Name :- Shivani. S
Reg NO :- 312217104151
Subcode :- IT8761
Subject :- Security lab.

Question :-

Develop a java program to implement the DES encryption. key in numeric / hexadecimal / binary will be given. Input plaintext message will be an english statement / phrase, encrypted message should be in hexadecimal.

Algorithm / Procedure :-

Input :- 64 bit blocks of plaintext & 64 bit key.

Output :- DES encrypted 64 bit ciphertext.

Algorithm :-

(1) The message is divided into 64 bit blocks.

(2) The block is then passed to initial permutation.

(2) After a permutation the block is passed to Round1, where it accepts 2 inputs

(1) 64 bit block
(2) 48 bit key.

The process happening in Rounds:-

(1) There are totally 16 rounds in DES algorithm.

(2) The 64 bit block is divided into 2, 32 bit blocks of left and right.

(3) In the right side (32bit block) first expansion happens and 32bit is converted into 48 bits.

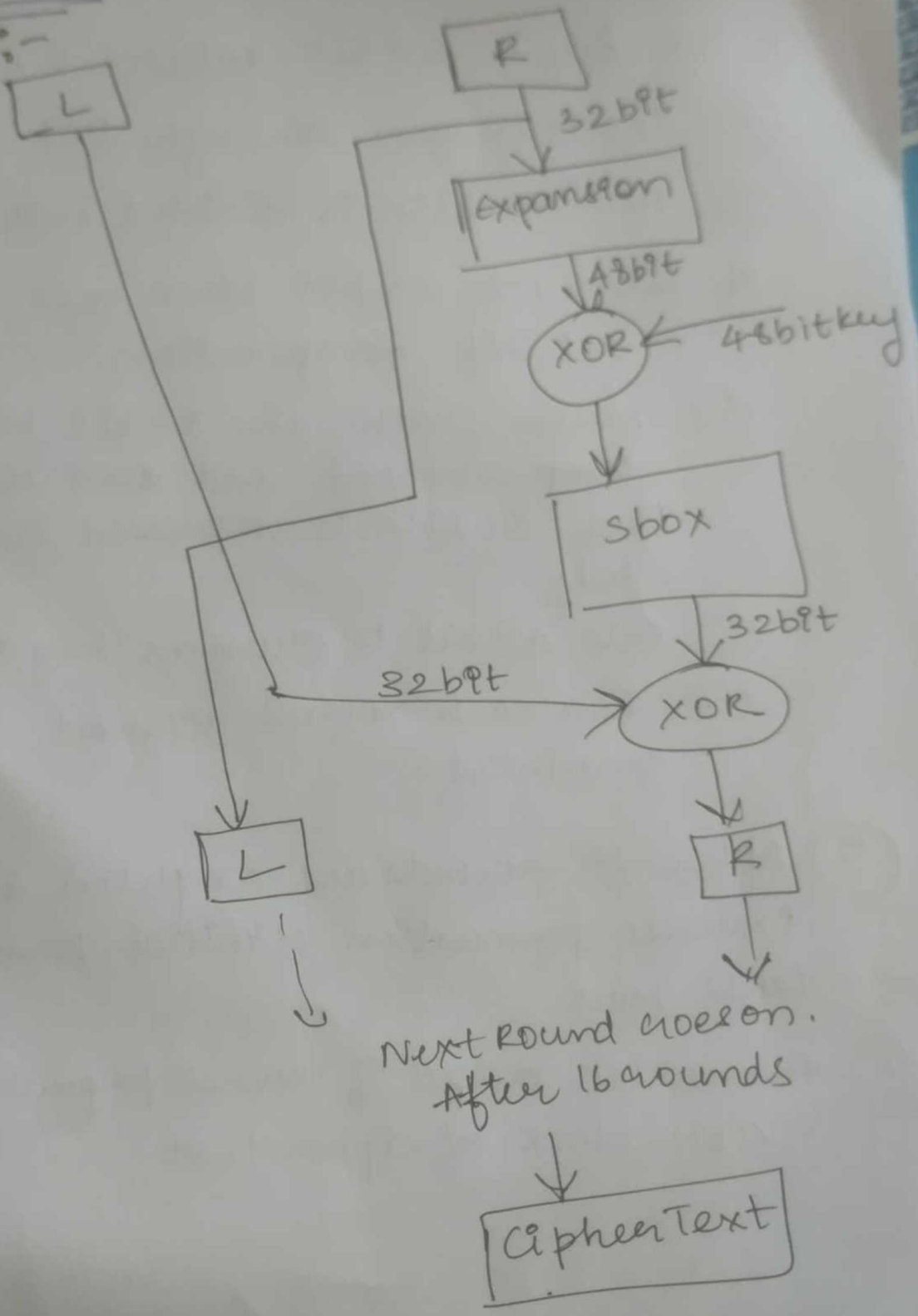
(4) This 48bit is XOR with 48 bit key

(5) Later this result is passed to S-box and a result of 32bit is generated

(6) This 32 bit is XOR with left side 32bit

flow chart :-

Rounds :-



(4) Key Generation :-

(1) The Key is 64 bit where 8 bit is parity
(2) The key goes through PCI.
(3) The 56 bits is divided into 2, 28 bits
(4) ~~The~~ each 28 bit block goes through left shift ~~op~~ operation.
(5) After both the 28 bit block are ~~comp~~ combined and sent to PC2 where 56 bits is compressed to ~~48~~ 48 bit key
(6) This 48 bit is given as key to rounds.
(7) For each round different keys are generated.

(5) After 16 rounds are completed and inverse permutation of initial permutation takes place

(6) ~~Hence~~ The output of inverse permutation is 64 bit block of cipher text.

NAME:SHIVANI S
REG NO:312217104151

CODE:

```
import java.util.*;


class DES {
  private static class DES1 {

    // 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 };

