**EX1A – CAESAR CIPHER**

**PROGRAM:**

**public** **class** Ex01a\_CaesarCipher {

**public** **static** String crypt(String input, **int** key, **boolean** encrypt) {

StringBuilder cipher = **new** StringBuilder("");

**for** (**char** i : input.toCharArray()) {

cipher.append((**char**) (((i - 'a' + (encrypt ? 1 : -1) \* key) % 26 + 26) % 26 + 'a'));

}

**return** cipher.toString();

}

**public** **static** **void** main(String[] args) {

String plain = "hello";

**int** key = 20;

System.***out***.println("PLAIN TEXT : " + plain);

System.***out***.println("KEY TEXT : " + key);

System.***out***.println("CIPHER TEXT : "

+ *crypt*(plain, key, **true**).toUpperCase());

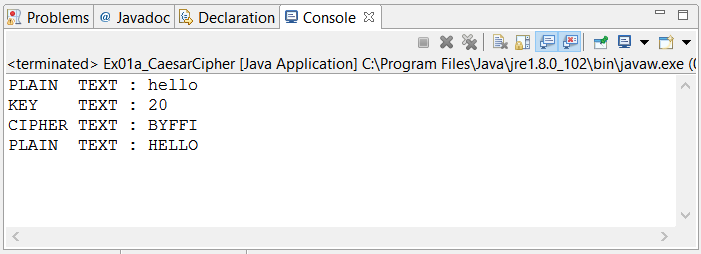
System.***out***.println("PLAIN TEXT : "

+ *crypt*(*crypt*(plain, key, **true**), key, **false**).toUpperCase());

}

}

**OUTPUT:**



**RESULT:**

**THE CAESAR CIPHER WAS SUCCESSFULLY CREATED**

**EX1B – PLAYFAIR CIPHER**

**PROGRAM:**

**public** **class** Ex01b\_PlayFair {

**public** **static** **int**[][] processKey(String key) {

**int**[][] keyMat = **new** **int**[26][2];

**int** l = 0;

**for** (**char** i : (key + "abcdefghiklmnopqrstuvwxyz").toCharArray()) {

**if** (key.indexOf(i + "") < 0 || l < key.length()) {

keyMat[i - 'a'][0] = l / 5;

keyMat[i - 'a'][1] = l++ % 5;

**if** (i == 'i') {

keyMat[i - 'a' + 1][0] = l / 5;

keyMat[i - 'a' + 1][1] = l % 5;

}

}

}

**return** keyMat;

}

**public** **static** String crypt(String inputText, String key, **boolean** encrypt) {

**int**[][] keyMat = *processKey*(key);

**char**[][] indMat = **new** **char**[5][5];

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

indMat[keyMat[i][0]][keyMat[i][1]] = (**char**) ('a' + i);

}

String cipherText = "";

**for** (**int** i = 0; i < inputText.length(); i += 2) {

**char** first = inputText.charAt(i);

**char** second = i + 1 == inputText.length()

|| first == inputText.charAt(i + 1) ? 'x' : inputText

.charAt(i + 1);

**int** fRow = keyMat[first - 'a'][0];

**int** fCol = keyMat[first - 'a'][1];

**int** sRow = keyMat[second - 'a'][0];

**int** sCol = keyMat[second - 'a'][1];

**if** (fRow == sRow) {

fCol = ((fCol + (encrypt ? 1 : -1)) % 5 + 5) % 5;

sCol = ((sCol + (encrypt ? 1 : -1)) % 5 + 5) % 5;

} **else** **if** (fCol == sCol) {

fRow = ((fRow + (encrypt ? 1 : -1)) % 5 + 5) % 5;

sRow = ((sRow + (encrypt ? 1 : -1)) % 5 + 5) % 5;

} **else** {

**int** tCol = fCol;

fCol = sCol;

sCol = tCol;

}

cipherText += (indMat[fRow][fCol]) + "" + (indMat[sRow][sCol]);

}

**return** cipherText;

}

**public** **static** **void** main(String[] args) {

String plain = "karthik";

String key = "monarchy";

System.***out***.println("PLAIN TEXT : " + plain);

System.***out***.println("KEY TEXT : " + key);

System.***out***.println("CIPHER TEXT : "

+ *crypt*(plain, key, **true**).toUpperCase());

