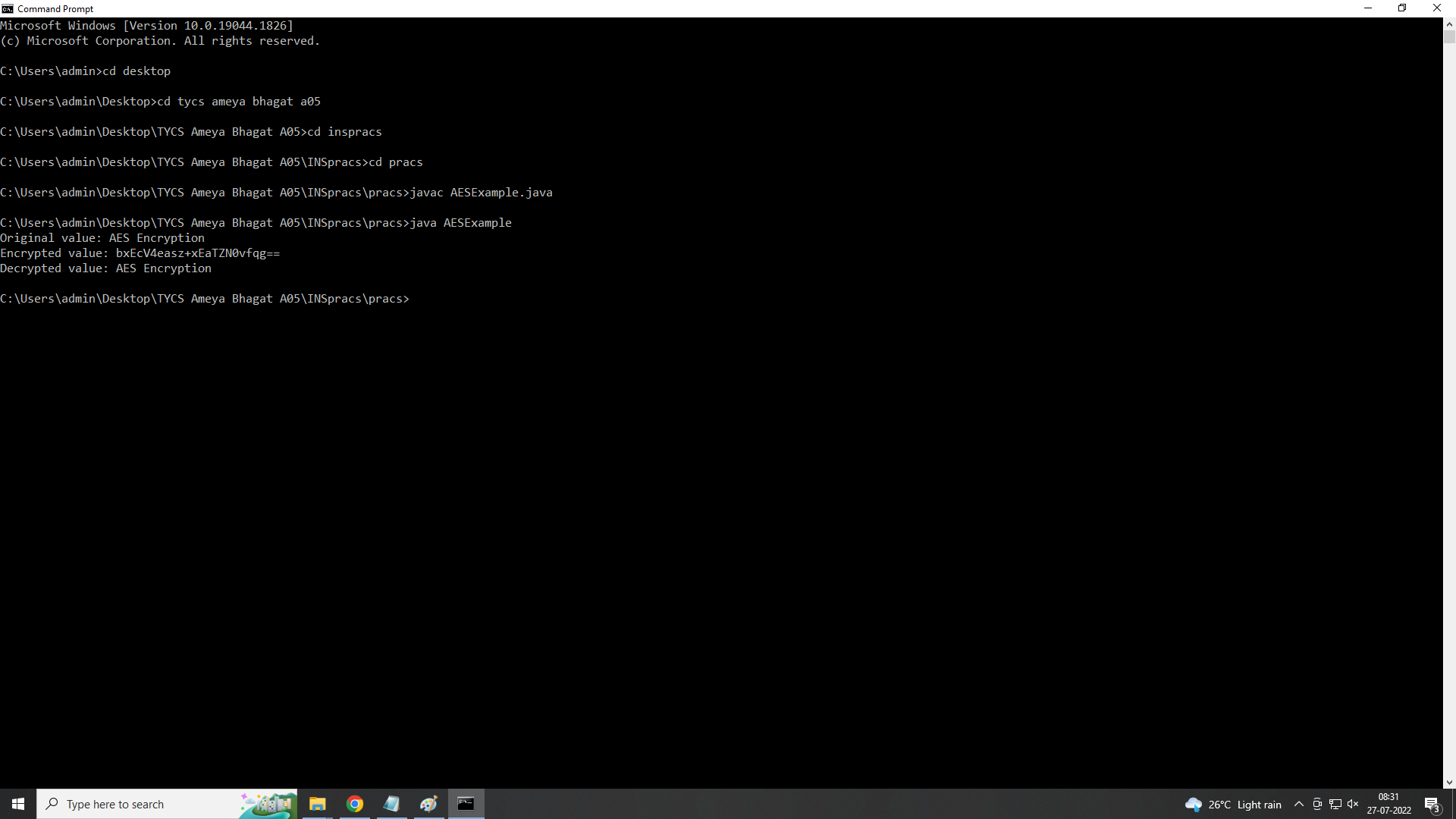
System.out.println("Encrypted value: " + encryptedval);

