Experiment no:5

Aim : Write a program to encrypt a long message using DES algorithm

CODE:

```
import java.util.*;

public class DESJava {

    // Hex to binary map
    static Map<Character, String> hexToBinMap = new HashMap<>();
    static Map<String, Character> binToHexMap = new HashMap<>();
    static {
        String[] bin = {"0000","0001","0010","0011","0100","0101","0110","0111",
                        "1000","1001","1010","1011","1100","1101","1110","1111"};
        char[] hex = "0123456789ABCDEF".toCharArray();
        for (int i = 0; i < 16; i++) {
            hexToBinMap.put(hex[i], bin[i]);
            binToHexMap.put(bin[i], hex[i]);
        }
    }

    // Initial Permutation
    static int[] initial_perm = {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};

    // Expansion D-box
    static int[] exp_d = {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
    static int[] per = {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};

    // Final Permutation
    static int[] final_perm = {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,
```

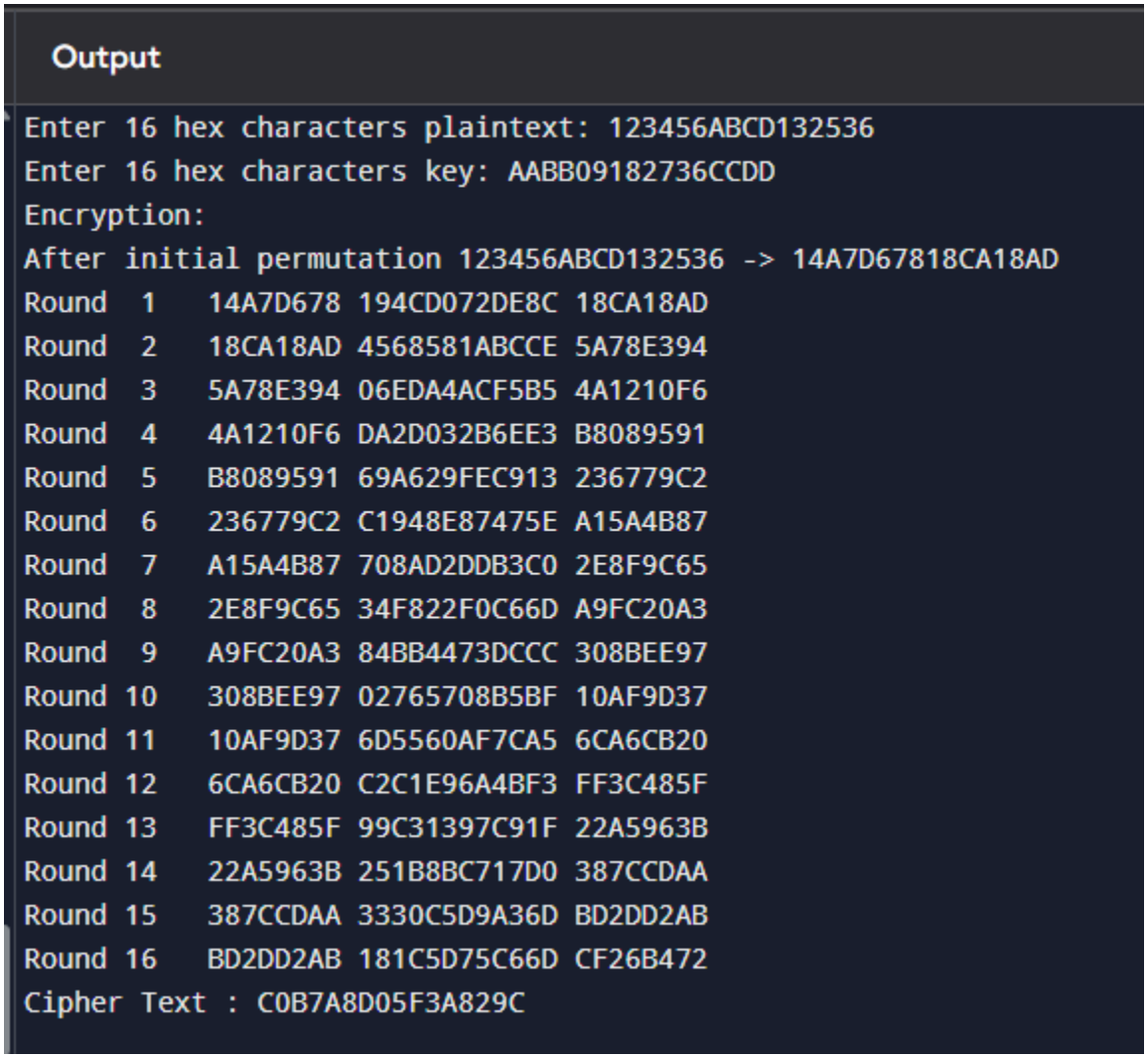
```
                    36,4,44,12,52,20,60,28,35,3,43,11,51,19,59,27,
                    34,2,42,10,50,18,58,26,33,1,41,9,49,17,57,25};

// Parity drop table
static int[] keyp = {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};

// Key compression table
static int[] key_comp = {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};

// Shifts
static int[] shift_table = {1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};

// S-boxes
static int[][][] sbox = new int[][][] {
   {{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}}
};

// Convert hex string to binary string
static String hex2bin(String s) {
   StringBuilder bin = new StringBuilder();
   for (char c : s.toCharArray())
      bin.append(hexToBinMap.get(Character.toUpperCase(c)));
   return bin.toString();
```

```
}

// Convert binary string to hex string
static String bin2hex(String s) {
    StringBuilder hex = new StringBuilder();
    for (int i = 0; i < s.length(); i += 4)
        hex.append(binToHexMap.get(s.substring(i, i + 4)));
    return hex.toString();
}

// Permutation
static String permute(String k, int[] arr) {
    StringBuilder perm = new StringBuilder();
    for (int i : arr)
        perm.append(k.charAt(i - 1));
    return perm.toString();
}

// Left shift
static String shiftLeft(String k, int n) {
    return k.substring(n) + k.substring(0, n);
}

// XOR two binary strings
