

**Shri Ramdeobaba College of Engineering and Management, Nagpur**  
**Department of Electronics and Communication Engineering**

**Statement of clear goals, use of appropriate methods, significance of results, effective presentation**

**Course : Error Correcting Codes**

**Course Code: ECT453-1**

**Semester VII**

**Session 2024-25**

The “Error Correcting Codes” course plays a foundational role in preparing students for advanced studies and professional work in digital communication and information theory. It introduces students to both classical and modern techniques used to detect and correct errors in data transmission and storage, with direct relevance to real-world applications such as wireless communications, data networks, and satellite systems.

Key topics typically covered include linear block codes, cyclic redundancy check (CRC), Hamming codes, convolutional codes, turbo codes, and LDPC (Low-Density Parity-Check) codes.

**Although this course does not include a formal laboratory component, students were given the opportunity to bridge theory and practice through hands-on programming assignments. Each student was encouraged to implement the encoding and decoding algorithms for selected error-correcting codes using a programming language of their choice. This flexibility allowed them to apply their existing computational skills while exploring algorithmic logic, in a practical context.**

**Problem statements:**

**Assignment 1:**

Write a generic program to demonstrate Shano Fano Source Coding/Huffman Source Coding method.

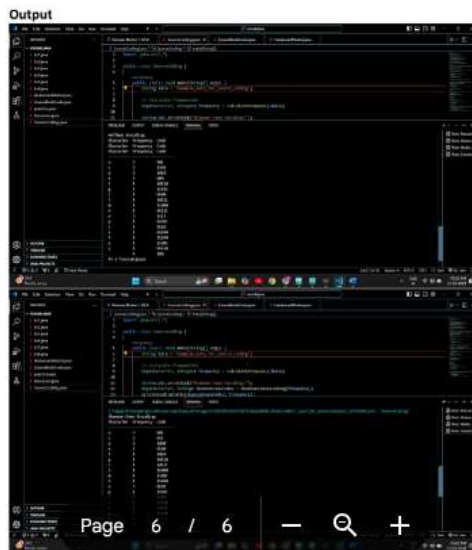
Sample answers:

```
J:\HuffmanCoding\java> javac HuffmanCoding_1.java
public class HuffmanCoding {
    public static void main(String[] args) {
        HuffmanCoding huffmanCoding = new HuffmanCoding();
        huffmanCoding.run();
    }

    private void run() {
        int n = 6;
        double[] probabilities = {0.3, 0.25, 0.2, 0.12, 0.08, 0.05};
        HuffmanCoding huffmanCoding = new HuffmanCoding(n, probabilities);
        huffmanCoding.calculateCodes();
    }

    private void calculateCodes() {
        System.out.println("The Huffman Codes are:");
        for (int i = 0; i < probabilities.length; i++) {
            System.out.println("Symbol " + i + " : Probability: " + probabilities[i] + " Code: " + huffmanCoding.getCodes()[i]);
        }
        System.out.println("Average Length: " + huffmanCoding.getAverageLength());
        System.out.println("Efficiency: " + huffmanCoding.getEfficiency());
        System.out.println("Redundancy: " + huffmanCoding.getRedundancy());
    }
}
```

ViniyaBhise (VII A Roll No 67)



Hemant Paunikar (VII A Roll No 54)

```
PS C:\Users\HP\OneDrive\Documents\Space\JavaPractice> cd "c:\Users\HP\OneDrive\Documents\Space\JavaPractice\Heaps_\"; if ($?) { javac huffman_code.java ; i
f ($?) { java huffman_code }
Enter the total number of symbols: 6
Enter the probabilities of symbols: 0.3 0.25 0.2 0.12 0.08 0.05
The calculated codes are:
Probability: 0.05 --- Code: 1001
Probability: 0.08 --- Code: 1000
Probability: 0.12 --- Code: 101
Probability: 0.2 --- Code: 11
Probability: 0.25 --- Code: 01
Probability: 0.3 --- Code: 00
```

Nikhil Kakde (VII B, Roll No 54)

## Assignment 2:

Write a code to detect single bit error for a (7,4) linear block code whose parity check matrix is

$$H = \begin{matrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{matrix}$$

(Hint : Use Syndrome Decoder block diagram to design the flow of code.

Generate all code vectors, Generate all possible syndrome vectors, and map the things.)