    // Permuted choice 1 Table 52/7

    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 };

// Permuted choice 2 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 } }
};

// shift bits
int[] shiftBits = { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };


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;
}


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;
}


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;
}


String xor(String a, String b)
{

  long t_a = Long.parseUnsignedLong(a, 16);

  long t_b = Long.parseUnsignedLong(b, 16);

  t_a = t_a ^ t_b;

  a = Long.toHexString(t_a);
```

```
      while (a.length() < b.length())
        a = "0" + a;
      return a;
    }


    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;
    }


    String[] getKeys(String key)
    {
                    String keys[] = new String[16];

                    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]);

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


    String sBox(String input)
```

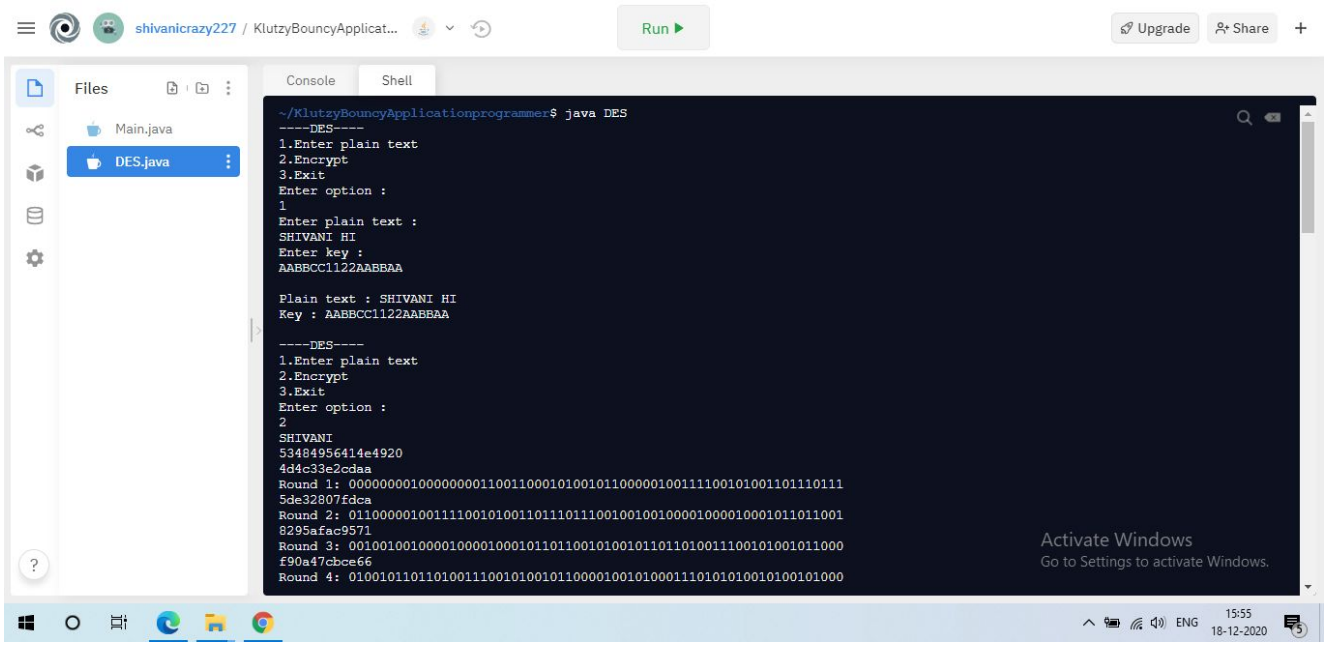
```
{
  String output = "";
  input = hextoBin(input);
  for (int i = 0; i < 48; i += 6) {
    String temp = input.substring(i, i + 6);
    int num = i / 6;
    int row = Integer.parseInt(
      temp.charAt(0) + "" + temp.charAt(5), 2);
    int col = Integer.parseInt(
      temp.substring(1, 5), 2);
    output += Integer.toHexString(
      sbox[num][row][col]);
  }
  return output;
}

String round(String input, String key)
{

  String left = input.substring(0, 8);
  String temp = input.substring(8, 16);
  String right = temp;

  temp = permutation(EP, temp);
  temp = xor(temp, key);
  temp = sBox(temp);
  temp = permutation(P, temp);
   left = xor(left, temp);
  return right + left;
}

String encrypt(String plainText, String key)
{
  int i;
  // get round keys
  String keys[] = getKeys(key);

  // initial permutation
  plainText = permutation(IP, plainText);
```

