Name- Prathamesh Baban Wagadare

Roll No-BCOA49

Assignment No-1

package com.css.ghrcem;

import java.util.Scanner;

public class CaesarCipher {

public static StringBuffer encrypt(String text, int s)

{

StringBuffer result= new StringBuffer();

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

{

if (Character.*isUpperCase*(text.charAt(i)))

{

char ch = (char)(((int)text.charAt(i) + s - 65)%26 + 65);

result.append(ch);

}

else

{

char ch = (char)(((int)text.charAt(i) + s - 97)%26 + 97);

result.append(ch);

}

}

return result;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***) ;

String text=sc.nextLine();

int s = 4;

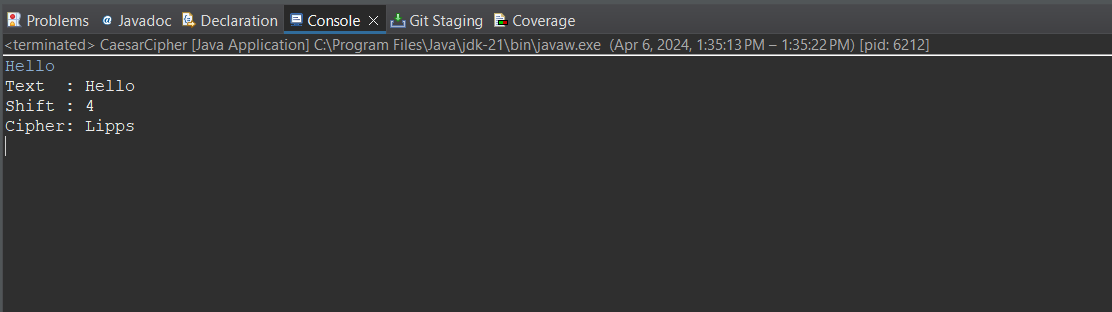
System.***out***.println("Text : " + text);

System.***out***.println("Shift : " + s);

System.***out***.println("Cipher: " + *encrypt*(text, s));

sc.close();

}}



Name- Prathamesh Baban Wagadare

Roll No-BCOA49

Assignment No-2

package com.css.ghrcem;

import java.util.Arrays;

import java.util.Scanner;

public class Playfair {

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("[^A-Z]", "");

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;

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;

}

}

}

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;

}}}

char decryptedCh1, decryptedCh2;

if (row1 == row2) {

decryptedCh1 = keySquare[row1][(col1 + 4) % 5];

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

} else if (col1 == col2) {

decryptedCh1 = keySquare[(row1 + 4) % 5][col1];

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

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);

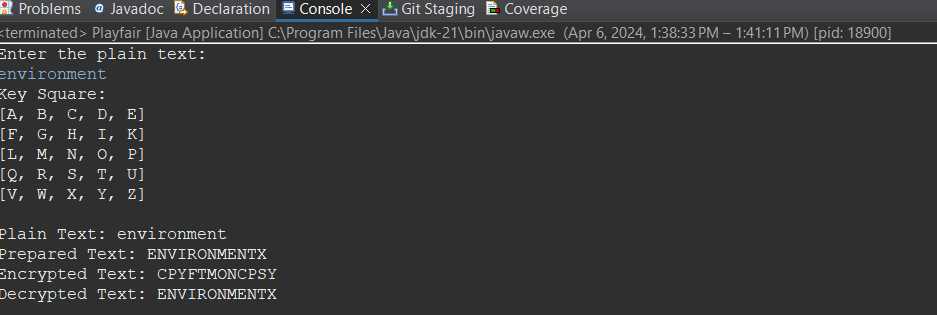
System.out.println("Prepared Text: " + preparedText);

System.out.println("Encrypted Text: " + encryptedText);

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

scan.close(); // Close the scanner

}}



Name- Prathamesh Baban Wagadare

Roll No-BCOA49

Assignment No-3

package com.css.ghrcem;

//Java program to illustrate Rail Fence Cipher

//Encryption and Decryption

import java.util.Arrays;

class RailFence {

public static String encryptRailFence(String text,int key)

{

char[][] rail = new char[key][text.length()];

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++) {

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

dirDown = !dirDown;

rail[row][col++] = text.charAt(i);

if (dirDown)

row++;

else

row--;

}

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();

}

public static String decryptRailFence(String cipher,int key)

{

char[][] rail = new char[key][cipher.length()];

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

Arrays.fill(rail[i], '\n');

boolean dirDown = true;

int 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;

// place the marker

rail[row][col++] = '\*';

if (dirDown)

row++;

else

row--;

}

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;

// place the marker

if (rail[row][col] != '\*')

result.append(rail[row][col++]);

if (dirDown)

row++;

else

row--;

}

return result.toString();

}

public static void main(String[] args)

{ 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));

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

System.out.println(

decryptRailFence("atc toctaka ne", 2));