System.out.println("Decrypted value: " + decryptedval);

}

}

**Output**:



**Practical 5**: *Write a program to implement RSA algorithm to perform encryption / decryption of a given string*.

RSA (PART-1)

**Code**:

import javax.crypto.Cipher;

import java.security.KeyPair;

import java.security.KeyPairGenerator;

import java.security.PrivateKey;

import java.security.PublicKey;

import java.util.Base64;

public class RSA {

private PrivateKey privateKey;

private PublicKey publicKey;

public RSA() {

try {

KeyPairGenerator generator = KeyPairGenerator.getInstance("RSA");

generator.initialize(1024);

KeyPair pair = generator.generateKeyPair();

privateKey = pair.getPrivate();

publicKey = pair.getPublic();

} catch (Exception ignored) {

}

}

public String encrypt(String message) throws Exception{

byte[] messageToBytes = message.getBytes();

Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");

cipher.init(Cipher.ENCRYPT\_MODE,publicKey);

byte[] encryptedBytes = cipher.doFinal(messageToBytes);

return encode(encryptedBytes);

}

private String encode(byte[] data){

return Base64.getEncoder().encodeToString(data);

}

public String decrypt(String encryptedMessage) throws Exception{

byte[] encryptedBytes = decode(encryptedMessage);

Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");

cipher.init(Cipher.DECRYPT\_MODE,privateKey);

byte[] decryptedMessage = cipher.doFinal(encryptedBytes);

return new String(decryptedMessage,"UTF8");

}

private byte[] decode(String data){

return Base64.getDecoder().decode(data);

}

public static void main(String[] args) {

RSA rsa = new RSA();

try{

String encryptedMessage = rsa.encrypt("Hello World");

String decryptedMessage = rsa.decrypt(encryptedMessage);

System.err.println("Encrypted:\n"+encryptedMessage);

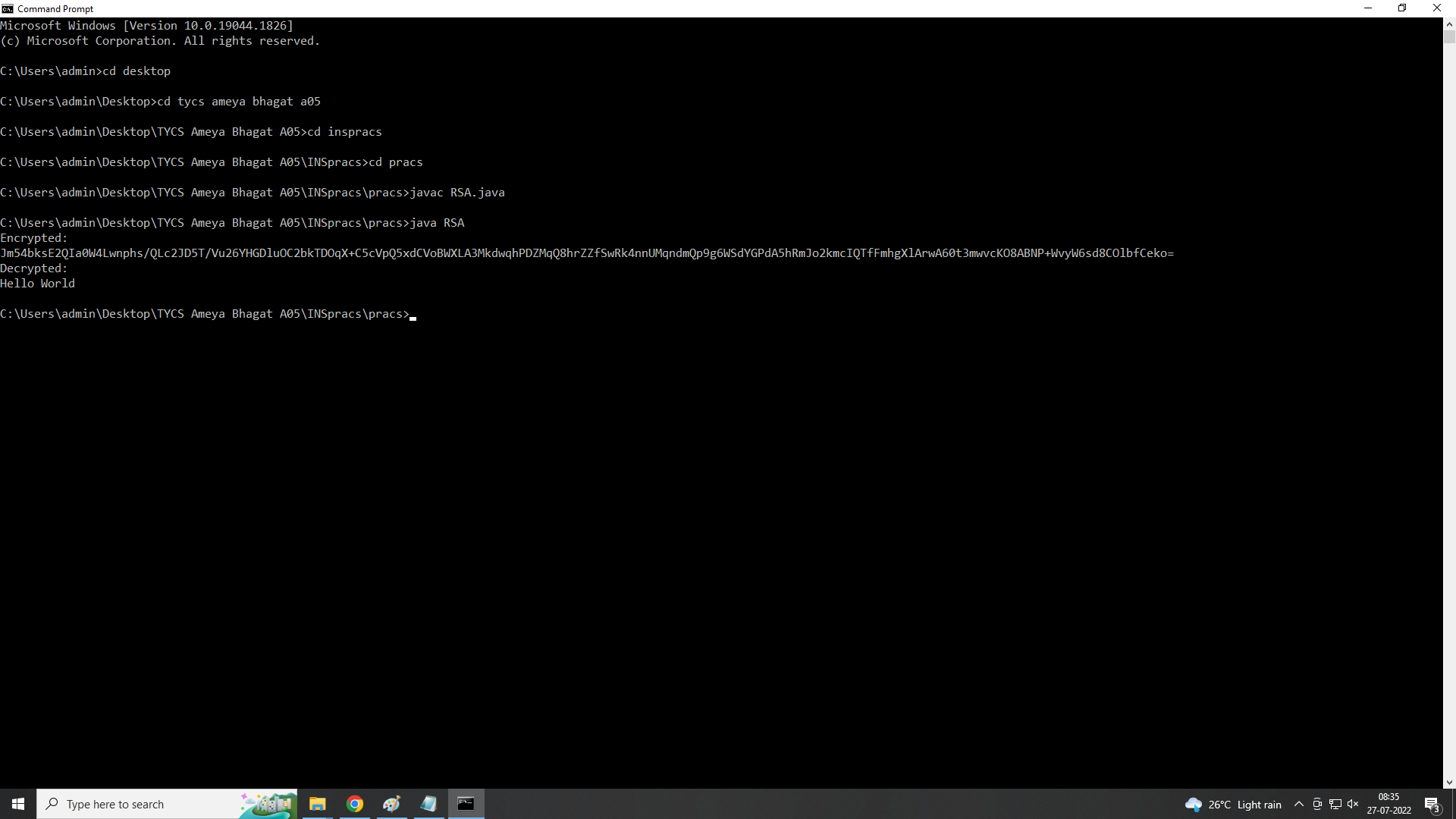
System.err.println("Decrypted:\n"+decryptedMessage);

