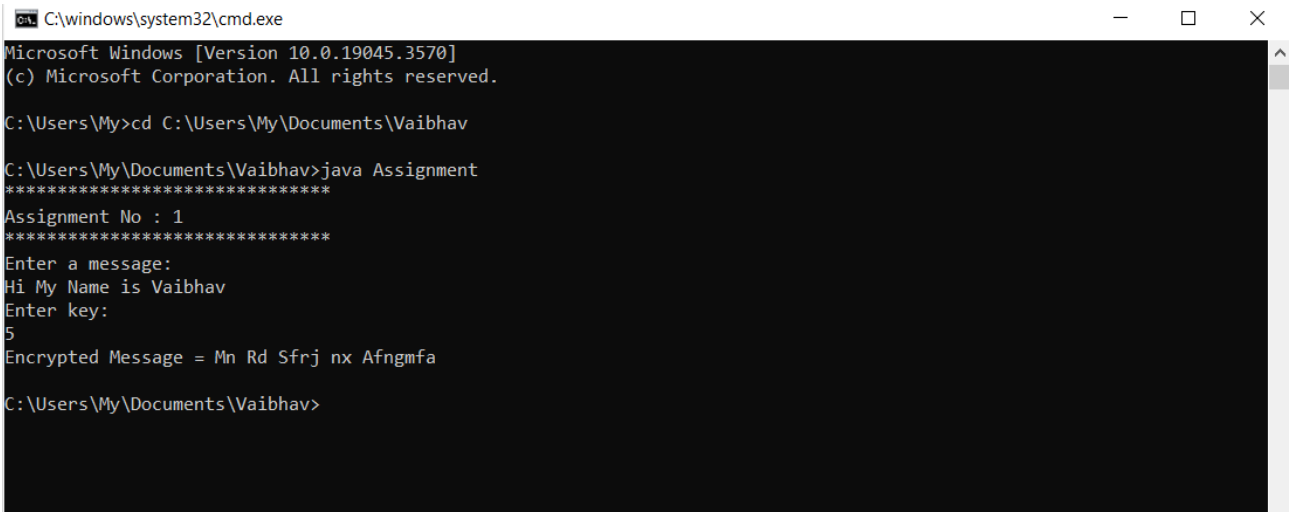
Assignment 1

```
import java.util.Scanner; public
class Assignment1
{
   public static void main(String s[])
   {
      String message, encryptedMessage = "";
      int key;
char ch;
      Scanner sc = new Scanner(System.in);
      System.out.println("*******************************");
      System.out.println("Assignment No : 1");
      System.out.println("*******************************");
System.out.println("Enter a message: ");         message =
sc.nextLine();          System.out.println("Enter key: ");         key =
sc.nextInt();         for(int i = 0; i < message.length(); ++i)
      {
         ch = message.charAt(i);
if(ch >= 'a' && ch <='z')
         {
            ch = (char)(ch + key);
            if(ch > 'z')
            {
               ch = (char)(ch - 'z' + 'a' - 1);
            }
            encryptedMessage += ch;
         }
else
         if(ch >= 'A' && ch <= 'Z')
         {
            ch = (char)(ch + key); if(ch
> 'Z')
```

```
            {
                ch = (char)(ch - 'Z' + 'A' - 1);
            }
            encryptedMessage += ch;
        }
    else
        {
            encryptedMessage += ch;
        }
    }
    System.out.println("Encrypted Message = " + encryptedMessage);
  }
}
```