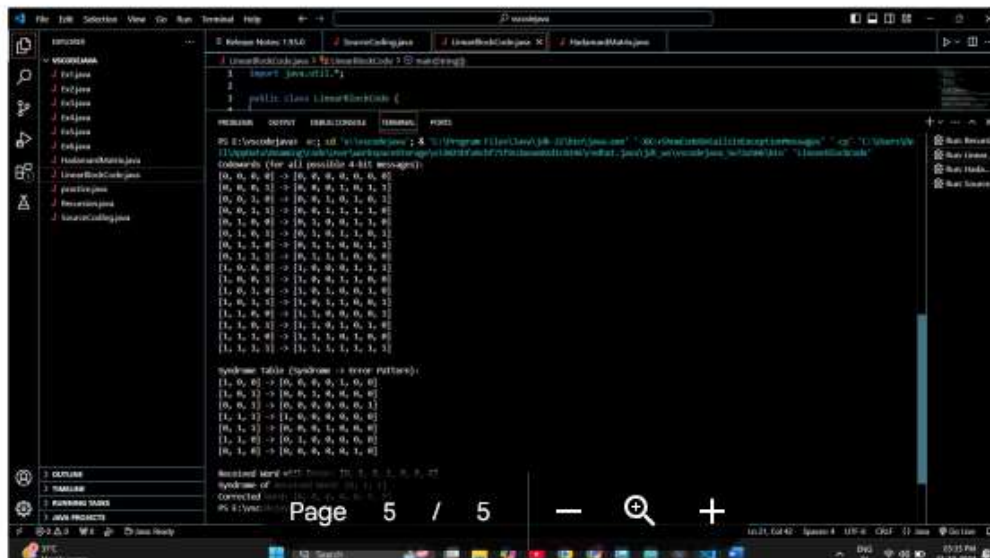
Sample Answers:



```
Run  Explo
"C:\Users\adity\AppData\Local\Programs\Eclipse Adoptium\jdk-17.0.9-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.3.2\lib
1 1 0 1 0 0
1 1 0 1 0 1 0
1 0 0 1 0 0 1
1 0 1
Bit 3 is erroneous.
The correct codeword is
1 1 1 1 1 1 1
Process finished with exit code 0
```

Aditya Sanghi (VII A Roll No 11)