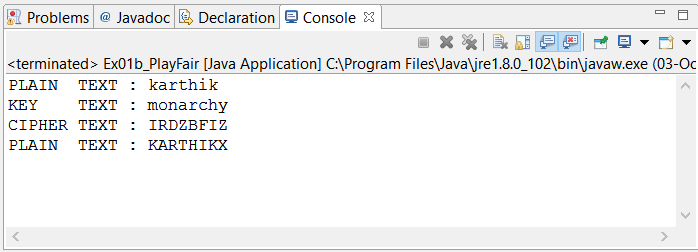
System.***out***.println("PLAIN TEXT : "

+ *crypt*(*crypt*(plain, key, **true**), key, **false**).toUpperCase());

}

}

**OUTPUT:**



**RESULT:**

**THE PLAY FAIR CIPHER ALGORITHM WAS IMPLEMENTED AND TESTED.**

**EX02A – VIGENERE CIPHER**

**PROGRAM:**

**public** **class** Ex02b\_Vigenere {

**public** **static** String cryptic(String input, String key, **boolean** encrypt) {

StringBuilder output = **new** StringBuilder("");

**int** j = 0;

**for** (**char** i : input.toCharArray()) {

output.append((**char**) (((i - 'a' + (encrypt ? key.charAt(j) - 'a'

: -key.charAt(j) + 'a')) % 26 + 26) % 26 + 'a'));

j = (j + 1) % key.length();

}

**return** output.toString();

}

**public** **static** **void** main(String[] args) {

String plain = "karthik", key = "hello";

System.***out***.println("PLAIN TEXT : " + plain);

System.***out***.println("KEY TEXT : " + key);

System.***out***.println("CIPHER TEXT : " + *cryptic*(plain, key, **true**));

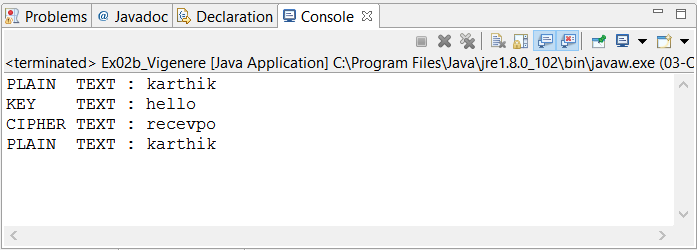
System.***out***.println("PLAIN TEXT : "

+ *cryptic*(*cryptic*(plain, key, **true**), key, **false**));

}

}

**OUTPUT:**



**RESULT:**

**THE VIGENERE CIPHER ALGORITHM WAS SUCCESSFULLY IMPLEMENTED AND TESTED.**

**EX03A – RAIL FENCE ALGORITHM**

**PROGRAM:**

**public** **class** Ex03a\_RailFence {

**public** **static** String crypt(String msg, **int** key, **boolean** encrypt) {

**char**[] res = **new** **char**[msg.length()];

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

**int** inc = 2 \* (key - i - 1);

// format to take chars is j....(j + inc)....(j + 2 \* (key - 1))

**for** (**int** j = i; j < msg.length(); j += 2 \* (key - 1)) {

res[encrypt ? k++ : j] = msg.charAt(encrypt ? j : k++);

**if** (i != key - 1 && i != 0 && (j + inc) < msg.length())

res[encrypt ? k++ : j + inc] = msg.charAt(encrypt ? j + inc

: k++);

}

}

**return** **new** String(res);

}

**public** **static** **void** main(String[] args) {

String plain = "karthikmam";

**int** key = 4;

System.***out***.println("PLAIN TEXT : " + plain);

System.***out***.println("KEY TEXT : " + key);

System.***out***.println("CIPHER TEXT : "

+ *crypt*(plain, key, **true**).toUpperCase());

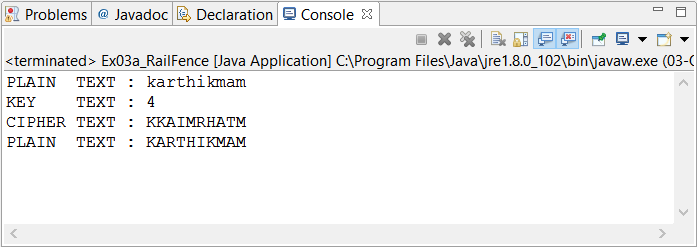
System.***out***.println("PLAIN TEXT : "

+ *crypt*(*crypt*(plain, key, **true**), key, **false**).toUpperCase());