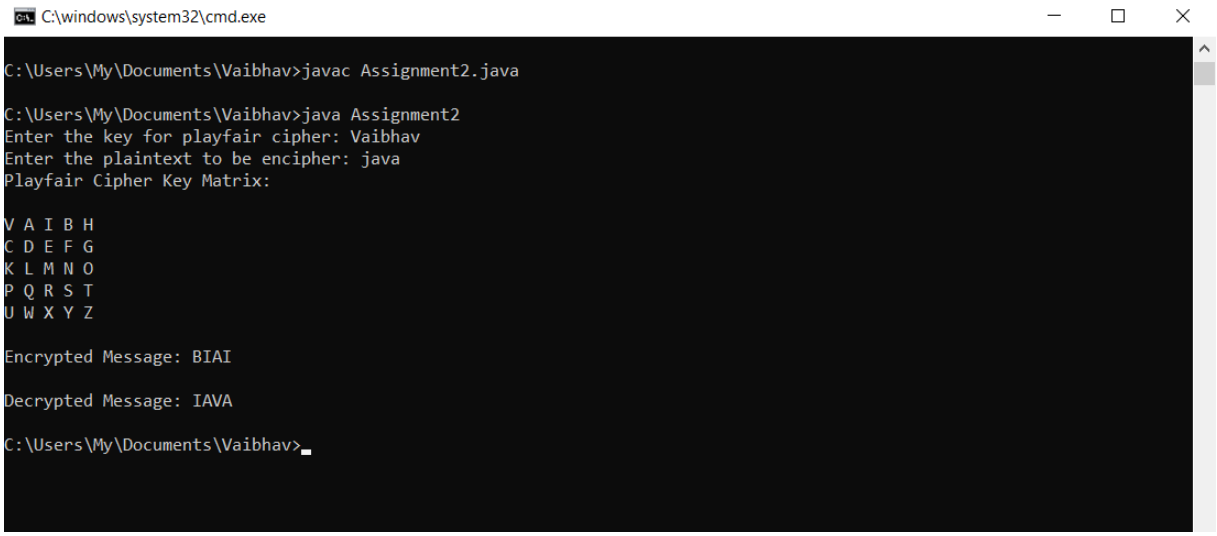


Assignment 2

```
package course; import
java.util.Arrays; import
java.util.Scanner; public
class Assignment2
  {
    private static char[][] keySquare;
    private static void
generateKeySquare(String key)
    {
      key = key.replace("J",
"I").toUpperCase();        key =
key.replaceAll("[^A-Z]", "");
      String alphabet =
"ABCDEFGHIKLMNOPQRSTUVWXYZ";
      String combinedKey = key + alphabet;
      combinedKey =
combinedKey.replaceAll("(.)(?=.*\\1)", ""); //
Remove duplicate characters
keySquare = new char[5][5];        int
rowIndex = 0;        int colIndex = 0;
      for (char ch :
combinedKey.toCharArray())
      {
        keySquare[rowIndex][colIndex] = ch;
        colIndex++;

        if (colIndex == 5)
        {
colIndex = 0;
rowIndex++;
        }
    private static String
preparePlainText(String plainText)
{
      plainText = plainText.replace("J",
"I").toUpperCase();
      plainText = plainText.replaceAll("[^AZ]",
"");
      StringBuilder preparedText = new
StringBuilder(plainText);

      for (int i = 0; i < preparedText.length(); i
+= 2)
      {
        if (i + 1 == preparedText.length())

{
          preparedText.append('X');
        }
        else if (preparedText.charAt(i) ==
preparedText.charAt(i + 1))
        {
          preparedText.insert(i + 1, 'X');
        }
      }
      return preparedText.toString();
    }
    private static String encrypt(String
plainText)
    {
      StringBuilder encryptedText = new
StringBuilder();

      for (int i = 0; i < plainText.length(); i +=
2)
      {
```

```
            char ch1 = plainText.charAt(i);
char ch2 = plainText.charAt(i + 1);
            int row1 = -1, col1 = -1, row2 = -1,
col2 = -1               char encryptedCh1,
encryptedCh2;               if (row1 == row2)
            {
                encryptedCh1 =
keySquare[row1][(col1 + 1) % 5];
                encryptedCh2 =
keySquare[row2][(col2 + 1) % 5];
            }
            else if (col1 == col2)
            {
                encryptedCh1 = keySquare[(row1 +
1) % 5][col1];
                encryptedCh2 = keySquare[(row2 +
1) % 5][col2];
            }
else
            {
                encryptedCh1 =
keySquare[row1][col2];
                encryptedCh2 =
keySquare[row2][col1];
            }


encryptedText.append(encryptedCh1).append
(encryptedCh2);
        }


        return encryptedText.toString();
    }

    private static String decrypt(String
encryptedText)
    {
        StringBuilder decryptedText = new
StringBuilder();
        for (int i = 0; i < encryptedText.length();
i += 2)
        {
            char ch1 = encryptedText.charAt(i);
            char ch2 = encryptedText.charAt(i +
1);
            int row1 = -1, col1 = -1, row2 = -1,
col2 = -1;


            for (int row = 0; row < 5; row++)
            {
                for (int col = 0; col < 5; col++)
                {
                    if (keySquare[row][col] == ch1)
                    {
                        row1 = row;
col1 = col;
                    }


                    if (keySquare[row][col] == ch2)
                    {
row2 = row;
col2 = col;
                    }
                }
            }


                decryptedCh2 = keySquare[(row2 +
4) % 5][col2];
            }
else
{

                decryptedCh1 =
keySquare[row1][col2];
```

```
                decryptedCh2 =
keySquare[row2][col1];
            }


decryptedText.append(decryptedCh1).append
(decryptedCh2);
        }


        return decryptedText.toString();
    }


public static void main(String[] args)
{
    String key = "KEYWORD";
generateKeySquare(key);
    Scanner scan = new Scanner(System.in); //
Take input from user using scanner class
    String plainText = scan.nextLine();
    String preparedText =
preparePlainText(plainText);
    String encryptedText =
encrypt(preparedText);     String
decryptedText = decrypt(encryptedText);
    System.out.println("Key Square:");
    for (char[] row : keySquare)
    {
        System.out.println(Arrays.toString(row));
    }


    System.out.println("\nPlain Text: " +
plainText);
    System.out.println("Prepared Text: " +
preparedText);
    System.out.println("Encrypted Text: " +
encryptedText);
    System.out.println("Decrypted Text: " +
decryptedText);
}
}
```

