**GCD**

import java.util.Scanner;

public class gcd {

static int find(int a,int b){

int r1=a;

int r2=b;

int q=0;

int r=0;

while(r2>0){

q=r1/r2;

r=r1-q\*r2;

r1=r2;

r2=r;

}

return r1;

}

public static void main(String[] args){

Scanner in=new Scanner(System.in);

System.out.print("enter a: ");

int a=in.nextInt();

System.out.print("enter b: ");

int b=in.nextInt();

System.out.println("GCD: "+find(a,b));

}

}

**Extended GCD**

package test;

import java.util.Scanner;

public class extendedgcd {

static int find(int a,int b,int s1,int t1) {

int s2=0,t2=1,r1=a,r2=b,q=0,r=0,s,t;

while(r2>0){

q=r1/r2;

r=r1-q\*r2;

r1=r2;

r2=r;

s=s1-q\*s2;

s1=s2;

s2=s;

t=t1-q\*t2;

t1=t2;

t2=t;

}

System.out.println("s: "+s1+" t: "+t1);

return r1;

}

public static void main(String[] args){

Scanner in=new Scanner(System.in);

System.out.print("enter a: ");

int a=in.nextInt();

System.out.print("enter b: ");

int b=in.nextInt();

int s1=1; int t1=0;

System.out.println("GCD: "+find(a,b,s1,t1));

}

}

**CAESER**

import java.util.Scanner;

public class ceasar

{

static String alphabet="abcdefghijklmnopqrstuvwxyz";

public static void main(String arg[])

{

Scanner sc=new Scanner(System.in);

String plain,cipher,decipher;

System.out.println("Enter the plain text:");

plain=sc.nextLine();

cipher="";

decipher="";

plain=plain.toLowerCase();

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

{

cipher+=alphabet.charAt((alphabet.indexOf(plain.charAt(i))+3)%26);

}

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

System.out.println(cipher);

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

{

int check=(alphabet.indexOf(cipher.charAt(i))-3)%26;

if(check<0)

check=26+check;

decipher+=alphabet.charAt(check);

}

System.out.println("Decrypted Text:");

System.out.println(decipher);

}

}

**Affine**

import java.util.Scanner;

public class Affine {

public static void main (String[] args) {

Scanner sc=new Scanner(System.in);

int a,b;

int x=0;

String plain,cipher,decipher;

cipher="";decipher="";

System.out.println("Enter plain text : ");

plain=sc.nextLine();

System.out.println("Enter a and b :");

a = sc.nextInt();

b=sc.nextInt();

if(gcd(a,26)==1) {

String alpha="abcdefghijklmnopqrstuvwxyz";

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

char c=plain.charAt(i);

int y=(a\*(alpha.indexOf(c))+b)%26;

cipher+=alpha.charAt(y);

}

System.out.println("Cipher :"+cipher);

int invert=inver(a);

System.out.println("Inverse :"+invert);

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

char c=cipher.charAt(i);

int y=invert\*(alpha.indexOf(c)-b);

if(y<0)

x=y+26;

else

x=y%26;

decipher+=alpha.charAt(x);

}

System.out.println("Decipher :"+decipher);

}

else {

System.out.println("Affine cipher not possible");

}

}

public static int inver(int a) {

int ans=0;

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

if(((a\*j)%26)==1) {

ans=j;

break;

}

}

return ans;

}

public static int gcd(int a, int b) {

if(b==0) return a;

else

return gcd(b,a%b);

}

}

**Railfence**

import java.util.Scanner;

public class railfence {

public static String encrypt(String plain,int key) {

String cipher="";

int r=key,len=plain.length();

int c=(len/r);

char mat[][]=new char[r][c];

int k=0;

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

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

if(k!=len)

mat[j][i]=plain.charAt(k++);

else

mat[j][i]='X';

}

}

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

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

cipher+=mat[i][j];

}

}

return cipher;

}

public static String decrypt(String cipher, int key) {

String decipher="";

int r=key,len=cipher.length();

int c=(len/r);

char mat[][]=new char[r][c];

int k=0;

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

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

mat[i][j]=cipher.charAt(k++);

}

}

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

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

decipher+=mat[j][i];

}

}

return decipher;

}

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

String plain,cipher,decipher;

int key;

System.out.println("Enter plain text :");

plain=sc.nextLine();