}

}

**OUTPUT:**



**RESULT:**

**THE RAIL FENCE ALGORITHM WAS SUCCESSFULLY IMPLEMENTED AND TESTED.**

**EX03B – ROW COLUMN CIPHER**

**PROGRAM:**

**import** java.util.Arrays;

**public** **class** Ex03b\_RowColumn {

**public** **static** String crypt(String msg, **int**[] key, **boolean** encrypt) {

**char**[] res = **new** **char**[msg.length()];

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

**for** (**int** j = key[i]; j < msg.length(); j += key.length)

res[encrypt ? k++ : j] = msg.charAt(encrypt ? j : k++);

**return** **new** String(res);

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

String plain = "KARTHIKMAM";

**int**[] key = { 1, 2, 0 };

System.***out***.println("PLAIN TEXT : " + plain);

System.***out***.println("KEY TEXT : " + Arrays.*toString*(key));

System.***out***.println("CIPHER TEXT : " + *crypt*(plain, key, **true**));

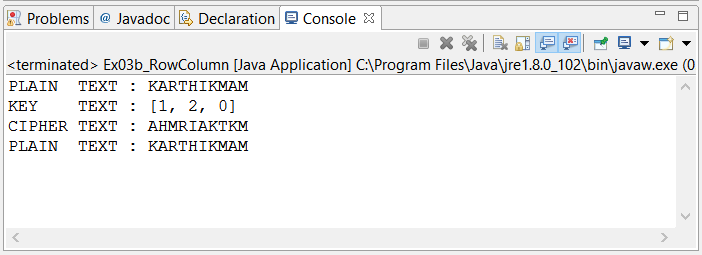
System.***out***.println("PLAIN TEXT : "

+ *crypt*(*crypt*(plain, key, **true**), key, **false**));

}

}

**OUTPUT:**



**RESULT:**

**THE ROW COLUMN CIPHER WAS SUCCESSFULLY IMPLEMENTED AND TESTED.**

**EX04 – DES**

**PROGRAM:**

**import** java.math.BigInteger;

**public** **class** Ex04\_DES {

**private** **static** **final** **long** ***GET\_32B*** = (1L << 32) - 1;

**private** **static** **final** **long** ***GET\_28B*** = (1L << 28) - 1;

**private** **static** **final** **long** ***GET\_56B*** = (1L << 56) - 1;

**private** **static** **final** **short**[] ***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 };

**private** **static** **final** **short**[] ***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 };

**private** **static** **final** **short**[] ***L\_ROT*** = { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };

**private** **static** **final** **short**[] ***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 };

**private** **static** **short**[] *IP\_1* = {

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 };

**private** **static** **final** **short**[] ***E*** = {

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 };

**private** **static** **long**[][] *S* = {

{ 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 } };

**private** **static** **short**[] *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 };

**private** **static** **long** mutate(**long** input, **short**[] table, **long** originalLength) {

**long** result = 0;

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

result = (result << 1) | (input >>> (originalLength - table[i]))

% 2;

// System.out.printf("%x \n", result);

}

**return** result;

}

**private** **long**[] keys = **new** **long**[16];

**public** Ex04\_DES(**long** key) {

**long** pKey = *mutate*(key, ***PC1***, 64) & ***GET\_56B***;

**long** c = pKey >>> 28;

**long** d = pKey & ***GET\_28B***;

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

c = ((c << ***L\_ROT***[i]) | (c >>> (28 - ***L\_ROT***[i]))) & ***GET\_28B***;

d = ((d << ***L\_ROT***[i]) | (d >>> (28 - ***L\_ROT***[i]))) & ***GET\_28B***;

keys[i] = *mutate*((c << 28) | d, ***PC2***, 56);

}

}

**public** **long** crypt(**long** msg, **boolean** encrypt) {

msg = *mutate*(msg, ***IP***, 64);

**long** l = msg >>> 32;

**long** r = msg & ***GET\_32B***;

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

**long** temp = r;

r = l ^ f(r, keys[encrypt ? i : 16 - i - 1]);

l = temp;

// System.out.printf("%16s %16s %16x \n", Long.toHexString(r),

// Long.toHexString(l), keys[encrypt ? i : 16 - i - 1]);

}

**return** *mutate*((r << 32) | l, *IP\_1*, 64);

}

**private** **long** f(**long** r, **long** key) {

r = *mutate*(r & ***GET\_32B***, ***E***, 32) ^ key;