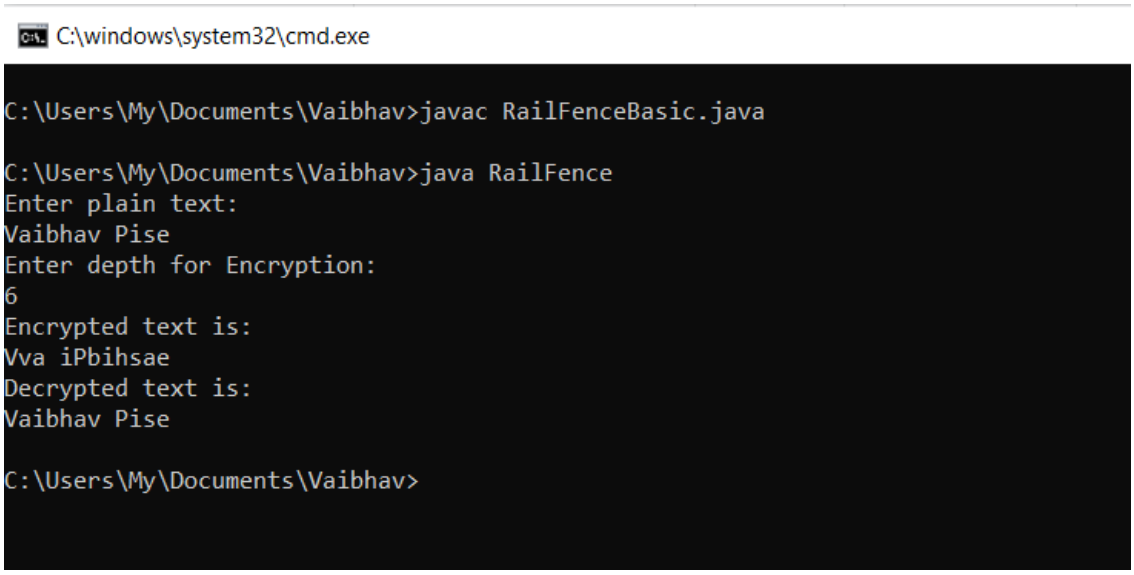


Assignment 3

```
import java.util.Arrays;
class Assignment3
{
   // function to encrypt a message     public static String
encryptRailFence(String text,  int key)
   {
      // create the matrix to cipher plain text
// key = rows , length(text) = columns
char[][] rail = new char[key][text.length()];
// filling the rail matrix to distinguish filled
      // spaces from blank ones
for (int i = 0; i < key; i++)
Arrays.fill(rail[i], '\n');
      boolean dirDown = false;
      int row = 0, col = 0;        for (int i =
0; i < text.length(); i++) {           //
check the direction of flow
         // reverse the direction if we've just
// filled the top or bottom rail           if (row
== 0 || row == key - 1)              dirDown =
!dirDown;           // fill the corresponding
alphabet           rail[row][col++] =
text.charAt(i);           // find the next row
using direction flag           if (dirDown)
row++;           else              row--;
      }
      // now we can construct the cipher using the rail
      // matrix
      StringBuilder result = new StringBuilder();
for (int i = 0; i < key; i++)           for (int j = 0; j
< text.length(); j++)
```

```
            if (rail[i][j] != '\n')
result.append(rail[i][j]);        return
result.toString();
   }
   // This function receives cipher-text and key     // and
returns the original text after decryption     public static
String decryptRailFence(String cipher, int key)
   {
         Arrays.fill(rail[i], '\n');
// to find the direction
boolean dirDown = true;
      int row = 0, col = 0;        // mark the
places with '*'        for (int i = 0; i <
cipher.length(); i++) {           // check the
direction of flow
         if (row == 0)
dirDown = true;           if
(row == key - 1)
dirDown = false;           //
place the marker
rail[row][col++] = '*';
         // find the next row using direction flag
if (dirDown)               row++;           else
row--;
      }
      // now we can construct the fill the rail matrix
int index = 0;        for (int i = 0; i < key; i++)
for (int j = 0; j < cipher.length(); j++)               if
(rail[i][j] == '*'
               && index < cipher.length())
rail[i][j] = cipher.charAt(index++);
      StringBuilder result = new StringBuilder();
      row = 0;        col = 0;        for (int i =
0; i < cipher.length(); i++) {           //
check the direction of flow
```

```
        if (row == 0)              dirDown = true;
if (row == key - 1)              dirDown = false;
if (rail[row][col] != '*')
result.append(rail[row][col++]);           // find
the next row using direction flag     public
static void main(String[] args)
  {
     // Encryption
     System.out.println("Encrypted Message: ");
     System.out.println(encryptRailFence("attack at once", 2));
     System.out.println(  encryptRailFence("GeeksforGeeks ", 3));
     System.out.println(encryptRailFence("defend the east wall", 3));