System.out.println("Enter key :");

key=sc.nextInt();

cipher=encrypt(plain,key);

System.out.println("Cipher :"+cipher);

decipher=decrypt(cipher,key);

System.out.println("Decipher :"+decipher);

}

}

**COLUMNAR**

public class columnar {

     public static void main(String[] args) {

     Scanner scan=new Scanner(System.in);

         System.out.println("columnar");

     System.out.println("Enter the Plain Text ");

     String PT=scan.next();

     PT=PT.replaceAll(" ", "");

     System.out.println("Enter the Keyword with the no of integers ");

     int n = scan.nextInt();

     int k[]=new int[n];

     for(int i=0;i<n;i++)

     {

         k[i]=scan.nextInt();

     }

     int l=PT.length();

     if(l%3 != 0){

         l=(l/n)+1;

     }else{

          l=l/n;

     }

     int mat[][]=new int[l][n];

     int p=0;

     for(int i=0;i<l;i++)

     {

         for(int j=0;j<n;j++){

             if(p != PT.length() ){

             mat[i][j]=(PT.charAt(p)-97);

             }else{

                 mat[i][j]=-2;

             }

             p++;

         }

     }

     for (int a=0;a<l;a++){

         for (int j=0;j<n;j++){

                System.out.print((char)(mat[a][j]+97)+" ");

         }

         System.out.println();

     }

     String CT="";

     int s=1,i;

     while(s<=n){

         for( i=0;i<n;i++){

       if(k[i]==s){

           for(int j=0;j<l;j++){

            CT+=(char)(mat[j][i]+97);

        }

       }

     }

      s++;

     }

     System.out.println("The Cipher Text is "+CT);

    PT="";

    for (int a=0;a<l;a++){

         for (int j=0;j<n;j++){

             char b=(char)(mat[a][j]+97);

             if(b!='\_')

             {

                 PT+=b;

             }

         }

     }

    System.out.println("The Plain Text is "+ PT);

}

}

**HILL CIPHER**

public class hillcipher {

    static String alphabet="abcdefghijklmnopqrstuvwxyz";

public static int check(int a)

{

     while(a<0)

         a+=26;

     return a;

}

public static int[][] multiply(int r,int nc,int a[][],int b[][])

{

     int c[][]=new int[r][nc];

     for(int i=0;i<r;i++)

         for(int j=0;j<nc;j++)

         {

             c[i][j]=0;

             for(int k=0;k<r;k++)

             {

                 c[i][j]+=(a[i][k]\*b[k][j]);

             }

             c[i][j]%=26;

             while(c[i][j]<0)

                 c[i][j]+=26;

         }

     return c;

}

public static int gcd(int a,int b)

{

     if(b==0)

         return a;

     return gcd(b,a%b);

}

public static int inverse(int a)

{

     int q,r1=26,r2=a,r,t1=0,t2=1,t;

     while(r2>0)

     {

         q=r1/r2;

         r=r1%r2;

         t=t1-q\*t2;

         t1=t2;

         t2=t;

         r1=r2;

         r2=r;

     }

     if(t1<0)

         t1=t1+26;

     return t1;

}

public static void main(String[] args)

{

     Scanner sc=new Scanner(System.in);

     String plaintext,encrypt,decrypt;

     System.out.println("hill cipher");

     System.out.println("Enter the plaintext to be encrypted:");

     plaintext=sc.nextLine();

     encrypt="";

     decrypt="";

     int r=0,c=0,kd=0,ch=1;

     while(ch==1)

     {

         System.out.println("Enter the row and column of key matrix:");

         r=sc.nextInt();

         c=sc.nextInt();

            if((r==2&&c==2)||(r==3&&c==3))

                 ch=0;

         else

             System.out.println("Enter the valid row and column");

     }

     int k[][],ki[][],kdi;

     if(r==2)

         k=new int[2][2];

     else

         k=new int[3][3];

     ch=1;

     while(ch==1)

     {

         if(r==2&&c==2)

         {

             System.out.println("Enter the key matrix:(0-25)");

             for(int i=0;i<r;i++)

                 for(int j=0;j<c;j++)

                     k[i][j]=sc.nextInt();

                kd=(k[0][0]\*k[1][1])-(k[0][1]\*k[1][0]);

                kd%=26;

             while(kd<0)

                 kd+=26;