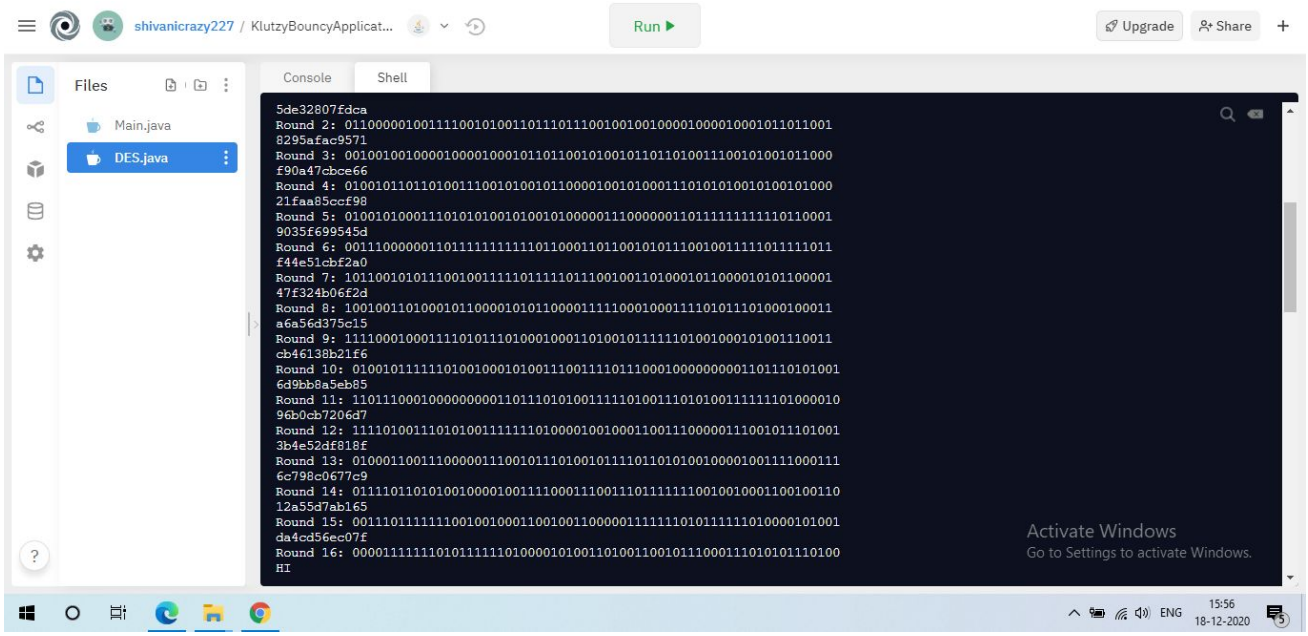
```
        // 16 rounds
        for (i = 0; i < 16; i++) {
                                    System.out.println(keys[i]);
          plainText = round(plainText, keys[i]);
                                    System.out.println("Round "+(i+1)+":
"+hextoBin(plainText));
        }

        // 32-bit swap
        plainText = plainText.substring(8, 16)
              + plainText.substring(0, 8);

        // final permutation
        plainText = permutation(IP1, plainText);
        return plainText;
      }


  }
  public static String ASCIItoHEX(String ascii)
  {
    String hex = "";
    for (int i = 0; i < ascii.length(); i++) {

      char ch = ascii.charAt(i);
      int in = (int)ch;
      String part = Integer.toHexString(in);
      hex += part;
    }
    return hex;
  }
  public static String hexToASCII(String hex)
  {
    String ascii = "";

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

      String part = hex.substring(i, i + 2);
      char ch = (char)Integer.parseInt(part, 16);
      ascii = ascii + ch;
```