     // Now decryption of the same cipher-text
     System.out.println("\nDecrypted Message: ");
     System.out.println(decryptRailFence("atc toctaka ne", 2));
     System.out.println(decryptRailFence("GsGsekfrek eoe", 3));
     System.out.println(decryptRailFence("dnhaweedtees alf  tl", 3));
  }
}
```



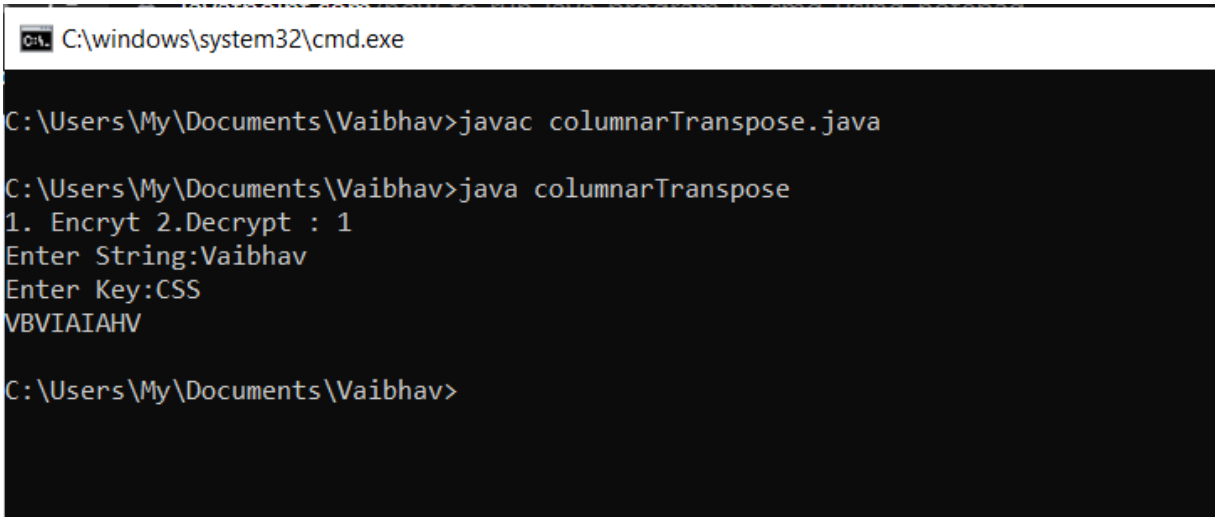
Assignment 4

```
import java.util.*;
import java.io.*;
import java.lang.*;

public class columnarTranspose {
   public static void main(String[] args) {
      Scanner scan = new Scanner(System.in);
      String line = System.getProperty("line.separator");
      scan.useDelimiter(line);

      System.out.print("1. Encryt 2.Decrypt : ");
      int option = scan.nextInt();
      switch (option) {
         case 1:
            System.out.print("Enter String:");
            String text = scan.next();

            System.out.print("Enter Key:");
            String key = scan.next();

            System.out.println(encryptCT(key, text).toUpperCase());
            break;
         case 2:
            System.out.print("Enter Encrypted String:");
            text = scan.next();

            System.out.print("Enter Key:");
            key = scan.next();

            System.out.println(decryptCT(key, text));
            break;
         default:
            break;
      }
```

```
}

public static String encryptCT(String key, String text) {
   int[] arrange = arrangeKey(key);

   int lenkey = arrange.length;
   int lentext = text.length();

   int row = (int) Math.ceil((double) lentext / lenkey);

   char[][] grid = new char[row][lenkey];
   int z = 0;
   for (int x = 0; x < row; x++) {
      for (int y = 0; y < lenkey; y++) {
         if (lentext == z) {
            // at random alpha for trailing null grid
            grid[x][y] = RandomAlpha();
            z--;
         } else {
            grid[x][y] = text.charAt(z);
         }

         z++;
      }
   }
   String enc = "";
   for (int x = 0; x < lenkey; x++) {
      for (int y = 0; y < lenkey; y++) {
         if (x == arrange[y]) {
            for (int a = 0; a < row; a++) {
               enc = enc + grid[a][y];
            }
         }
      }
   }
```

```
    return enc;
}

public static String decryptCT(String key, String text) {
    int[] arrange = arrangeKey(key);
    int lenkey = arrange.length;
    int lentext = text.length();

    int row = (int) Math.ceil((double) lentext / lenkey);

    String regex = "(?<=\\G.{" + row + "})";
    String[] get = text.split(regex);

    char[][] grid = new char[row][lenkey];

    for (int x = 0; x < lenkey; x++) {
        for (int y = 0; y < lenkey; y++) {
            if (arrange[x] == y) {
                for (int z = 0; z < row; z++) {
                    grid[z][y] = get[arrange[y]].charAt(z);
                }
            }
        }
    }

    String dec = "";
    for (int x = 0; x < row; x++) {
        for (int y = 0; y < lenkey; y++) {
            dec = dec + grid[x][y];
        }
    }

    return dec;
}
```