             //System.out.println("d: "+kd);

             if(gcd(kd,26)==1)

                 ch=0;

             else

                    System.out.println("Enter a valid key matrix");

         }

         else if(r==3&&c==3)

         {

             System.out.println("Enter the key matrix:(0-25)");

             for(int i=0;i<r;i++)

                 for(int j=0;j<c;j++)

                     k[i][j]=sc.nextInt();

                kd=(k[0][0]\*k[1][1]\*k[2][2])-(k[0][0]\*k[1][2]\*k[2][1])+(k[0][1]\*k[1][2]\*k[2][0])-(k[0][1]\*k[1][0]\*k[2][2])+(k[0][2]\*k[1][0]\*k[2][1])-(k[0][2]\*k[1][1]\*k[2][0]);

             kd%=26;

             while(kd<0)

                 kd+=26;

             if(gcd(kd,26)==1)

                 ch=0;

             else

                    System.out.println("Enter a valid key matrix");

             //System.out.println("d:"+kd);

                //System.out.println("d:"+(kd%26));

         }

     }

     int p[][],e[][],d[][];

     if(r==2&&c==2)

     {

         int nc;

         if(plaintext.length()%2==0)

         {

             nc=plaintext.length()/2;

             p=new int[2][nc];

         }

         else

         {

             nc=plaintext.length()/2+1;

             p=new int[2][nc];

         }

         //System.out.println(nc);

         int l=0;

         for(int i=0;i<nc;i++)

             for(int j=0;j<r;j++)

             {

                    if(i==nc-1&&j==r-1&&(plaintext.length()%2==1))

                        p[j][i]=alphabet.indexOf('x');

                else

                        p[j][i]=alphabet.indexOf(plaintext.charAt(l++));

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

             }

            //System.out.println("\n");

         e=new int[r][nc];

         e=multiply(r,nc,k,p);

         for(int i=0;i<nc;i++)

             for(int j=0;j<r;j++)

             {

                    encrypt+=alphabet.charAt(e[j][i]);

             }

         System.out.println("Encrypted text: "+encrypt);

         ki=new int[2][2];

         kdi=inverse(kd);

         //System.out.println("di: "+kdi);

         ki[0][0]=kdi\*k[1][1]%26;

         ki[1][1]=kdi\*k[0][0]%26;

         ki[0][1]=kdi\*(-k[0][1])%26;

         ki[1][0]=kdi\*(-k[1][0])%26;

         d=new int[r][nc];

         d=multiply(r,nc,ki,e);

         for(int i=0;i<nc;i++)

             for(int j=0;j<r;j++)

             {

                    decrypt+=alphabet.charAt(d[j][i]);

             }

            if(decrypt.charAt(decrypt.length()-1)=='x')

         {

             int len=decrypt.length();

             char[] temp = decrypt.toCharArray();

             decrypt="";

             for(int i=0;i<len-1;i++)

                 decrypt+=temp[i];

         }

         System.out.println("Decrypted text: "+decrypt);

     }

     else if(r==3&&c==3)

     {

         int nc;

         if(plaintext.length()%3==0)

         {

             nc=plaintext.length()/3;

             p=new int[3][nc];

         }

         else

         {

             nc=plaintext.length()/3+1;

             p=new int[3][nc];

         }

         //System.out.println(nc);

         int l=0;

        for(int i=0;i<nc;i++)

             for(int j=0;j<r;j++)

             {

                    //System.out.println("i:"+i+" j:"+j);

                    if(i==nc-1&&j==r-1&&((plaintext.length()%3==2)||(plaintext.length()%3==1)))

                    p[j][i]=alphabet.indexOf('x');

                 else if(i==nc-1&&j==r-2&&(plaintext.length()%3==1))

                        p[j][i]=alphabet.indexOf('x');

                 else

                        p[j][i]=alphabet.indexOf(plaintext.charAt(l++));

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

             }

            //System.out.println("\n");

         e=new int[r][nc];

         e=multiply(r,nc,k,p);

         for(int i=0;i<nc;i++)

             for(int j=0;j<r;j++)

             {

                    encrypt+=alphabet.charAt(e[j][i]);

             }

         System.out.println("Encrypted text: "+encrypt);

         ki=new int[3][3];

         kdi=inverse(kd);