**long** result = 0;

**for** (**int** i = 7; i >= 0; i--) {

**byte** box = (**byte**) (r & 0x3F);

r = r >>> 6;

**int** row = ((box >>> 5) << 1) | (box & 1);

**int** col = (box >>> 1) & 0xF;

result |= *S*[i][row \* 16 + col] << (28 - i \* 4);

}

**return** *mutate*(result, *P*, 32);

}

**public** **static** **void** main(String[] args) {

**long** plain = **new** BigInteger("Plain".getBytes()).longValue();

**long** key = **new** BigInteger("Hello".getBytes()).longValue();

Ex04\_DES x = **new** Ex04\_DES(key);

System.***out***.printf("PLAIN TEXT : %16s \n", **new** String(**new** BigInteger(

plain + "").toByteArray()));

System.***out***.printf("KEY TEXT : %16s \n", **new** String(**new** BigInteger(

key + "").toByteArray()));

System.***out***.printf("CIPHER TEXT : %16s \n",

Long.*toHexString*(x.crypt(plain, **true**)));

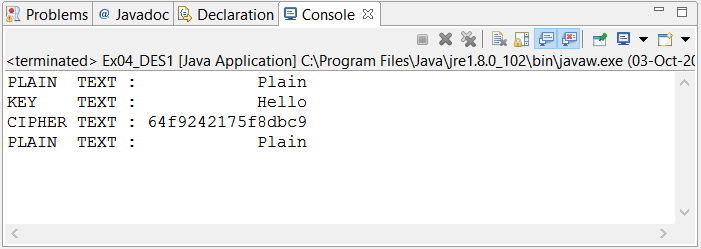
System.***out***.printf("PLAIN TEXT : %16s \n", **new** String(**new** BigInteger(""

+ x.crypt(x.crypt(plain, **true**), **false**)).toByteArray()));

}

}

**OUTPUT:**



**RESULT:**

**THE DES ALGORITHM WAS SUCCESSFULLY IMPLEMENTED AND TESTED.**

**EX05 – RSA**

**PROGRAM:**

**import** java.math.BigInteger;

**import** java.util.Random;

**import** javax.xml.bind.DatatypeConverter;

**public** **class** Ex05\_RSA {

**private** **static** **int** *bitLength* = 128;

**private** BigInteger n, e, d;

**public** Ex05\_RSA() {

Random rnd = **new** Random();

BigInteger p = BigInteger.*probablePrime*(*bitLength*, rnd);

BigInteger q = BigInteger.*probablePrime*(*bitLength*, rnd);

**this**.n = p.multiply(q);

BigInteger phi = p.subtract(BigInteger.***ONE***).multiply(

q.subtract(BigInteger.***ONE***));

**this**.e = BigInteger.*probablePrime*(*bitLength* / 2, rnd);

**while** (e.gcd(phi).compareTo(BigInteger.***ONE***) == 1

&& e.compareTo(phi) < 1) {

e.add(BigInteger.***ONE***);

}

**this**.d = e.modInverse(phi);

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

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

System.***out***.println("N : " + n);

System.***out***.println();

}

**public** **byte**[] crypt(**byte**[] input, **boolean** encrypt) {

**return** **new** BigInteger(input).modPow(encrypt ? e : d, n).toByteArray();

}

**public** **static** **void** main(String[] args) {

Ex05\_RSA rsa = **new** Ex05\_RSA();

String plain = "hello";

System.***out***.println("PLAIN TEXT : " + plain);

System.***out***.println("CIPHER TEXT : "

+ DatatypeConverter.*printHexBinary*(rsa.crypt(plain.getBytes(),

**true**)));

System.***out***.println("PLAIN TEXT : "

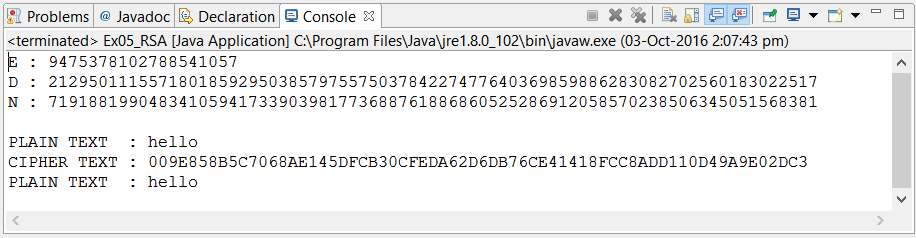
+ **new** String(

rsa.crypt(rsa.crypt(plain.getBytes(), **true**), **false**)));