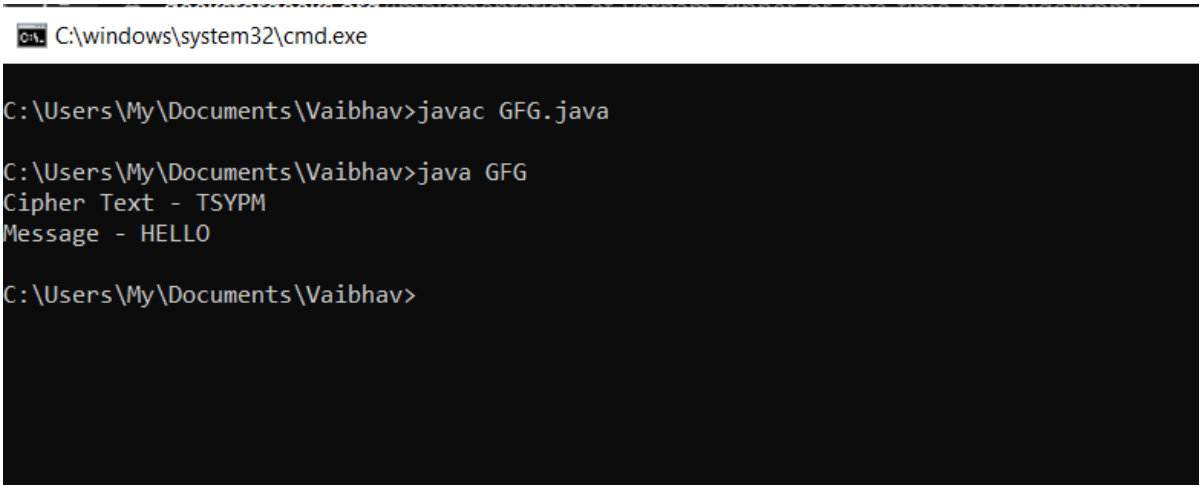
```
    public static char RandomAlpha() {
        //generate random alpha for null space
        Random r = new Random();
        return (char)(r.nextInt(26) + 'a');
    }

    public static int[] arrangeKey(String key) {
        //arrange position of grid
        String[] keys = key.split("");
        Arrays.sort(keys);
        int[] num = new int[key.length()];
        for (int x = 0; x < keys.length; x++) {
            for (int y = 0; y < key.length(); y++) {
                if (keys[x].equals(key.charAt(y) + "")) {
                    num[y] = x;
                    break;
                }
            }
        }

        return num;
    }

}
```