         //System.out.println("di: "+kdi);

            ki[0][0]=kdi\*((k[1][1]\*k[2][2])-(k[1][2]\*k[2][1]))%26;

            ki[0][1]=kdi\*((k[0][2]\*k[2][1])-(k[0][1]\*k[2][2]))%26;

            ki[0][2]=kdi\*((k[0][1]\*k[1][2])-(k[0][2]\*k[1][1]))%26;

        ki[1][0]=kdi\*((k[1][2]\*k[2][0])-(k[1][0]\*k[2][2]))%26;

            ki[1][1]=kdi\*((k[0][0]\*k[2][2])-(k[0][2]\*k[2][0]))%26;

            ki[1][2]=kdi\*((k[0][2]\*k[1][0])-(k[0][0]\*k[1][2]))%26;

            ki[2][0]=kdi\*((k[1][0]\*k[2][1])-(k[1][1]\*k[2][0]))%26;

            ki[2][1]=kdi\*((k[0][1]\*k[2][0])-(k[0][0]\*k[2][1]))%26;

            ki[2][2]=kdi\*((k[0][0]\*k[1][1])-(k[0][1]\*k[1][0]))%26;

         for(int i=0;i<3;i++)

             for(int j=0;j<3;j++)

             {

                 if(ki[i][j]<0)

                        ki[i][j]=check(ki[i][j]);

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

             }

         d=new int[r][nc];

         d=multiply(r,nc,ki,e);

         for(int i=0;i<nc;i++)

             for(int j=0;j<r;j++)

             {

                    decrypt+=alphabet.charAt(d[j][i]);

             }

            if(decrypt.charAt(decrypt.length()-1)=='x')

         {

             int len=decrypt.length();

             char[] temp = decrypt.toCharArray();

                if(decrypt.charAt(decrypt.length()-2)=='x')

             {

                 decrypt="";

                 for(int i=0;i<len-2;i++)

                     decrypt+=temp[i];

            }

             else

             {

                 decrypt="";

                 for(int i=0;i<len-1;i++)

                     decrypt+=temp[i];

             }

         }

         System.out.println("Decrypted text: "+decrypt);

     }}}

**RSA**

import java.util.\*;

import java.math.\*;

class rsa

{

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) // e is for public key exponent

{

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 is for private key exponent

{

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);

//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("Derypted message is : -");

System.out.println(msgback);

}

static int gcd(int a, int b)

{

if(b==0)

return a;

else

return gcd(b,a%b);

}

}

**Bruteforce attack**

import java.util.\*;

public class Bruteforce {

public static String plain(String ct,int key)

{

String out="";

int j=ct.length();

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

{

int k=ct.charAt(i)-97;

if(k<0 || k>26)

out+=ct.charAt(i);

else

{

int add=k-key;

if(add<0)

add+=26;

int mod=add%26;

int c=mod +97;

char h=(char)c;

out+=h;

}

}

return out;

}

public static void main(String[] args) {

Scanner s= new Scanner(System.in);

//int key=3;

System.out.print("Cipher text: ");

String name=s.nextLine();

System.out.println("Brute Force Attack: List Of Plain Text");

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

{

String pt=plain(name,i);

System.out.println("PT for key \*"+i+"\* : "+pt);

}

}

}

**AES**

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 aes

{

/\* Private variable declaration \*/

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

private static final String SALTVALUE = "abcdefg";

/\* Encryption Method \*/

public static String encrypt(String strToEncrypt)

{

try

{

   /\* Declare a byte array. \*/

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

   IvParameterSpec ivspec = new IvParameterSpec(iv);

   /\* Create factory for secret keys. \*/

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

   /\* PBEKeySpec class implements KeySpec interface. \*/

   KeySpec spec = new PBEKeySpec(SECRET\_KEY.toCharArray(), SALTVALUE.getBytes(), 65536, 256);

   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);

   /\* Retruns encrypted value. \*/

   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;

}

/\* Decryption Method \*/

public static String decrypt(String strToDecrypt)

{

try

{

   /\* Declare a byte array. \*/

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

   IvParameterSpec ivspec = new IvParameterSpec(iv);

   /\* Create factory for secret keys. \*/

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

   /\* PBEKeySpec class implements KeySpec interface. \*/

   KeySpec spec = new PBEKeySpec(SECRET\_KEY.toCharArray(), SALTVALUE.getBytes(), 65536, 256);

   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);

   /\* Retruns decrypted value. \*/

   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;