}catch (Exception ingored){}

}

}

**Output**:



RSA (PART-2)

**Code:**

import java.util.\*;

import java.math.\*;

class RSA1

{

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

int p,q,n,z,d=0,e,i;

System.out.println("Enter the number to be encrypted and decrypted");

int msg = sc.nextInt();

double c;

BigInteger msgback;

System.out.println("Enter 1st Prime Number p");

p=sc.nextInt();

System.out.println("Enter 2nd Prime Number q");

q=sc.nextInt();

n=p\*q;

z=(p-1)\*(q-1);

System.out.println("The value of Z = "+z);

for (e=2;e<z;e++)

{

if(gcd(e,z)==1)

{

break;

}

}

System.out.println("The value of E = "+e);

for(i=0;i<=9;i++)

{

int x=1+(i\*z);

if(x%e==0)

{

d=x/e;

break;

}

}

System.out.println("The value of D = "+d);

c=(Math.pow(msg,e))%n;

System.out.println("Encrypted Message is : -");

System.out.println(c);

BigInteger N = BigInteger.valueOf(n);

BigInteger C = BigDecimal.valueOf(c).toBigInteger();

msgback = (C.pow(d)).mod(N);

System.out.println("Decrypted message is : -");

System.out.println(msgback);

}

static int gcd(int e, int z)

{

if(e==0)

return z;