Assignment 5

```
import java.util.Random;
import java.util.Scanner; public
class Assignment5 {
// Function to generate a random key (pad) of the same length as the plaintext
public static String generateRandomKey(int length) { Random random = new
Random();
StringBuilder keyBuilder = new StringBuilder();
for (int i = 0; i < length; i++) {
char randomChar = (char) (random.nextInt(26) + 'A'); // Generates a random uppercase letter
keyBuilder.append(randomChar);
}
return keyBuilder.toString();
}
// Function to perform one-time pad encryption
public static String encrypt(String plaintext, String key) { if (plaintext.length() !=
key.length()) { throw new IllegalArgumentException("Plaintext and key must have the
same length.");
}
StringBuilder ciphertextBuilder = new StringBuilder();
for (int i = 0; i < plaintext.length(); i++) { char encryptedChar = (char)
((plaintext.charAt(i) + key.charAt(i)) % 26 + 'A');
ciphertextBuilder.append(encryptedChar);
}
return ciphertextBuilder.toString();
}
// Function to perform one-time pad decryption public static String decrypt(String
ciphertext, String key) { if (ciphertext.length() != key.length()) { throw new
IllegalArgumentException("Ciphertext and key must have the same length.");
}
StringBuilder decryptedBuilder = new StringBuilder();
for (int i = 0; i < ciphertext.length(); i++) { char decryptedChar = (char)
((ciphertext.charAt(i) - key.charAt(i) + 26) % 26 + 'A');
decryptedBuilder.append(decryptedChar);
```

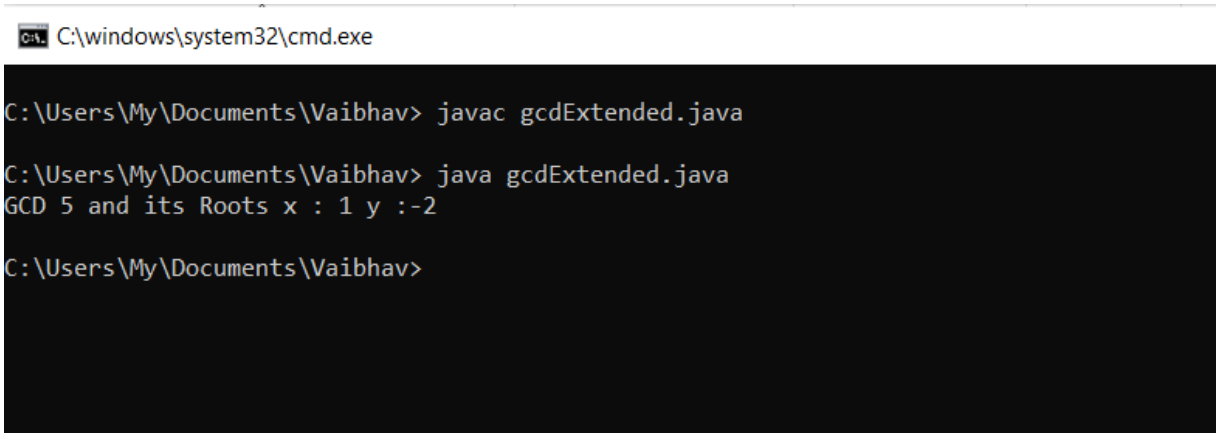
```
}
return decryptedBuilder.toString();
}
public static void main(String[] args) {
// Input string from user
Scanner scan = new Scanner(System.in);
String randomtext = scan.nextLine();
String plaintext = randomtext.toUpperCase();
String key = generateRandomKey(plaintext.length());
System.out.println("Plaintext: " + plaintext);
System.out.println("Key: " + key);
String ciphertext = encrypt(plaintext, key);
System.out.println("Ciphertext: " + ciphertext);
String decryptedText = decrypt(ciphertext, key);
System.out.println("Decrypted Text: " + decryptedText);
}
}
```



Assignment 6

```
// Java program to demonstrate working of extended
// Euclidean Algorithm

import java.util.*;
import java.lang.*;

class GFG {
	static public void gcdExtended(long a, long b)
	{
		long x = 0, y = 1, lastx = 1, lasty = 0, temp;
		while (b != 0)
		{
			long q = a / b;
			long r = a % b;

			a = b;
			b = r;

			temp = x;
			x = lastx - q * x;
			lastx = temp;

			temp = y;
			y = lasty - q * y;
			lasty = temp;
		}
```