}

/\* Driver Code \*/

public static void main(String[] args)

{

     /\* Message to be encrypted. \*/

     String originalval = "AES Encryption";

     /\* Call the encrypt() method and store result of encryption. \*/

     String encryptedval = encrypt(originalval);

     /\* Call the decrypt() method and store result of decryption. \*/

     String decryptedval = decrypt(encryptedval);

     /\* Display the original message, encrypted message and decrypted message on the console. \*/

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

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

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

}

}

**S-DES**

class SDES

    {

     public int K1, K2;

     public static final int P10[] = { 3, 5, 2, 7, 4, 10, 1, 9, 8, 6};

     public static final int P10max = 10;

     public static final int P8[] = { 6, 3, 7, 4, 8, 5, 10, 9};

     public static final int P8max = 10;

     public static final int P4[] = { 2, 4, 3, 1};

     public static final int P4max = 4;

     public static final int IP[] = { 2, 6, 3, 1, 4, 8, 5, 7};

     public static final int IPmax = 8;

     public static final int IPI[] = { 4, 1, 3, 5, 7, 2, 8, 6};

     public static final int IPImax = 8;

     public static final int EP[] = { 4, 1, 2, 3, 2, 3, 4, 1};

     public static final int EPmax = 4;

     public static final int S0[][] = {{ 1, 0, 3, 2},{ 3, 2, 1, 0},{ 0, 2, 1,

                                                              3},{ 3, 1, 3, 2}};

     public static final int S1[][] = {{ 0, 1, 2, 3},{ 2, 0, 1, 3},{ 3, 0, 1,

                                                          2},{ 2, 1, 0, 3}};

    public static int permute( int x, int p[], int pmax)

    {

      int y = 0;

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

       {

         y <<= 1;

         y |= (x >> (pmax - p[i])) & 1;

     }

      return y;

    }

    public static int F( int R, int K)

    {

       int t = permute( R, EP, EPmax) ^ K;

       int t0 = (t >> 4) & 0xF;

       int t1 = t & 0xF;

       t0 = S0[ ((t0 & 0x8) >> 2) | (t0 & 1) ][ (t0 >> 1) & 0x3 ];

       t1 = S1[ ((t1 & 0x8) >> 2) | (t1 & 1) ][ (t1 >> 1) & 0x3 ];

        t = permute( (t0 << 2) | t1, P4, P4max);

      return t;

  }

  public static int fK( int m, int K)

     {

         int L = (m >> 4) & 0xF;

         int R = m & 0xF;

         return ((L ^ F(R,K)) << 4) | R;

     }

  public static int SW( int x)

  {

   return ((x & 0xF) << 4) | ((x >> 4) & 0xF);

  }

   public byte encrypt( int m)

    {

      System.out.println("\nEncryption Process Starts........\n\n");

       m = permute( m, IP, IPmax);

      System.out.print("\nAfter Permutation : ");

       printData( m, 8);

       m = fK( m, K1);

       System.out.print("\nbefore Swap : ");

       printData( m, 8);

       m = SW( m);

       System.out.print("\nAfter Swap : ");

       printData( m, 8);

       m = fK( m, K2);

       System.out.print("\nbefore IP inverse : ");

       printData( m, 8);

       m = permute( m, IPI, IPImax);

       return (byte) m;

     }

     public byte decrypt( int m)

     {

       System.out.println("\nDecryption Process Starts........\n\n");

       printData( m, 8);

       m = permute( m, IP, IPmax);

       System.out.print("\nAfter Permutation : ");

       printData( m, 8);

       m = fK( m, K2);

       System.out.print("\nbefore Swap : ");

       printData( m, 8);

       m = SW( m);

       System.out.print("\nAfter Swap : ");

       printData( m, 8);

       m = fK( m, K1);

       System.out.print("\nBefore Extraction Permutation : ");

       printData( m, 4);

       m = permute( m, IPI, IPImax);

       System.out.print("\nAfter Extraction Permutation : ");

       printData( m, 8);

       return (byte) m;

     }

     public static void printData( int x, int n)

      {

        int mask = 1 << (n-1);

        while( mask > 0)

        {

        System.out.print( ((x & mask) == 0) ? '0' : '1');

        mask >>= 1;

        }

     }

    public SDES( int K)

