EX.NO.4: DATA ENCRYPTION STANDARD (DES)

CODE:

```
import java.util.*;

class DES {
  String plainText = new String();
  String cipherText = new String();
  String key = new String();
  String keys[] = new String[16];

// 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[] FP = { 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-Permutation 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 };


  String leftCircularShift(String input, int numBits) {
    int n = input.length();
    String shifted = "";
    shifted+=input.substring(numBits,n);
    shifted+=input.substring(0,numBits);
    return shifted;
```

```
}

String permutation(int[] table, String input) {
  String output = "";
  for (int i = 0; i < table.length; i++) {
    output += input.charAt(table[i] - 1);
  }
  return output;
}

String asciiToBinary(String ascii) {

  String bin_T = "";
  int n = ascii.length();

  for (int i = 0; i < n; i++) {
    int val = Integer.valueOf(ascii.charAt(i));
    String bin = "";
    while (val > 0) {
      if (val % 2 == 1)
          bin += '1';
      else
          bin += '0';
      val /= 2;
    }
    bin = reverse(bin);
    if (bin.length() < 8) {
      int l = 8 - bin.length();
      for (int j = 0; j < l; j++)
          bin = '0' + bin;
    }
    bin_T += bin + "";
  }
  return bin_T;
}

String reverse(String input) {
  char[] a = input.toCharArray();
  int l, r = 0;
  r = a.length - 1;
```

```
    for (l = 0; l < r; l++, r--) {
      char temp = a[l];
      a[l] = a[r];
      a[r] = temp;
    }
    return String.valueOf(a);
  }

  void generateKeys() {
    key = asciiToBinary(key);
    key = permutation(PC1, key);
    System.out.println("\nOutput of PC1 (56-bit) in hex is
"+binaryToHex(key).toUpperCase());
    String roundKey = key;
    System.out.println("\nThe round keys (48-bit) in hex are: ");
    for (int i = 0; i < 16; i++) {
      roundKey =
        leftCircularShift(roundKey.substring(0, 28), shiftBits[i]) +
        leftCircularShift(roundKey.substring(28, 56), shiftBits[i]);
      keys[i] = permutation(PC2, roundKey);
      System.out.println("Key "+(i+1)+":
"+binaryToHex(keys[i]).toUpperCase());
    }
  }

  String xor(String a, String b) {
    String ans = "";
    int n = a.length();
    for (int i = 0; i < n; i++) {
      if (a.charAt(i) == b.charAt(i))
        ans += "0";
      else
        ans += "1";
    }
    return ans;
  }

  int binaryToDecimal(String binary) {
```

```
  int dec = 0;
  int base = 1;
  int len = binary.length();

  for (int i = len - 1; i >= 0; i--) {
    if (binary.charAt(i) == '1')
      dec += base;
    base = base * 2;
  }
  return dec;
}

String sBox(String input) {
  String output = "";
  String bin = "";
  for (int i = 0; i < 48; i += 6) {
    String temp = input.substring(i, i + 6);
    int num = i / 6;
    int row = binaryToDecimal(temp.charAt(0) + "" + temp.charAt(5));
    int col = binaryToDecimal(temp.substring(1, 5));
    bin = Integer.toBinaryString(sbox[num][row][col]);
    if (bin.length() < 4) {
      int l = 4 - bin.length();
      for (int j = 0; j < l; j++)
          bin = '0' + bin;
    }
    output += bin;
  }
  return output;
}

String round(String input, String key, int r_num) {
  String left = input.substring(0, 32);
  String right = input.substring(32, 64);
  String temp = right;
  // Expansion permutation 32 to 48 bit