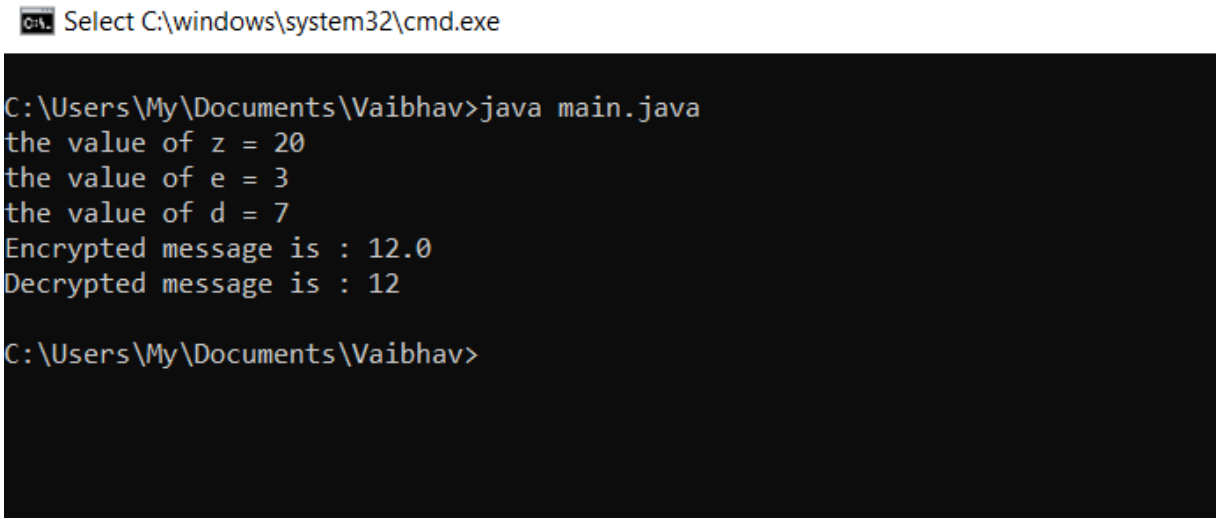
```
        System.out.println("GCD "+a+" and its Roots x : "+ lastx +" y :"+ lasty);


        }


        // Driver Program
        public static void main(String[] args)
        {


                long a = 35, b = 15;
        //this will print result like
        //Roots x : 1 y :-2
        gcdExtended(a, b);


        }
}
```



Assignment 7

```
// Java Program to Implement the RSA Algorithm
import java.math.*;
import java.util.*;

class RSA {
	public static void main(String args[])
	{
		int p, q, n, z, d = 0, e, i;

		// The number to be encrypted and decrypted
		int msg = 12;
		double c;
		BigInteger msgback;

		// 1st prime number p
		p = 3;

		// 2nd prime number q
		q = 11;
		n = p * q;
		z = (p - 1) * (q - 1);
		System.out.println("the value of z = " + z);

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

			// e is for public key exponent
			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);

			// d is for private key exponent
			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 : " + c);

		// converting int value of n to BigInteger
		BigInteger N = BigInteger.valueOf(n);
```

```
                // converting float value of c to BigInteger
                BigInteger C = BigDecimal.valueOf(c).toBigInteger();
                msgback = (C.pow(d)).mod(N);
                System.out.println("Decrypted message is : "
                                                        + msgback);
        }

        static int gcd(int e, int z)
        {
                if (e == 0)
                        return z;
                else
                        return gcd(z % e, e);
        }
}
```



Assignment 8

```
import java.security.*; import
java.util.Base64;


public class Assignment8 { public static void
main(String[] args) throws Exception {
// Generate a key pair
KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
keyPairGenerator.initialize(2048);
KeyPair keyPair = keyPairGenerator.generateKeyPair();
// Get the private key
PrivateKey privateKey = keyPair.getPrivate();
// Get the message to be signed
String message = "This is a message to be signed.";
// Create a signature object
Signature signature = Signature.getInstance("SHA256withRSA"); //
Initialize the signature object with the private key
signature.initSign(privateKey);
// Add the message to the signature object signature.update(message.getBytes());
// Calculate the signature byte[]
signatureBytes = signature.sign();
// Save the signature
String signatureString = Base64.getEncoder().encodeToString(signatureBytes);
System.out.println("Signature: " + signatureString);
// Verify the signature
Signature verificationSignature = Signature.getInstance("SHA256withRSA");
// Initialize the verification signature object with the public key
verificationSignature.initVerify(keyPair.getPublic()); // Add the
message to the verification signature object
verificationSignature.update(message.getBytes());
// Verify the signature
boolean isVerified = verificationSignature.verify(signatureBytes);
```

```
System.out.println("Signature verified: " + isVerified);
}
}
```