   {

       K = permute( K, P10, P10max);

       int t1 = (K >> 5) & 0x1F;

       int t2 = K & 0x1F;

       t1 = ((t1 & 0xF) << 1) | ((t1 & 0x10) >> 4);

       t2 = ((t2 & 0xF) << 1) | ((t2 & 0x10) >> 4);

       K1 = permute( (t1 << 5)| t2, P8, P8max);

       t1 = ((t1 & 0x7) << 2) | ((t1 & 0x18) >> 3);

       t2 = ((t2 & 0x7) << 2) | ((t2 & 0x18) >> 3);

       K2 = permute( (t1 << 5)| t2, P8, P8max);

     }

   }

// Main operations

   public class simpledes

   {

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

     {

      DataInputStream inp=new DataInputStream(System.in);

      System.out.println("Enter the 10 Bit Key :");

       int K = Integer.parseInt(inp.readLine(),2);

       SDES A = new SDES( K);

       System.out.println("Enter the 8 Bit message To be Encrypt  : ");

       int m = Integer.parseInt(inp.readLine(),2);

       System.out.print("\nKey K1: ");

       SDES.printData( A.K1, 8);

       System.out.print("\nKey K2: ");

       SDES.printData( A.K2, 8);

       m = A.encrypt( m);

       System.out.print("\nEncrypted Message: ");

       SDES.printData( m, 8);

       m = A.decrypt( m);

       System.out.print("\nDecrypted Message: ");

       SDES.printData( m, 8);

       System.out.println(" ");

     }}

**DES**

class Main {

             private static class des {

                             // Initial Permutation Table

                             int[] IP

                                             = { 58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44,

                                                             36, 28, 20, 12, 4, 62, 54, 46, 38, 30, 22,

                                                             14, 6, 64, 56, 48, 40, 32, 24, 16, 8, 57,

                                                             49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35,

                                                             27, 19, 11, 3, 61, 53, 45, 37, 29, 21, 13,

                                                             5, 63, 55, 47, 39, 31, 23, 15, 7 };

                             // Inverse Initial Permutation Table

                             int[] IP1

                                             = { 40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47,

                                                             15, 55, 23, 63, 31, 38, 6, 46, 14, 54, 22,

                                                             62, 30, 37, 5, 45, 13, 53, 21, 61, 29, 36,

                                                             4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11,

                                                         51, 19, 59, 27, 34, 2, 42, 10, 50, 18, 58,

                                                             26, 33, 1, 41, 9, 49, 17, 57, 25 };

                             // first key-hePermutation Table

                             int[] PC1

                                             = { 57, 49, 41, 33, 25, 17, 9, 1, 58, 50,

                                                             42, 34, 26, 18, 10, 2, 59, 51, 43, 35,

                                                             27, 19, 11, 3, 60, 52, 44, 36, 63, 55,

                                                             47, 39, 31, 23, 15, 7, 62, 54, 46, 38,

                                                             30, 22, 14, 6, 61, 53, 45, 37, 29, 21,

                                                             13, 5, 28, 20, 12, 4 };

                             // second key-Permutation Table

                             int[] PC2

                                             = { 14, 17, 11, 24, 1, 5, 3, 28, 15, 6,

                                                             21, 10, 23, 19, 12, 4, 26, 8, 16, 7,

                                                         27, 20, 13, 2, 41, 52, 31, 37, 47, 55,

                                                             30, 40, 51, 45, 33, 48, 44, 49, 39, 56,

                                                             34, 53, 46, 42, 50, 36, 29, 32 };

                             // Expansion D-box Table

                             int[] EP = { 32, 1, 2, 3, 4, 5, 4, 5, 6, 7,

                                                                             8, 9, 8, 9, 10, 11, 12, 13, 12, 13,

                                                                             14, 15, 16, 17, 16, 17, 18, 19, 20, 21,

                                                                             20, 21, 22, 23, 24, 25, 24, 25, 26, 27,

                                                                             28, 29, 28, 29, 30, 31, 32, 1 };

                             // Straight Permutation Table

                             int[] P

                                             = { 16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23,

                                                             26, 5, 18, 31, 10, 2, 8, 24, 14, 32, 27,

                                                             3, 9, 19, 13, 30, 6, 22, 11, 4, 25 };