  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 " +(r_num + 1) +":
"+binaryToHex(right).toUpperCase() +" "
+binaryToHex(left).toUpperCase());
    // swapping
    return right + left;
  }

  void encrypt() {

    plainText = asciiToBinary(plainText);
    cipherText = plainText;
    // initial permutation
    cipherText = permutation(IP, cipherText);
    System.out.println("\nOutput of initial permutation IP (in hex) : " +
binaryToHex(cipherText).toUpperCase()+"\n");
    // 16 rounds
    for (int i = 0; i < 16; i++) {
      cipherText = round(cipherText, keys[i], i);
    }
    // 32-bit swap
    cipherText = cipherText.substring(32, 64) + cipherText.substring(0,
32);
    // final permutation
    cipherText = permutation(FP, cipherText);
    cipherText = binaryToHex(cipherText);
    System.out.println("\nThe cipher text is (in hex) " +
cipherText.toUpperCase());
  }

  String binaryToHex(String binary) {
    String hexStr = "";
    for (int i = 0; i < binary.length(); i = i + 4) {
      int decimal = Integer.parseInt(binary.substring(i, i + 4), 2);
      hexStr = hexStr + Integer.toString(decimal, 16);
    }
```

```
  return hexStr;
}

String hexToBinary(String hex) {
  String bin = "", temp = "";

  int n = Integer.parseInt(hex.substring(0, 7), 16);
  bin = Integer.toBinaryString(n);
  if (bin.length() < 28) {
    int l = 28 - bin.length();
    for (int j = 0; j < l; j++) bin = '0' + bin;
  }

  n = Integer.parseInt(hex.substring(7, 14), 16);
  temp = Integer.toBinaryString(n);
  if (temp.length() < 28) {
    int l = 28 - temp.length();
    for (int j = 0; j < l; j++) temp = '0' + temp;
  }
  bin += temp;

  n = Integer.parseInt(hex.substring(14, 16), 16);
  temp = Integer.toBinaryString(n);
  if (temp.length() < 8) {
    int l = 8 - temp.length();
    for (int j = 0; j < l; j++) temp = '0' + temp;
  }
  bin += temp;
  return bin;
}

String binaryToAscii(String binary) {
  String res = "";
  for (int i = 0; i < binary.length(); i = i + 8) {
    char ch = (char) binaryToDecimal(binary.substring(i, i + 8));
    res += ch;
  }
  return res;
}
```

```
  void decrypt() {

    cipherText = hexToBinary(cipherText);
    plainText = cipherText;

    // initial permutation
    plainText = permutation(IP, plainText);
    System.out.println("\nOutput of initial permutation IP (in hex) : " +
binaryToHex(plainText).toUpperCase());
    // 16-rounds
    for (int i = 15; i > -1; i--) {
      plainText = round(plainText, keys[i], 15 - i);
    }
    // 32-bit swap
    plainText = plainText.substring(32, 64) + plainText.substring(0, 32);
    plainText = permutation(FP, plainText);
    plainText = binaryToAscii(plainText);
    System.out.println("\nThe plain text is (in ASCII) " +
plainText.toUpperCase());
  }

  boolean validateString(String str){
    return str.length()==8;
  }

  boolean validateCipherText(String str){
    return str.length()==16;
  }

  public static void main(String args[]) {
    Scanner sc = new Scanner(System.in);
    DES des = new DES();
    System.out.println("\nDATA ENCRYPTION STANDARD - DES");

    System.out.println("\nKEY - GENERATION");
    System.out.println("*****************");
    System.out.print("\nEnter the key of length 8 (in ASCII): ");
    des.key = sc.next();
    while (!des.validateString(des.key)) {
        System.out.println("\nInvalid key");
```

```
        System.out.print("\nEnter the key: ");