}

}

**OUTPUT:**



**RESULT:**

**THE RSA ALGORITHM WAS SUCCESSFULLY IMPLEMENTED.**

**EX06 – DIFFE HELLMAN KEY EXCHANGE ALGORITHM**

**PROGRAM:**

**import** java.math.BigInteger;

**import** java.util.ArrayList;

**import** java.util.Scanner;

**public** **class** Ex06\_DiffeHellman {

**public** **static** ArrayList<BigInteger> getPrimeFactors(BigInteger n) {

ArrayList<BigInteger> res = **new** ArrayList<BigInteger>();

**for** (BigInteger i = **new** BigInteger("2"); i.intValue() < Math.*sqrt*(n

.intValue()); i = i.add(BigInteger.***ONE***))

**if** (i.isProbablePrime(100) == **true** && n.mod(i).intValue() == 0)

res.add(i);

**return** res;

}

**public** **static** BigInteger primitiveRoot(BigInteger n) {

BigInteger phi = n.subtract(BigInteger.***ONE***);

ArrayList<BigInteger> primeFactors = *getPrimeFactors*(phi);

**for** (BigInteger i = **new** BigInteger("2"); i.intValue() < n.intValue(); i = i

.add(BigInteger.***ONE***)) {

**boolean** flag = **true**;

**for** (BigInteger j = BigInteger.***ZERO***; j.intValue() < primeFactors

.size(); j = j.add(BigInteger.***ONE***))

**if** (i.modPow(phi.divide(primeFactors.get(j.intValue())), n)

.longValue() == 1)

flag = **false**;

**if** (flag == **true**)

**return** i;

}

**return** BigInteger.***ZERO***;

}

**private** **static** Scanner *stdIn* = **new** Scanner(System.***in***);

**public** **static** **void** main(String[] args) {

System.***out***.print("PRIME NUMBER P : ");

BigInteger p = **new** BigInteger(*stdIn*.nextInt() + "");

BigInteger q = *primitiveRoot*(p);

System.***out***.println("PRIMITIVE ROOT Q : " + q);

System.***out***.println();

System.***out***.print("SECRET xA : ");

BigInteger xA = **new** BigInteger(*stdIn*.nextInt() + "");

BigInteger yA = q.modPow(xA, p);

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

System.***out***.println();

System.***out***.print("SECRET xB : ");

BigInteger xB = **new** BigInteger(*stdIn*.nextInt() + "");

BigInteger yB = q.modPow(xB, p);

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

System.***out***.println();

BigInteger sharedKeyA = yB.modPow(xA, p);

BigInteger sharedKeyB = yA.modPow(xB, p);

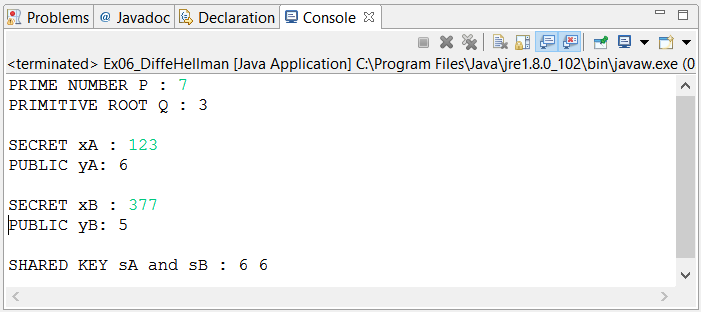
System.***out***.println("SHARED KEY sA and sB : " + sharedKeyA + " "

+ sharedKeyB);

}

}

**OUTPUT:**



**RESULT:**

**THE DH ALGORITHM WAS SUCCESSFULLY IMPLEMENTED.**

**EX07 – MD5 HASH ALGORITHM**

**PROGRAM:**

**import** java.util.Arrays;

**public** **class** Ex07\_MD5 {

**private** **static** **final** **int**[][] ***S*** = {

{ 7, 12, 17, 22 },

{ 5, 9, 14, 20 },

{ 4, 11, 16, 23 },

{ 6, 10, 15, 21 }

};