                             // S-box Table

                             int[][][] sbox

                                             = { { { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6,

                                                                             12, 5, 9, 0, 7 },

                                                             { 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12,

                                                                             11, 9, 5, 3, 8 },

                                                             { 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7,

                                                                             3, 10, 5, 0 },

                                                             { 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14,

                                                                             10, 0, 6, 13 } },

                                                             { { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13,

                                                                             12, 0, 5, 10 },

                                                             { 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10,

                                                                             6, 9, 11, 5 },

                                                             { 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6,

                                                                             9, 3, 2, 15 },

                                                             { 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12,

                                                                             0, 5, 14, 9 } },

                                                             { { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7,

                                                                             11, 4, 2, 8 },

                                                             { 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14,

                                                                             12, 11, 15, 1 },

                                                             { 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12,

                                                                             5, 10, 14, 7 },

                                                             { 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3,

                                                                             11, 5, 2, 12 } },

                                                             { { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5,

                                                                             11, 12, 4, 15 },

                                                             { 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12,

                                                                             1, 10, 14, 9 },

                                                             { 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3,

                                                                             14, 5, 2, 8, 4 },

                                                             { 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11,

                                                                             12, 7, 2, 14 } },

                                                             { { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15,

                                                                             13, 0, 14, 9 },

                                                             { 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15,

                                                                         10, 3, 9, 8, 6 },

                                                             { 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5,

                                                                             6, 3, 0, 14 },

                                                             { 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9,

                                                                             10, 4, 5, 3 } },

                                                             { { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4,

                                                                             14, 7, 5, 11 },

                                                             { 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14,

                                                                             0, 11, 3, 8 },

                                                             { 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10,

                                                                             1, 13, 11, 6 },

                                                             { 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7,

                                                                             6, 0, 8, 13 } },

                                                             { { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7,

                                                                             5, 10, 6, 1 },

                                                             { 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12,

                                                                             2, 15, 8, 6 },

                                                             { 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6,

                                                                             8, 0, 5, 9, 2 },

                                                             { 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15,

                                                                             14, 2, 3, 12 } },

                                                             { { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14,

                                                                             5, 0, 12, 7 },

                                                             { 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11,

                                                                             0, 14, 9, 2 },

                                                             { 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13,

                                                                             15, 3, 5, 8 },

                                                             { 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0,

                                                                             3, 5, 6, 11 } } };

                             int[] shiftBits = { 1, 1, 2, 2, 2, 2, 2, 2,

                                                                                                             1, 2, 2, 2, 2, 2, 2, 1 };

                             // hexadecimal to binary conversion

                             String hextoBin(String input)

                             {

                                             int n = input.length() \* 4;

                                             input = Long.toBinaryString(

                                                             Long.parseUnsignedLong(input, 16));

                                             while (input.length() < n)

                                                             input = "0" + input;

                                             return input;

                             }

                             // binary to hexadecimal conversion

                             String binToHex(String input)

                             {

                                             int n = (int)input.length() / 4;

                                             input = Long.toHexString(

                                                             Long.parseUnsignedLong(input, 2));

                                             while (input.length() < n)

                                                             input = "0" + input;

                                             return input;

                             }

                             // per-mutate input hexadecimal

                             // according to specified sequence

                             String permutation(int[] sequence, String input)

                             {

                                             String output = "";

                                             input = hextoBin(input);

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

                                                             output += input.charAt(sequence[i] - 1);

                                             output = binToHex(output);

                                             return output;

                             }

                             // xor 2 hexadecimal strings

                             String xor(String a, String b)

                             {

                                             // hexadecimal to decimal(base 10)

                                             long t\_a = Long.parseUnsignedLong(a, 16);

                                             // hexadecimal to decimal(base 10)

                                             long t\_b = Long.parseUnsignedLong(b, 16);

                                             // xor

                                             t\_a = t\_a ^ t\_b;

                                             // decimal to hexadecimal

                                             a = Long.toHexString(t\_a);

                                             // prepend 0's to maintain length

                                             while (a.length() < b.length())

                                                             a = "0" + a;

                                             return a;

                             }

                             // left Circular Shifting bits

                             String leftCircularShift(String input, int numBits)

                             {

                                             int n = input.length() \* 4;

                                             int perm[] = new int[n];

                                             for (int i = 0; i < n - 1; i++)

                                                             perm[i] = (i + 2);

                                             perm[n - 1] = 1;

                                             while (numBits-- > 0)

                                                             input = permutation(perm, input);

                                             return input;