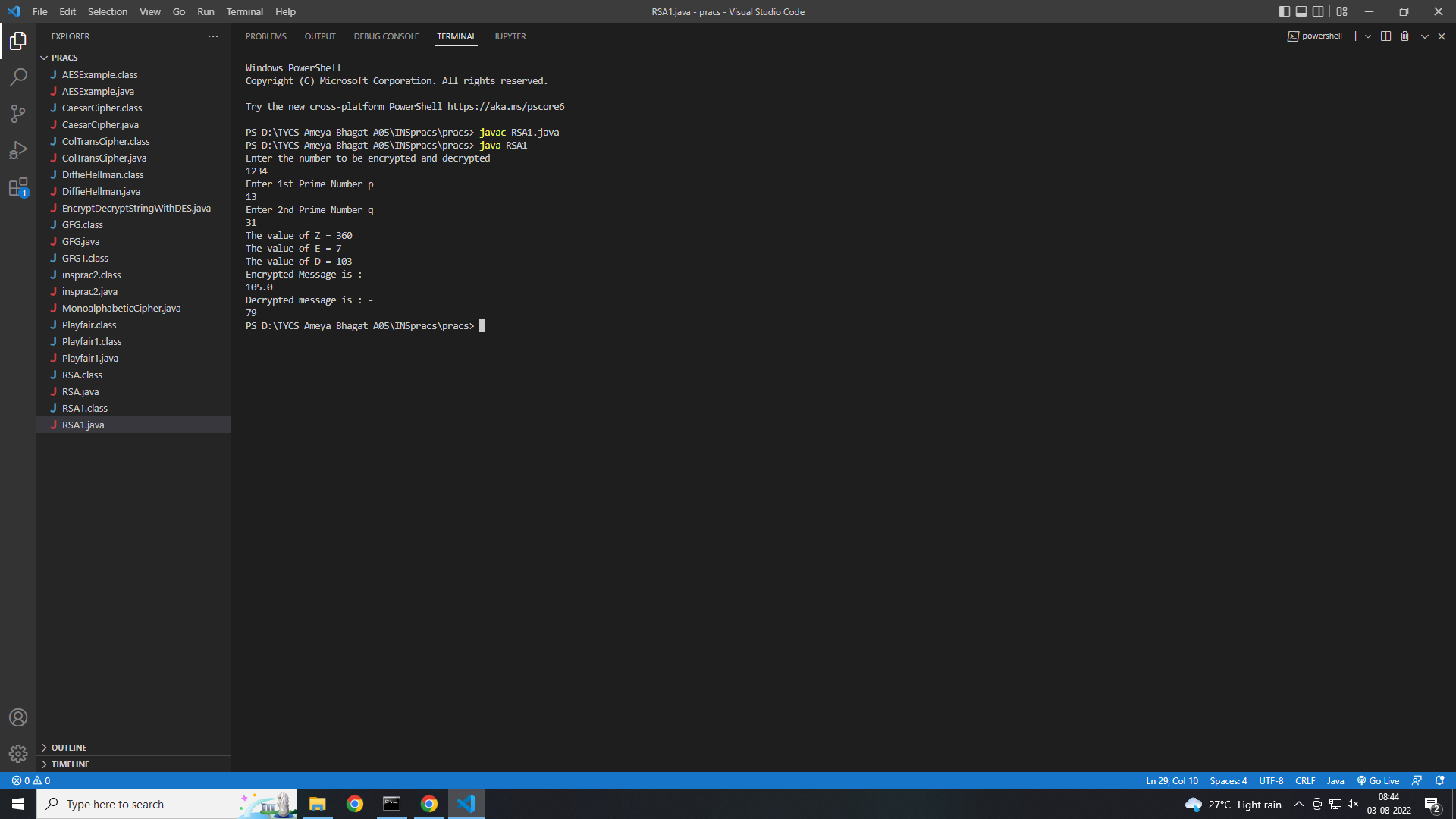
else

return gcd(z%e,e);

}

}

**OUTPUT**:



**Practical 6**: *Write a program to implement the Diffie-Hellman Key Agreement algorithm to generate symmetric keys*.

**Code**:

import java.util.\*;

class DiffieHellman {

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter the first prime number : ");

int p=sc.nextInt();

System.out.println("Enter the root of "+p);

int g=sc.nextInt();

System.out.println("Select 1st secret no. of Alice");

int a=sc.nextInt();

System.out.println("Select 2nd secret no. of Bob");

int b=sc.nextInt();

int A = (int)Math.pow(g,a)%p;

int B = (int)Math.pow(g,b)%p;

int S\_A = (int)Math.pow(B,a)%p;

int S\_B = (int)Math.pow(A,b)%p;

if(S\_A==S\_B)