Output

Assignment 9

```
import java.math.BigInteger;

EllipticCurvePoint {

 private BigInteger x;

private BigInteger y;

public EllipticCurvePoint(BigInteger x, BigInteger y) {

 this.x =

x;

this.y = y;

   }

   public static EllipticCurvePoint add(EllipticCurvePoint P, EllipticCurvePoint Q, BigInteger a,
BigInteger p) {

      BigInteger

slope, x3, y3;       if

(P.equals(Q)) {

         slope =
(BigInteger.valueOf(3).multiply(P.x.pow(2)).add(a)).multiply(P.y.multiply(BigInteger.valueOf(2
)).modIn verse(p));

      } else {

         slope = (Q.y.subtract(P.y)).multiply(Q.x.subtract(P.x).modInverse(p));

      }

      x3 =

slope.pow(2).subtract(P.x).subtract(Q.x).mod(p);

y3 =

slope.multiply(P.x.subtract(x3)).subtract(P.y).mod(p

);       return new EllipticCurvePoint(x3, y3);

   }

   public static EllipticCurvePoint scalarMultiply(EllipticCurvePoint P, BigInteger k, BigInteger
a, BigInteger p) {

      EllipticCurvePoint result = new EllipticCurvePoint(BigInteger.ZERO,

BigInteger.ZERO);       EllipticCurvePoint current = P;       while

(k.compareTo(BigInteger.ZERO) > 0) {
```

```
        if (k.testBit(0)) {
result = add(result, current, a,
p);
        }
        current = add(current,
current, a, p);          k =
k.shiftRight(1);
      }
      return result;
   }
   @Override
public String
toString() {
return "(" + x + ", " +
y + ")";
   }
   public boolean equals(EllipticCurvePoint
other) {       return x.equals(other.x) &&
y.equals(other.y);
   }
}
public class Exp9 {    public
static void main(String[]
args) {
      // Define the elliptic curve parameters
      BigInteger a = new BigInteger("1");
      BigInteger b = new BigInteger("6");
      BigInteger p = new BigInteger("11");
      // Define a base point P on the curve
      EllipticCurvePoint P = new EllipticCurvePoint(new BigInteger("2"), new BigInteger("7"));
      // Scalar multiplication: Compute 3P
      BigInteger scalar = new BigInteger("3");
      EllipticCurvePoint result = EllipticCurvePoint.scalarMultiply(P, scalar, a, p);
      System.out.println("Base Point P: " + P);
```

```
        System.out.println("Result of Scalar Multiplication (3P): " + result);
    }
}
```

```
G:\My Drive\Study material\B Tech\7th sem\css\Practical\Code>
Base Point P: (2, 7)
Result of Scalar Multiplication (3P): (2, 7)
```

Assignment 10

```
// This program calculates the Key for two persons using the
Diffie-Hellman Key exchange algorithm class Assignment10 {
        // Power function to return value of a ^ b mod P
private static long power(long a, long b, long p)
        {
                if (b == 1)
                        return a;
                else
                        return (((long)Math.pow(a, b)) % p);
        }
        public static void main(String[] args)
        {
                long P, G, x, a, y, b, ka, kb;
             // Both the persons will be agreed upon the public
keys G and P

                // A prime number P is taken
                P = 23;
                System.out.println("The value of P:" + P);
                // A primitive root for P, G is taken
                G = 9;
                System.out.println("The value of G:" + G);
                // Alice will choose the private key a
                // a is the chosen private key
a = 4;
           System.out.println("The private key a for Alice:" + a);
                // Gets the generated key
        x = power(G, a, P);
                // Bob will choose the private key b
```

```
                    // b is the chosen private key
b = 3;
                System.out.println("The private key b for Bob:" + b);


                    // Gets the generated key
            y = power(G, b, P);
                   // Generating the secret key after the exchange
                    // of keys
                    ka = power(y, a, P); // Secret key for Alice
            kb = power(x, b, P); // Secret key for Bob


              System.out.println("Secret key for the Alice is:" + ka);
               System.out.println("Secret key for the Bob is:" + kb);
            }
}
```

C:\windows\system32\cmd.exe

```
C:\Users\My\Documents\Vaibhav>javac Assignment10.java

C:\Users\My\Documents\Vaibhav>java Assignment10
The value of P:23
The value of G:9
The private key a for Alice:4
The private key b for Bob:3
Secret key for the Alice is:9
Secret key for the Bob is:9

C:\Users\My\Documents\Vaibhav>
```