                             }

                             // preparing 16 keys for 16 rounds

                             String[] getKeys(String key)

                             {

                                             String keys[] = new String[16];

                                             // first key permutation

                                             key = permutation(PC1, key);

                                             for (int i = 0; i < 16; i++) {

                                                             key = leftCircularShift(key.substring(0, 7),

                                                                                                                                                             shiftBits[i])

                                                                             + leftCircularShift(

                                                                                             key.substring(7, 14),

                                                                                             shiftBits[i]);

                                                             // second key permutation

                                                             keys[i] = permutation(PC2, key);

                                             }

                                             return keys;

                             }

                             // s-box lookup

                             String sBox(String input)

                             {

                                             String output = "";

                                             input = hextoBin(input);

                                             for (int i = 0; i < 48; i += 6) {

                                                             String temp = input.substring(i, i + 6);

                                                             int num = i / 6;

                                                             int row = Integer.parseInt(

                                                                             temp.charAt(0) + "" + temp.charAt(5),

                                                                             2);

                                                             int col = Integer.parseInt(

                                                                             temp.substring(1, 5), 2);

                                                             output += Integer.toHexString(

                                                                             sbox[num][row][col]);

                                             }

                                             return output;

                             }

                             String round(String input, String key, int num)

                             {

                                             // fk

                                             String left = input.substring(0, 8);

                                             String temp = input.substring(8, 16);

                                             String right = temp;

                                             // Expansion permutation

                                             temp = permutation(EP, temp);

                                             // xor temp and round key

                                             temp = xor(temp, key);

                                             // lookup in s-box table

                                             temp = sBox(temp);

                                             // Straight D-box

                                             temp = permutation(P, temp);

                                             // xor

                                             left = xor(left, temp);

                                             System.out.println("Round " + (num + 1) + " "

                                                                                                             + right.toUpperCase() + " "

                                                                                                         + left.toUpperCase() + " "

                                                                                                             + key.toUpperCase());

                                             // swapper

                                             return right + left;

                             }

                             String encrypt(String plainText, String key)

                             {

                                             int i;

                                             // get round keys

                                             String keys[] = getKeys(key);

                                             // initial permutation

                                             plainText = permutation(IP, plainText);

                                             System.out.println("After initial permutation: "

                                                                                                             + plainText.toUpperCase());

                                             System.out.println(

                                                             "After splitting: L0="

                                                             + plainText.substring(0, 8).toUpperCase()

                                                             + " R0="

                                                             + plainText.substring(8, 16).toUpperCase()

                                                             + "\n");

                                             // 16 rounds

                                             for (i = 0; i < 16; i++) {

                                                             plainText = round(plainText, keys[i], i);

                                             }

                                             // 32-bit swap

                                             plainText = plainText.substring(8, 16)

                                                                                             + plainText.substring(0, 8);

                                             // final permutation

                                             plainText = permutation(IP1, plainText);

                                             return plainText;

                             }

                             String decrypt(String plainText, String key)

                             {

                                             int i;

                                             // get round keys

                                             String keys[] = getKeys(key);

                                             // initial permutation

                                             plainText = permutation(IP, plainText);

                                             System.out.println("After initial permutation: "

                                                                                                             + plainText.toUpperCase());

                                             System.out.println(

                                                             "After splitting: L0="

                                                             + plainText.substring(0, 8).toUpperCase()

                                                             + " R0="

                                                             + plainText.substring(8, 16).toUpperCase()

                                                             + "\n");

                                             // 16-rounds

                                             for (i = 15; i > -1; i--) {

                                                             plainText

                                                                             = round(plainText, keys[i], 15 - i);

                                             }

                                             // 32-bit swap

                                             plainText = plainText.substring(8, 16)

                                                                                             + plainText.substring(0, 8);

                                             plainText = permutation(IP1, plainText);

                                             return plainText;

                             }

             }

             // Driver code

             public static void main(String args[])

             {

                             String text = "123456ABCD132536";

                             String key = "AABB09182736CCDD";

                             des cipher = new des();

                             System.out.println("Encryption:\n");

                             text = cipher.encrypt(text, key);

                             System.out.println(

                                             "\nCipher Text: " + text.toUpperCase() + "\n");

                             System.out.println("Decryption\n");

                             text = cipher.decrypt(text, key);

                             System.out.println("\nPlain Text: "+ text.toUpperCase());

             }