{

System.out.println("They shared a secret key that is = "+S\_A);

}

else

{

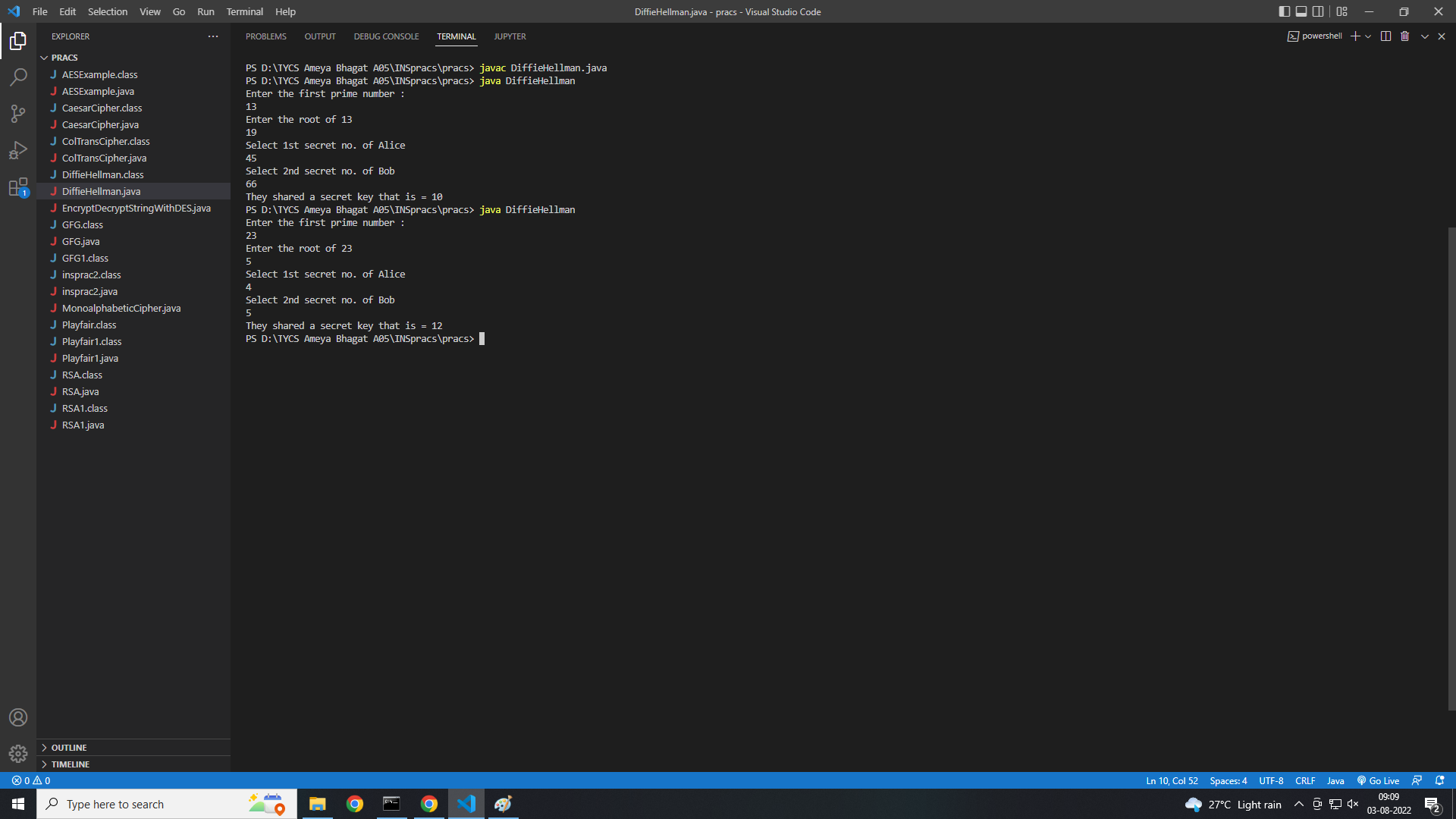
System.out.println("Alice and Bob cannot exchange information with each other");