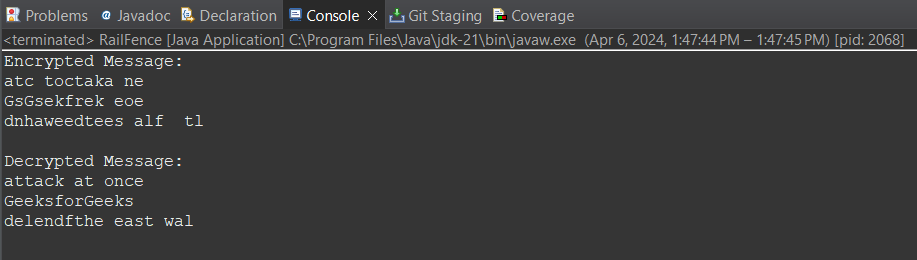
System.out.println(

decryptRailFence("GsGsekfrek eoe", 3));

System.out.println(

decryptRailFence("dnhaweedtees alf tl", 3));

}}

}} 

Name- Prathamesh Baban Wagadare

Roll No-BCOA49

Assignment No-4

package com.css.ghrcem;

import java.util.\*;

public class ColumnarTranspositionCipher {

// Key for Columnar Transposition

static final String ***key*** = "ZEBRAS";

static Map<Character, Integer> *keyMap* = new HashMap<>();

static void setPermutationOrder() {

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

*keyMap*.put(***key***.charAt(i), i);

}

}

static String encryptMessage(String msg) {

int row, col;

StringBuilder cipher = new StringBuilder();

/\* Calculate the number of columns in the matrix \*/

col = ***key***.length();

/\* Calculate the maximum number of rows in the matrix \*/

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

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

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

for (int j = 0; j < col; ) {

if (k < msg.length()) {

char ch = msg.charAt(k);

if (Character.*isLetter*(ch) || ch == ' ') {

matrix[i][j] = ch;

j++;

}

k++;

} else {

matrix[i][j] = '\_';

j++;

}}}

for (Map.Entry<Character, Integer> entry : *keyMap*.entrySet()) {

int columnIndex = entry.getValue();

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

if (Character.*isLetter*(matrix[i][columnIndex]) || matrix[i][columnIndex] == ' ' || matrix[i][columnIndex] == '\_') {

cipher.append(matrix[i][columnIndex]);

}}}

return cipher.toString();

}

// Decryption

static String decryptMessage(String cipher) {

int col = ***key***.length();

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

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

/\* Add characters into the matrix column-wise \*/

int k = 0;

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

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

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

k++;

}}

int index = 0;

for (Map.Entry<Character, Integer> entry : *keyMap*.entrySet()) {

entry.setValue(index++);

}

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

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

int columnIndex = *keyMap*.get(***key***.charAt(l));

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

decCipher[i][l] = cipherMat[i][columnIndex];

}

}

StringBuilder msg = new StringBuilder();

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

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

if (decCipher[i][j] != '\_') {

msg.append(decCipher[i][j]);

}}}

return msg.toString();

}

public static void main(String[] args) {

String msg = "SECRET MESSAGE";

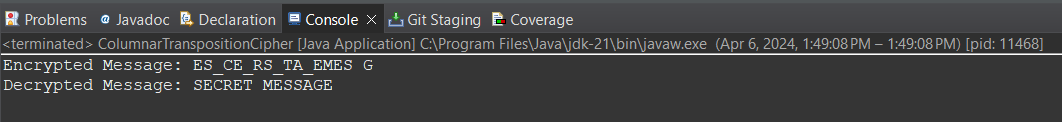
*setPermutationOrder*();

String cipher = *encryptMessage*(msg);

System.***out***.println("Encrypted Message: " + cipher);

System.***out***.println("Decrypted Message: " + *decryptMessage*(cipher));

}}



Name-Prathamesh Baban Wagadare

Roll No-BCOA49

Assignment No-5

package com.css.ghrcem;

import java.util.Random;

import java.util.Scanner;

public class OneTimePad {

// 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);

System.out.println("Enter a string to encrypt:");

String plaintext = scan.nextLine().toUpperCase();

// Generate random key

String key = generateRandomKey(plaintext.length());

// Encrypt and print

String ciphertext = encrypt(plaintext, key);

System.out.println("Plaintext: " + plaintext);

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

System.out.println("Ciphertext: " + ciphertext);

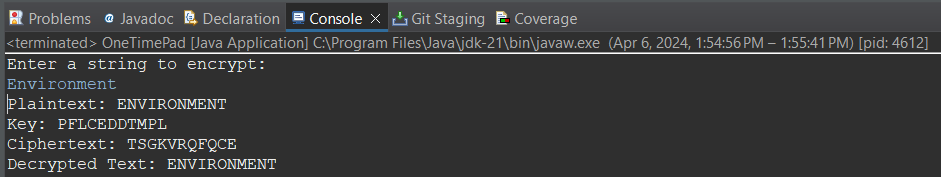
// Decrypt and print

String decryptedText = decrypt(ciphertext, key);