        des.key = sc.next();
    }
    des.generateKeys();

    System.out.println("\nENCRYPTION");
    System.out.println("**********");
    System.out.print("\nEnter the plainText of length 8 (in ASCII): ");
    des.plainText = sc.next();
    while (!des.validateString(des.plainText)) {
        System.out.println("\nInvalid plain text length");
        System.out.print("\nEnter the plainText of length 8 (in ASCII): ");
        des.plainText = sc.next();
    }
    des.encrypt();

    System.out.println("\nDECRYPTION");
    System.out.println("**********");
    System.out.print("\nEnter the cipherText of length 16 (in hex): ");
    des.cipherText = sc.next();
    while (!des.validateCipherText(des.cipherText)) {
        System.out.println("\nInvalid cipher text length");
        System.out.print("\nEnter the cipherText of length 8 (in ASCII): ");
        des.cipherText = sc.next();
    }
    System.out.print("\nEnter the key of length 8 (in ASCII): ");
    des.key = sc.next();
    while (!des.validateString(des.key)) {
        System.out.println("\nInvalid key");
        System.out.print("\nEnter the key: ");
        des.key = sc.next();
    }
    des.generateKeys();
    des.decrypt();
  }
}
```

OUTPUT:

Example 1:

```
C:\Users\WELCOME\Desktop\CNS lab\ex4>java DES

DATA ENCRYPTION STANDARD - DES

KEY - GENERATION
*****************

Enter the key of length 8 (in ASCII): BACTERIA

Output of PC1 (56-bit) in hex is 00FF0022518408

The round keys (48-bit) in hex are:
Key 1: A092C2436200
Key 2: A01252008A83
Key 3: 245A50160415
Key 4: 0671500B01C0
Key 5: 0E455100E101
Key 6: 4F4109620404
Key 7: 0B8189C8018A
Key 8: 19088B045209
Key 9: 190A88280C40
Key 10: 10388C48C016
Key 11: 102C44054488
Key 12: 406C24881041
Key 13: C0A52482C224
Key 14: C08623100F80
Key 15: E19222980011
Key 16: A0922AA100E4

ENCRYPTION
**********

Enter the plainText of length 8 (in ASCII): ENVELOPE

Output of initial permutation IP (in hex) : FF44BFA900003226

Round 1: 00003226 06cb4772
Round 2: 06cb4772 55e2cecc
Round 3: 55e2cecc 131c8dd0
Round 4: 131c8dd0 132021a6
Round 5: 132021a6 031bf1f1
Round 6: 031bf1f1 22981f2d
```

```
Round 5: 132021a6 031bf1f1
Round 6: 031bf1f1 22981f2d
Round 7: 22981f2d 4db4e528
Round 8: 4db4e528 63d92a26
Round 9: 63d92a26 245a4f2e
Round 10: 245a4f2e 984a900b
Round 11: 984a900b 22d1d926
Round 12: 22d1d926 2e714bb6
Round 13: 2e714bb6 a29e9f14
Round 14: a29e9f14 af88dd5e
Round 15: af88dd5e de80ae2a
Round 16: de80ae2a 0336d261

The cipher text is (in hex) 41DE988A941B85AC

DECRYPTION
**********

Enter the cipherText of length 16 (in hex): 41DE988A941B85AC

Enter the key of length 8 (in ASCII): BACTERIA

Output of PC1 (56-bit) in hex is 00FF0022518408

The round keys (48-bit) in hex are:
Key 1: A092C2436200
Key 2: A01252008A83
Key 3: 245A50160415
Key 4: 0671500B01C0
Key 5: 0E455100E101
Key 6: 4F4109620404
Key 7: 0B8189C8018A
Key 8: 19088B045209
Key 9: 190A88280C40
Key 10: 10388C48C016
Key 11: 102C44054488
Key 12: 406C24881041
Key 13: C0A52482C224
Key 14: C08623100F80
Key 15: E19222980011
Key 16: A0922AA100E4

Output of initial permutation IP (in hex) : 0336D261DE80AE2A
```

```
Output of initial permutation IP (in hex) : 0336D261DE80AE2A
Round 1: de80ae2a af88dd5e
Round 2: af88dd5e a29e9f14
Round 3: a29e9f14 2e714bb6
Round 4: 2e714bb6 22d1d926
Round 5: 22d1d926 984a900b
Round 6: 984a900b 245a4f2e
Round 7: 245a4f2e 63d92a26
Round 8: 63d92a26 4db4e528