static String xor(String a, String b) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < a.length(); i++)
        sb.append(a.charAt(i) == b.charAt(i) ? '0' : '1');
    return sb.toString();
}

// Convert binary string to decimal
static int bin2dec(String bin) {
    return Integer.parseInt(bin, 2);
}

// Convert decimal to 4-bit binary string
static String dec2bin(int val) {
    String s = Integer.toBinaryString(val);
    while (s.length() < 4) s = "0" + s;
    return s;
}

// DES encryption/decryption function with verbose output
```

```
    static String des(String pt, String[] rkb, boolean isEncryption) {
        System.out.println((isEncryption ? "Encryption" : "Decryption") + ":");
        String binPt = hex2bin(pt);
        binPt = permute(binPt, initial_perm);
        System.out.println("After initial permutation " + pt + " -> " + bin2hex(binPt));

        String left = binPt.substring(0, 32);
        String right = binPt.substring(32);

        for (int i = 0; i < 16; i++) {
            System.out.printf("Round %2d   %8s %8s %8s\n", i + 1, bin2hex(left), bin2hex(rkb[i]),
bin2hex(right));
            String rightExp = permute(right, exp_d);
            String xorOut = xor(rightExp, rkb[i]);
            StringBuilder sboxStr = new StringBuilder();
            for (int j = 0; j < 8; j++) {
                String sixBits = xorOut.substring(j*6, j*6+6);
                int row = bin2dec("" + sixBits.charAt(0) + sixBits.charAt(5));
                int col = bin2dec(sixBits.substring(1,5));
                sboxStr.append(dec2bin(sbox[j][row][col]));
            }
            sboxStr = new StringBuilder(permute(sboxStr.toString(), per));
            String temp = xor(left, sboxStr.toString());
            left = right;
            right = temp;
        }

        String combine = right + left; // swap halves
        combine = permute(combine, final_perm);
        String result = bin2hex(combine);

        System.out.println("Cipher Text : " + result + "\n");

        return result;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter 16 hex characters plaintext: ");
        String pt = sc.nextLine().trim();
        while (pt.length() != 16 || !pt.matches("[0-9A-Fa-f]+")) {
            System.out.print("Invalid input. Enter 16 hex characters plaintext: ");
            pt = sc.nextLine().trim();
```

```
        }

        System.out.print("Enter 16 hex characters key: ");
        String key = sc.nextLine().trim();
        while (key.length() != 16 || !key.matches("[0-9A-Fa-f]+")) {
            System.out.print("Invalid input. Enter 16 hex characters key: ");