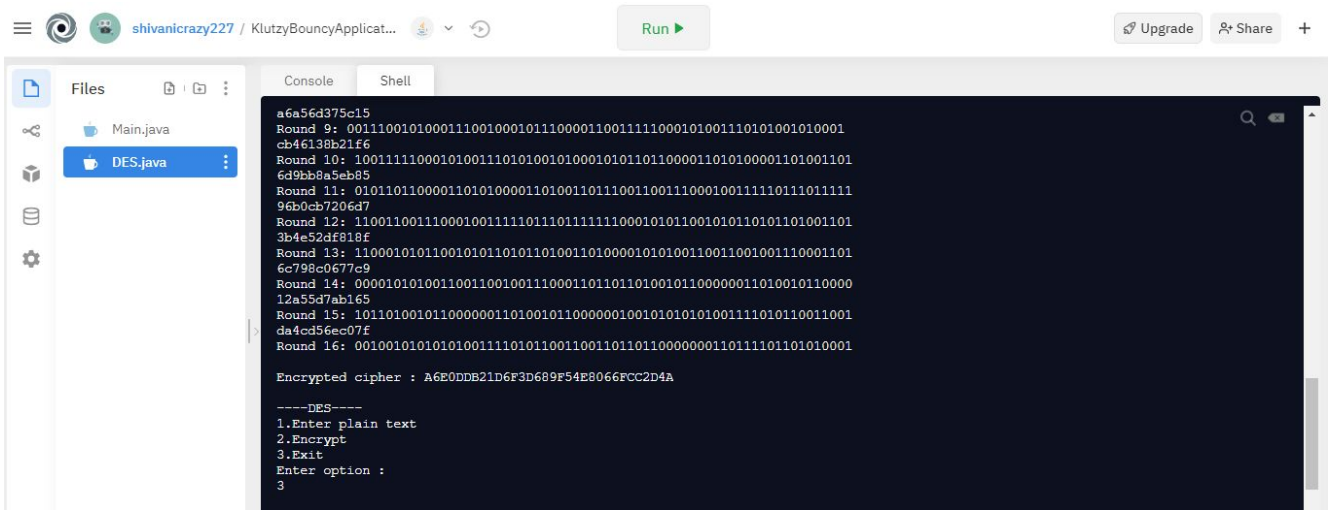
```
    }
    return ascii;
  }
  public static void main(String args[])
  {
    String text = "Hello world";
    String key = "0f1571c947d9e859";
    int o=1;
    Scanner input = new Scanner(System.in);
    Scanner in = new Scanner(System.in);
    DES1 cipher = new DES1();
    StringBuilder e = new StringBuilder("");
    StringBuilder d = new StringBuilder("");

    while(o!=3){
    System.out.println("----DES----");
    System.out.println("1.Enter plain text");
    System.out.println("2.Encrypt");
    System.out.println("3.Exit");
    System.out.println("Enter option : ");
    o = input.nextInt();
    if(o==1){
      System.out.println("Enter plain text : ");
      text = in.nextLine();
      System.out.println("Enter key : ");
      key = in.nextLine();
      System.out.println("\nPlain text : " + text);
      System.out.println("Key : " + key);
    }
    else if(o==2){
      text = ASCIItoHEX(text);
      int i=0;
      while(i<text.length()){
        int cnt=0;
        String t = "";
        StringBuilder enc = new StringBuilder("");
        while(i<text.length() && cnt<16){
          enc.append(text.charAt(i));
          i++;
          cnt++;
```

```
        }
                                while(cnt<16 && i>=text.length()){
                                    enc.append('0');
                                    i++;
                                    cnt++;
                                }
                            t = enc.toString();
                                System.out.println(hexToASCII(t));
                                System.out.println(t);
          e.append(cipher.encrypt(t,key));
        }
        System.out.println("\nEncrypted cipher : " +
e.toString().toUpperCase());
     }


     System.out.println();
     }
   }
}
```

OUTPUT:







RESULTS:

The DES encryption algorithm is implemented successfully