Round 9: 4db4e528 22981f2d
Round 10: 22981f2d 031bf1f1
Round 11: 031bf1f1 132021a6
Round 12: 132021a6 131c8dd0
Round 13: 131c8dd0 55e2cecc
Round 14: 55e2cecc 06cb4772
Round 15: 06cb4772 00003226
Round 16: 00003226 ff44bfa9

The plain text is (in ASCII) ENVELOPE

C:\Users\WELCOME\Desktop\CNS lab\ex4>
```

Example 2:

```
C:\Users\WELCOME\Desktop\CNS lab\ex4>java DES

DATA ENCRYPTION STANDARD - DES

KEY - GENERATION
*****************

Enter the key of length 8 (in ASCII): QWERTYUIOP

Invalid key

Enter the key: CDE34RF

Invalid key

Enter the key: MEDICINE

Output of PC1 (56-bit) in hex is 00FF00050C7690
```

```
Output of PC1 (56-bit) in hex is 00FF00050C7690

The round keys (48-bit) in hex are:
Key 1: A09242B0D036
Key 2: A012528B491C
Key 3: 245250017390
Key 4: 065150F10025
Key 5: 0E4151C20A8E
Key 6: 0F410914339D
Key 7: 0B01893310E1
Key 8: 19088942A923
Key 9: 19088834C18B
Key 10: 10288C861443
Key 11: 102C04CEA360
Key 12: 402C2430CF48
Key 13: C0A424589412
Key 14: C08622CD6428
Key 15: E09222287A48
Key 16: A092225A0C31

ENCRYPTION
**********

Enter the plainText of length 8 (in ASCII): ZXSAQWERFD
```

```
Invalid plain text length

Enter the plainText of length 8 (in ASCII): SECURITY

Output of initial permutation IP (in hex) : FFD94AAF0000A015

Round 1: 0000A015 A1D52C5A
Round 2: A1D52C5A 9715753B
Round 3: 9715753B F69F656A
Round 4: F69F656A 52B44166
Round 5: 52B44166 A629B2A3
Round 6: A629B2A3 7739B6AF
Round 7: 7739B6AF C8739C2B
Round 8: C8739C2B AAA342F1
Round 9: AAA342F1 C386B771
Round 10: C386B771 0FB0FB8A
```

```
Round 10: C386B771 0FB0FB8A
Round 11: 0FB0FB8A A9A954F6
Round 12: A9A954F6 24DA6D79
Round 13: 24DA6D79 08B2BEC7
Round 14: 08B2BEC7 C24B5360
Round 15: C24B5360 86FD3BCE
Round 16: 86FD3BCE 5A34508E

The cipher text is (in hex): 28CBB36B7C3866A3

DECRYPTION
**********

Enter the cipherText of length 16 (in hex): 2A5F719E

Invalid cipher text length

Enter the cipherText of length 16 (in hex): 28CBB36B7C3866A3

Enter the key of length 8 (in ASCII): MEDICINE

Output of PC1 (56-bit) in hex is 00FF00050C7690
```

```
The round keys (48-bit) in hex are:
Key 1: A09242B0D036
Key 2: A012528B491C
Key 3: 245250017390
Key 4: 065150F10025
Key 5: 0E4151C20A8E
Key 6: 0F410914339D
Key 7: 0B01893310E1
Key 8: 19088942A923
Key 9: 19088834C18B
Key 10: 10288C861443
Key 11: 102C04CEA360
Key 12: 402C2430CF48
Key 13: C0A424589412
Key 14: C08622CD6428
Key 15: E09222287A48
Key 16: A092225A0C31
```

```
The round keys (48-bit) in hex are:
Key 1: A09242B0D036
Key 2: A012528B491C
Key 3: 245250017390
Key 4: 065150F10025
Key 5: 0E4151C20A8E
Key 6: 0F410914339D
Key 7: 0B01893310E1
Key 8: 19088942A923
Key 9: 19088834C18B
Key 10: 10288C861443
Key 11: 102C04CEA360
Key 12: 402C2430CF48
Key 13: C0A424589412
Key 14: C08622CD6428
Key 15: E09222287A48
Key 16: A092225A0C31

Output of initial permutation IP (in hex) : 5A34508E86FD3BCE
Round 1: 86FD3BCE C24B5360
Round 2: C24B5360 08B2BEC7
Round 3: 08B2BEC7 24DA6D79
Round 4: 24DA6D79 A9A954F6
Round 5: A9A954F6 0FB0FB8A
Round 6: 0FB0FB8A C386B771
Round 7: C386B771 AAA342F1
Round 8: AAA342F1 C8739C2B
Round 9: C8739C2B 7739B6AF
Round 10: 7739B6AF A629B2A3
Round 11: A629B2A3 52B44166
Round 12: 52B44166 F69F656A
Round 13: F69F656A 9715753B
Round 14: 9715753B A1D52C5A
Round 15: A1D52C5A 0000A015
Round 16: 0000A015 FFD94AAF

The plain text is (in ASCII): SECURITY

C:\Users\WELCOME\Desktop\CNS lab\ex4>
```