}

}

}

**OUTPUT**:



**Practical 7:** *Write a program to implement the MD5 algorithm compute the message digest.*

**Code:**

import java.math.BigInteger;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

public class JavaMD5Hash {

public static void main(String[] args) {

System.out.println("For null " + md5(""));

System.out.println("For simple text "+ md5("This is my text"));

System.out.println("For simple numbers " + md5("12345"));

}

public static String md5(String input) {

String md5 = null;

if(null == input) return null;

try {

MessageDigest digest = MessageDigest.getInstance("MD5");

digest.update(input.getBytes(), 0, input.length());

md5 = new BigInteger(1, digest.digest()).toString(16);

}

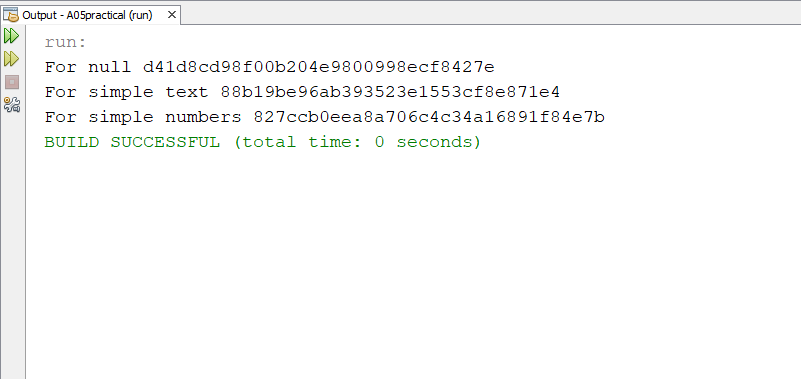
catch (NoSuchAlgorithmException e) {

e.printStackTrace();

}

return md5;}}

**Output:**

****

**Practical 8:** *Write a program to calculate HMAC-SHA1 Signature*

**Code:**

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import java.security.SignatureException;

import java.util.Formatter;

import javax.crypto.Mac;

import javax.crypto.spec.SecretKeySpec;

public class HmacSha1Signature {

private static final String HMAC\_SHA1\_ALGORITHM = "HmacSHA1";

private static String toHexString(byte[] bytes) {

Formatter formatter = new Formatter();

for (byte b : bytes) {

formatter.format("%02x", b);

}

return formatter.toString();

}

public static String calculateRFC2104HMAC(String data, String key)

throws SignatureException, NoSuchAlgorithmException, InvalidKeyException

{

SecretKeySpec signingKey = new SecretKeySpec(key.getBytes(), HMAC\_SHA1\_ALGORITHM);

Mac mac = Mac.getInstance(HMAC\_SHA1\_ALGORITHM);

mac.init(signingKey);

return toHexString(mac.doFinal(data.getBytes()));

}

public static void main(String[] args) throws Exception {

String hmac = calculateRFC2104HMAC("data", "key");

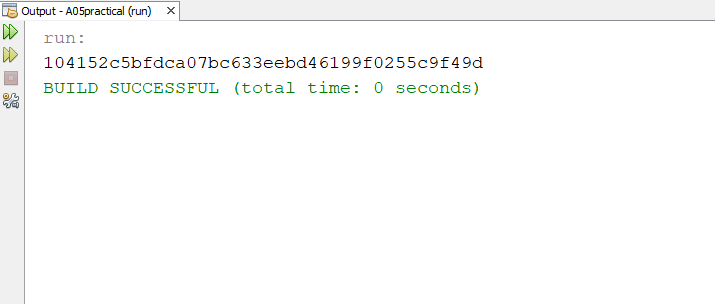
System.out.println(hmac);

assert hmac.equals("104152c5bfdca07bc633eebd46199f0255c9f49d");

}

}

**OUTPUT:**

****