**private** **static** **final** **int**[] ***T***;

**static** {

***T*** = **new** **int**[64];

**for**(**int** i = 0; i < 64; i++)

***T***[i] = (**int**) (**long**) ((1L << 32) \* Math.*abs*(Math.*sin*(i + 1)));

}

**private** **static** **final** **int** F(**int** x, **int** y, **int** z) { **return** (x & y) | (~x & z); }

**private** **static** **final** **int** G(**int** x, **int** y, **int** z) { **return** (x & z) | (y & ~z); }

**private** **static** **final** **int** H(**int** x, **int** y, **int** z) { **return** (x ^ y ^ z); }

**private** **static** **final** **int** I(**int** x, **int** y, **int** z) { **return** y ^ (x | ~z); }

**private** **static** **final** **int** R(**int** n, **int** i) { **return** (n << i) | (n >>> (32 - i)); }

**public** **static** String digest(String msg) {

**int**[] words = **new** **int**[(**int**) (((**long**) msg.length() + (64 - msg.length() % 64)) / 4)];

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

words[i >>> 2] |= msg.charAt(i) << (24 - (i % 4) \* 8);

words[msg.length() >>> 2] |= 0x80 << (24 - (msg.length() % 4) \* 8);

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

words[i] = Integer.*reverseBytes*(words[i]);

words[words.length - 2] = msg.length() \* 8;

words[words.length - 1] = (**int**) ((msg.length() \* 8) / (1L << 32));

**int** a = Integer.*reverseBytes*(0x01234567);

**int** b = Integer.*reverseBytes*(0x89abcdef);

**int** c = Integer.*reverseBytes*(0xfedcba98);

**int** d = Integer.*reverseBytes*(0x76543210);

**for** (**int** i = 0; i < words.length / 16; i += 16) {

**int**[] word = Arrays.*copyOfRange*(words, i, i + 16);

**int** aa = a;

**int** bb = b;

**int** cc = c;

**int** dd = d;

**int** count = -1;

**for** (**int** j = 0, inc = -1; j < 4; j++) {

a = b + *R*((a + *F*(b, c, d) + word[inc = ((inc + 1) % 16)] + ***T***[count += 1] ), ***S***[0][0]);

d = a + *R*((d + *F*(a, b, c) + word[inc = ((inc + 1) % 16)] + ***T***[count += 1] ), ***S***[0][1]);

c = d + *R*((c + *F*(d, a, b) + word[inc = ((inc + 1) % 16)] + ***T***[count += 1] ), ***S***[0][2]);

b = c + *R*((b + *F*(c, d, a) + word[inc = ((inc + 1) % 16)] + ***T***[count += 1] ), ***S***[0][3]);

}

**for** (**int** j = 0, inc = -4; j < 4; j++) {

a = b + *R*((a + *G*(b, c, d) + word[inc = ((inc + 5) % 16)] + ***T***[count += 1] ), ***S***[1][0]);

d = a + *R*((d + *G*(a, b, c) + word[inc = ((inc + 5) % 16)] + ***T***[count += 1] ), ***S***[1][1]);

c = d + *R*((c + *G*(d, a, b) + word[inc = ((inc + 5) % 16)] + ***T***[count += 1] ), ***S***[1][2]);

b = c + *R*((b + *G*(c, d, a) + word[inc = ((inc + 5) % 16)] + ***T***[count += 1] ), ***S***[1][3]);

}

**for** (**int** j = 0, inc = 2; j < 4; j++) {

a = b + *R*((a + *H*(b, c, d) + word[inc = ((inc + 3) % 16)] + ***T***[count += 1] ), ***S***[2][0]);

d = a + *R*((d + *H*(a, b, c) + word[inc = ((inc + 3) % 16)] + ***T***[count += 1] ), ***S***[2][1]);

c = d + *R*((c + *H*(d, a, b) + word[inc = ((inc + 3) % 16)] + ***T***[count += 1] ), ***S***[2][2]);

b = c + *R*((b + *H*(c, d, a) + word[inc = ((inc + 3) % 16)] + ***T***[count += 1] ), ***S***[2][3]);

}

**for** (**int** j = 0, inc = -7; j < 4; j++) {

a = b + *R*((a + *I*(b, c, d) + word[inc = ((inc + 7) % 16)] + ***T***[count += 1] ), ***S***[3][0]);

d = a + *R*((d + *I*(a, b, c) + word[inc = ((inc + 7) % 16)] + ***T***[count += 1] ), ***S***[3][1]);

c = d + *R*((c + *I*(d, a, b) + word[inc = ((inc + 7) % 16)] + ***T***[count += 1] ), ***S***[3][2]);

b = c + *R*((b + *I*(c, d, a) + word[inc = ((inc + 7) % 16)] + ***T***[count += 1] ), ***S***[3][3]);