            key = sc.nextLine().trim();
        }

        // Key processing
        String keyBin = hex2bin(key);
        keyBin = permute(keyBin, keyp);
        String left = keyBin.substring(0, 28);
        String right = keyBin.substring(28);

        String[] rkb = new String[16];
        for (int i = 0; i < 16; i++) {
            left = shiftLeft(left, shift_table[i]);
            right = shiftLeft(right, shift_table[i]);
            String combine = left + right;
            rkb[i] = permute(combine, key_comp);
        }

        // Encryption
        String cipher = des(pt, rkb, true);

        // Decryption (reverse round keys)
        String[] rkbRev = new String[16];
        for (int i = 0; i < 16; i++) rkbRev[i] = rkb[15 - i];

        // Decryption
        des(cipher, rkbRev, false);

        sc.close();
    }
}
```

OUTPUT:



```
Output

Enter 16 hex characters plaintext: 123456ABCD132536
Enter 16 hex characters key: AABB09182736CCDD
Encryption:
After initial permutation 123456ABCD132536 -> 14A7D67818CA18AD
Round  1   14A7D678 194CD072DE8C 18CA18AD
Round  2   18CA18AD 4568581ABCCE 5A78E394
Round  3   5A78E394 06EDA4ACF5B5 4A1210F6
Round  4   4A1210F6 DA2D032B6EE3 B8089591
Round  5   B8089591 69A629FEC913 236779C2
Round  6   236779C2 C1948E87475E A15A4B87
Round  7   A15A4B87 708AD2DDB3C0 2E8F9C65
Round  8   2E8F9C65 34F822F0C66D A9FC20A3
Round  9   A9FC20A3 84BB4473DCCC 308BEE97
Round 10   308BEE97 02765708B5BF 10AF9D37
Round 11   10AF9D37 6D5560AF7CA5 6CA6CB20
Round 12   6CA6CB20 C2C1E96A4BF3 FF3C485F
Round 13   FF3C485F 99C31397C91F 22A5963B
Round 14   22A5963B 251B8BC717D0 387CCDAA
Round 15   387CCDAA 3330C5D9A36D BD2DD2AB
Round 16   BD2DD2AB 181C5D75C66D CF26B472
Cipher Text : C0B7A8D05F3A829C
```

```
Decryption:
After initial permutation C0B7A8D05F3A829C -> 19BA9212CF26B472
Round  1   19BA9212 181C5D75C66D CF26B472
Round  2   CF26B472 3330C5D9A36D BD2DD2AB
Round  3   BD2DD2AB 251B8BC717D0 387CCDAA
Round  4   387CCDAA 99C31397C91F 22A5963B
Round  5   22A5963B C2C1E96A4BF3 FF3C485F
Round  6   FF3C485F 6D5560AF7CA5 6CA6CB20
Round  7   6CA6CB20 02765708B5BF 10AF9D37
Round  8   10AF9D37 84BB4473DCCC 308BEE97
Round  9   308BEE97 34F822F0C66D A9FC20A3
Round 10   A9FC20A3 708AD2DDB3C0 2E8F9C65
Round 11   2E8F9C65 C1948E87475E A15A4B87
Round 12   A15A4B87 69A629FEC913 236779C2
Round 13   236779C2 DA2D032B6EE3 B8089591
Round 14   B8089591 06EDA4ACF5B5 4A1210F6
Round 15   4A1210F6 4568581ABCCE 5A78E394
Round 16   5A78E394 194CD072DE8C 18CA18AD
Cipher Text : 123456ABCD132536
```