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

}

}



Name- Prathamesh Baban Wagadare

Roll No-BCOA49

Assignment No-6

package com.css.ghrcem;

import java.util.Scanner;

public class ExtendedEuclideanAlgorithm {

// Function to perform the extended Euclidean algorithm

public static int[] extendedEuclidean(int a, int b) {

if (b == 0) {

return new int[]{a, 1, 0};

}

int[] values = *extendedEuclidean*(b, a % b);

int gcd = values[0];

int s = values[2];

int t = values[1] - (a / b) \* values[2];

return new int[]{gcd, s, t};

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

System.***out***.println("Extended Euclidean Algorithm");

System.***out***.print("Enter the first number (a): ");

int a = scanner.nextInt();

System.***out***.print("Enter the second number (b): ");

int b = scanner.nextInt();

scanner.close();

int[] values = *extendedEuclidean*(a, b);

int gcd = values[0];

int s = values[1];

int t = values[2];

System.***out***.println("GCD of " + a + " and " + b + " is: " + gcd);

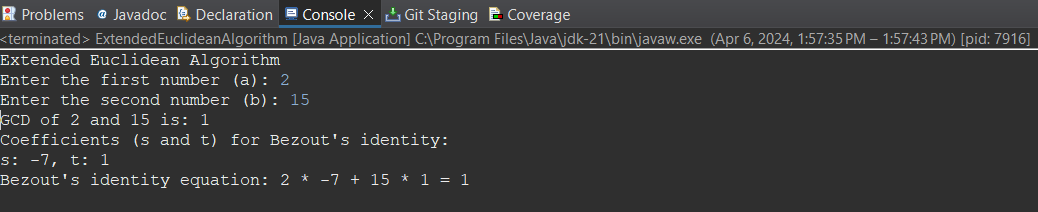
System.***out***.println("Coefficients (s and t) for Bezout's identity:");

System.***out***.println("s: " + s + ", t: " + t);

System.***out***.println("Bezout's identity equation: " + a + " \* " + s + " + " + b + " \* " + t + " = " + gcd);

}

}



Name- Prathamesh Baban Wagadare

Roll No-BCOA49

Assignment No-7

package com.css.ghrcem;

import java.io.\*;

class RSA {

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

int q, p, n, pn, publickey = 0, d = 0, msg;

double cipher, ptext;

int check, check1;

BufferedReader in = new BufferedReader(new InputStreamReader(System.***in***));

System.***out***.println("ENTER NO");

p = Integer.*parseInt*(in.readLine());

q = Integer.*parseInt*(in.readLine());

check = *prime*(p);

check1 = *prime*(q);

if (check != 1 || check1 != 1) {

System.*exit*(0);

}

n = p \* q;

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

for (int e = 2; e < pn; e++) {

if (*gcd*(e, pn) == 1) {

publickey = e;

System.***out***.println("PUBLIC KEY :" + e);

break;

}

}

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

d = i;

if (((d \* publickey) % pn) == 1)

break;

}

System.***out***.println("PRIVATE KEY :" + d);

System.***out***.println("ENTER MESSAGE ");

msg = Integer.*parseInt*(in.readLine());

cipher = Math.*pow*(msg, publickey);

cipher = cipher % n;

System.***out***.println("ENCRYPTED :" + (int) cipher);

ptext = Math.*pow*(cipher, d);

ptext = ptext % n;

System.***out***.println("DECRYPTED :" + (int) ptext);

}

static int prime(int a) {

int flag = 0;

for (int i = 2; i <= a / 2; i++) {

if (a % i == 0) {

System.***out***.println(a + " is not a Prime Number");

flag = 1;

return 0;

}

}

if (flag == 0)

return 1;

return 1;

}

static int gcd(int number1, int number2) {

if (number2 == 0) {

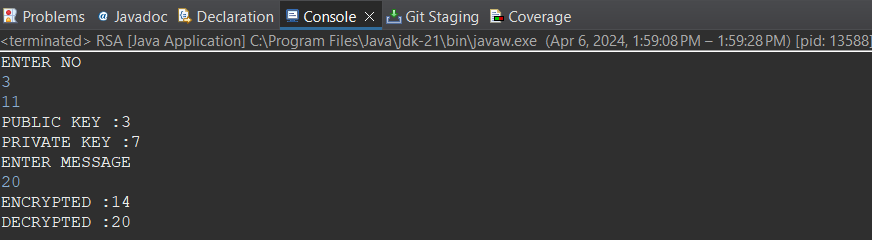
return number1;

}

return *gcd*(number2, number1 % number2);

}

}



Name- Prathamesh Baban Wagadare

Roll No-BCOA49

Assignment No-8

package com.css.ghrcem;

import java.security.\*;

import java.util.Base64;

public class DigitalSignature {

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

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

keyPairGenerator.initialize(2048);

KeyPair keyPair = keyPairGenerator.generateKeyPair();

PrivateKey privateKey = keyPair.getPrivate();

String message = "This is a message to be signed.";

Signature signature = Signature.getInstance("SHA256withRSA");

signature.initSign(privateKey);

signature.update(message.getBytes());

byte[] signatureBytes = signature.sign();

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);

}

}