}

a = a + aa;

b = b + bb;

c = c + cc;

d = d + dd;

}

**return** String.*format*("%x%x%x%x",

Integer.*reverseBytes*(a),

Integer.*reverseBytes*(b),

Integer.*reverseBytes*(c),

Integer.*reverseBytes*(d));

}

**public** **static** **void** main(String[] args) {

String msg = "hello";

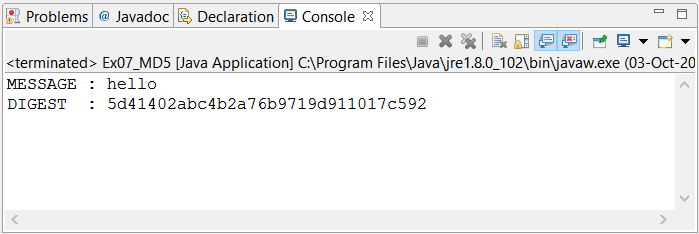
System.***out***.println("MESSAGE : " + msg);

System.***out***.println("DIGEST : " + *digest*(msg));

}

}

**OUTPUT:**



**RESULT:**

**THE MD5 HASH ALGORITHM WAS SUCCESSFULLY IMPLEMENTED.**

**EX08 – SHA1 ALGORITHM**

**PROGRAM:**

**public** **class** Ex08\_SHA1 {

**private** **static** **int** R(**int** n, **int** i) {

**return** (n << i) | (n >>> (32 - i));

}

**public** **static** String digest(String msg) {

**int**[] words = **new** **int**[(**int**) (((**long**) msg.length() + (64 - msg.length() % 64)) / 4)];

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

words[i >>> 2] |= msg.charAt(i) << (24 - (i % 4) \* 8);

words[msg.length() >>> 2] |= 0x80 << (24 - (msg.length() % 4) \* 8);

words[words.length - 1] = msg.length() \* 8;

**int**[] w = **new** **int**[80];

**int** h0 = Integer.*reverseBytes*(0x01234567);

**int** h1 = Integer.*reverseBytes*(0x89abcdef);

**int** h2 = Integer.*reverseBytes*(0xfedcba98);

**int** h3 = Integer.*reverseBytes*(0x76543210);

**int** h4 = Integer.*reverseBytes*(0xf0e1d2c3);

**for** (**int** i = 0; i < words.length; i += 16) {

**int** a = h0;

**int** b = h1;

**int** c = h2;

**int** d = h3;

**int** e = h4;

**for** (**int** j = 0; j < 80; j++) {

w[j] = (j < 16) ? words[i + j] : (*R*(w[j - 3]

^ w[j - 8] ^ w[j - 14] ^ w[j - 16], 1));

**int** t = *R*(a, 5) + e + w[j] +

( j < 20 ? (0x5a827999 + ((b & c) | ((~b) & d)))

: j < 40 ? (0x6ed9eba1 + (b ^ c ^ d))

: j < 60 ? (0x8f1bbcdc + ((b & c) | (b & d) | (c & d)))

: (0xca62c1d6 + (b ^ c ^ d)));

e = d;

d = c;

c = *R*(b, 30);

b = a;

a = t;

}

h0 += a;

h1 += b;

h2 += c;

h3 += d;

h4 += e;

}

**return** String.*format*("%x%x%x%x%x", h0, h1, h2, h3, h4);

}

**public** **static** **void** main(String args[]) {

String msg = "hello";

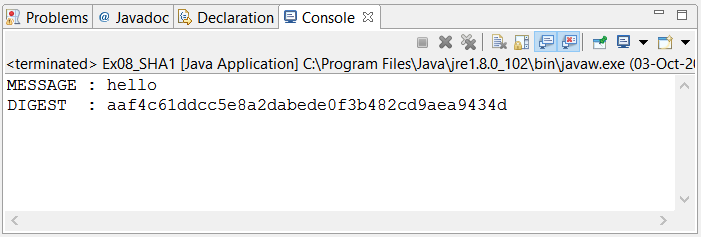
System.***out***.println("MESSAGE : " + msg);

System.***out***.println("DIGEST : " + *digest*(msg));

}

}

**OUTPUT:**



**RESULT:**

**THE SHA1 ALGORITHM WAS SUCCESSFULLY IMPLMENETED.**