## Output

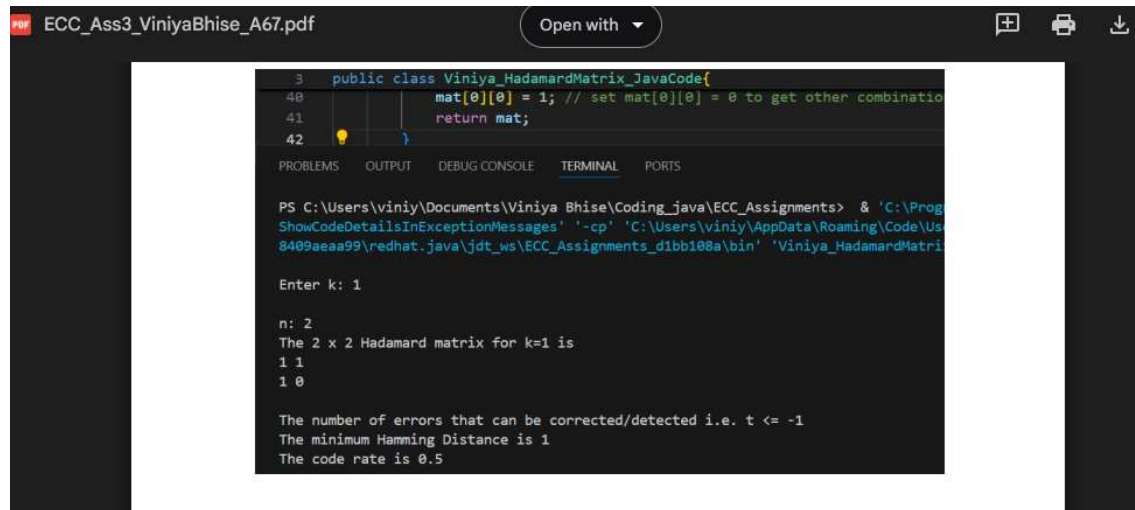


```
1 import java.util.*;
2
3 public class LinearBlockCode {
4
5     public static void main(String[] args) {
6         // Generate all possible 4-bit messages
7         List<String> messages = new ArrayList<>();
8         for (int i = 0; i < 16; i++) {
9             String message = Integer.toBinaryString(i).padStart(4, '0');
10            messages.add(message);
11        }
12
13        // Generate all possible 7-bit codewords
14        List<String> codewords = new ArrayList<>();
15        for (String message : messages) {
16            String codeword = message + parity(message);
17            codewords.add(codeword);
18        }
19
20        // Print the results
21        System.out.println("Messages (for all possible 4-bit messages):");
22        for (String message : messages) {
23            System.out.println(message);
24        }
25
26        System.out.println("Codewords (for all possible 7-bit codewords):");
27        for (String codeword : codewords) {
28            System.out.println(codeword);
29        }
30
31        // Syndrome Table (Syndrome -> Error Pattern)
32        System.out.println("Syndrome Table (Syndrome -> Error Pattern):");
33        for (String syndrome : syndromes) {
34            System.out.println(syndrome);
35        }
36    }
37
38    // Generate parity for a 4-bit message
39    private static String parity(String message) {
40        int p1 = Integer.parseInt(message.substring(0, 2));
41        int p2 = Integer.parseInt(message.substring(2, 4));
42        int p3 = p1 ^ p2;
43        return String.format("%4s", p1 + p2 + p3).replace(' ', '0');
44    }
45
46    // Generate syndrome for a 7-bit codeword
47    private static String syndrome(String codeword) {
48        int s1 = Integer.parseInt(codeword.substring(0, 3));
49        int s2 = Integer.parseInt(codeword.substring(3, 6));
50        int s3 = Integer.parseInt(codeword.substring(6, 7));
51        return String.format("%4s", s1 + s2 + s3).replace(' ', '0');
52    }
53
54    // Generate all possible syndrome vectors
55    private static List<String> syndromes = new ArrayList<>();
56    for (int i = 0; i < 8; i++) {
57        String syndrome = Integer.toBinaryString(i).padStart(3, '0');
58        syndromes.add(syndrome);
59    }
60
61    // Generate all possible error patterns
62    private static List<String> errorPatterns = new ArrayList<>();
63    for (int i = 0; i < 7; i++) {
64        String errorPattern = Integer.toBinaryString(i).padStart(7, '0');
65        errorPatterns.add(errorPattern);
66    }
67
68    // Map syndrome to error pattern
69    private static Map<String, String> syndromeToErrorPattern = new HashMap<>();
70    for (String syndrome : syndromes) {
71        for (String errorPattern : errorPatterns) {
72            syndromeToErrorPattern.put(syndrome, errorPattern);
73        }
74    }
75
76    // Decode a received codeword
77    private static String decode(String codeword) {
78        String syndrome = syndrome(codeword);
79        String errorPattern = syndromeToErrorPattern.get(syndrome);
80        return codeword.substring(0, 4) + errorPattern.substring(4, 7);
81    }
82
83    // Detect error in a received codeword
84    private static boolean detectError(String codeword) {
85        String syndrome = syndrome(codeword);
86        return !syndrome.equals("000");
87    }
88
89    // Correct a received codeword
90    private static String correct(String codeword) {
91        if (detectError(codeword)) {
92            return decode(codeword);
93        } else {
94            return codeword;
95        }
96    }
97
98    // Test the code
99    private static void test() {
100        List<String> testMessages = new ArrayList<>();
101        for (String message : messages) {
102            String codeword = message + parity(message);
103            testMessages.add(codeword);
104        }
105
106        for (String codeword : testMessages) {
107            System.out.println("Received word: " + codeword);
108            System.out.println("Syndrome: " + syndrome(codeword));
109            System.out.println("Corrected word: " + correct(codeword));
110        }
111    }
112
113    // Main method
114    public static void main(String[] args) {
115        test();
116    }
117
118 }
Page 5 / 5
```

Hemant Paunikar (VII A Roll No 54)

### Assignment 3:

Write a generic code to generate  $n \times n$  Hadamard Matrix. Also calculate number of errors that can be corrected and code rate.



```
3 public class Viniya_HadamardMatrix_JavaCode{
40     mat[0][0] = 1; // set mat[0][0] = 0 to get other combinatio
41     return mat;
42 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

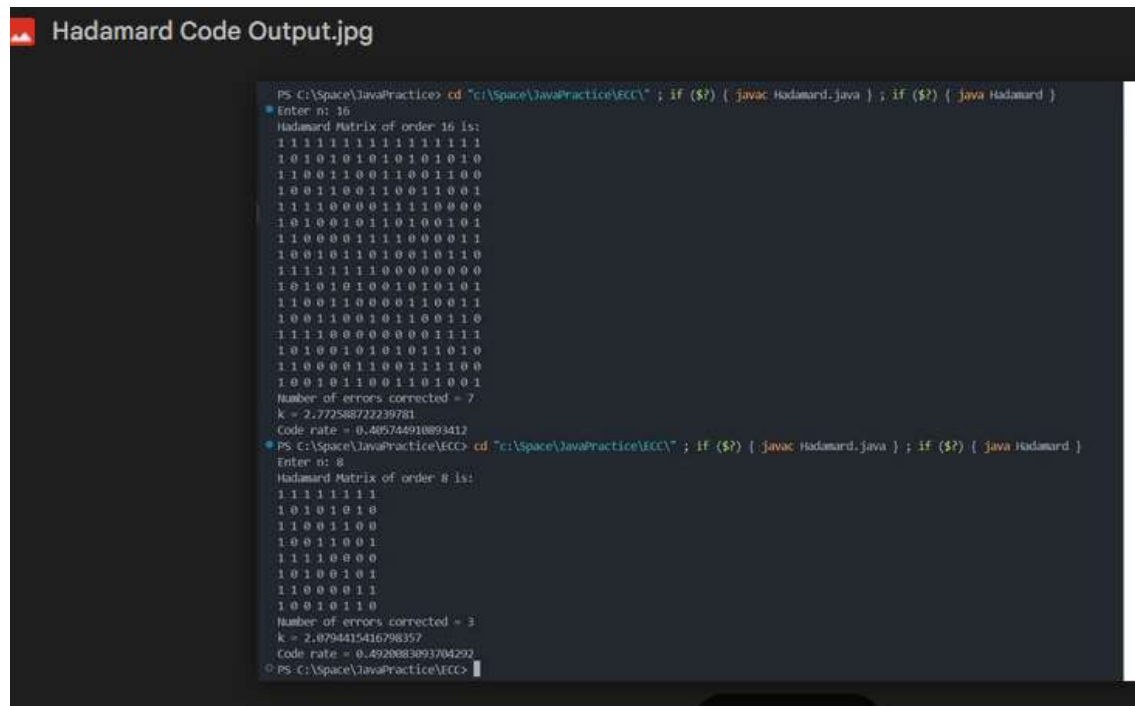
```
PS C:\Users\viniy\Documents\Viniya Bhise\Coding_java\ECC_Assignments> & 'C:\Prog
ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\viniy\AppData\Roaming\Code\Us
8409aeaa99\redhat.java\jdt_ws\ECC_Assignments_d1bb108a\bin' 'Viniya_HadamardMatri

Enter k: 1

n: 2
The 2 x 2 Hadamard matrix for k=1 is
1 1
1 0

The number of errors that can be corrected/detected i.e. t <= -1
The minimum Hamming Distance is 1
The code rate is 0.5
```

ViniyaBhise (VII A Roll No 67)



```
Hadamard Code Output.jpg

PS C:\Space\JavaPractice> cd "c:\Space\JavaPractice\ECC\" ; if ($?) { javac Hadamard.java } ; if ($?) { java Hadamard }
Enter n: 16
Hadamard Matrix of order 16 is:
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1
1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1
1 1 0 0 0 0 1 1 1 0 0 0 0 1 1
1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0
1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1
1 1 0 0 1 1 0 0 0 1 1 0 0 1 1
1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0
1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1
1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0
1 1 0 0 0 0 1 1 0 0 1 1 1 1 0 0
1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1
Number of errors corrected = 7
k = 2.77258872239781
Code rate = 0.405744910893412
PS C:\Space\JavaPractice\ECC> cd "c:\Space\JavaPractice\ECC\" ; if ($?) { javac Hadamard.java } ; if ($?) { java Hadamard }
Enter n: 8
Hadamard Matrix of order 8 is:
1 1 1 1 1 1 1 1
1 0 1 0 1 0 1 0
1 1 0 0 1 1 0 0
1 0 0 1 1 0 0 1
1 1 1 1 0 0 0 0
1 0 1 0 0 1 0 1
1 1 0 0 0 0 1 1
1 0 0 1 0 1 1 0
Number of errors corrected = 3
k = 2.0794415416798357
Code rate = 0.4920083093704292
PS C:\Space\JavaPractice\ECC>
```

Nikhil Kakde (VII A Roll No 48)

**Outcome:**The feedback from students has been positive, with many reporting that the programming tasks significantly enhanced their understanding of the subject matter. It also encouraged independent learning and problem-solving, both critical skills in engineering education.

Dr.(Mrs.) MridulaKorde  
Course Coordinator