Expt 4:

Consider a network with data frames transmitted from sender to receiver. Design a code for bit stuffing and un-stuffing mechanism considering the given input pattern.

Bit stuffing is a process used in data transmission protocols to prevent confusion with control information (like frame delimiters). When the sender detects a specific pattern in the data that could be mistaken for a control signal, it inserts a bit to break up the pattern. The receiver then removes this extra bit to retrieve the original data.

Let's assume we're working with a basic pattern like 01111110 (often used as a frame delimiter in protocols like HDLC), and we want to avoid having six consecutive 1 bits (111111) in our data.

Here's how you can implement bit stuffing and unstuffing in Java:

JAVA CODE:

```
import java.util.Scanner;


public class BitStuffing {


  // Method to perform bit stuffing
  public static String bitStuffing(String data) {
    StringBuilder stuffedData = new StringBuilder();
    int consecutiveOnes = 0;


    for (int i = 0; i < data.length(); i++) {
      char bit = data.charAt(i);
      stuffedData.append(bit);


      if (bit == '1') {
        consecutiveOnes++;
        if (consecutiveOnes == 5) {
          // Insert a '0' after five consecutive '1's
          stuffedData.append('0');
          consecutiveOnes = 0;
        }
      } else {
        consecutiveOnes = 0;
```

```
        }
    }

    return stuffedData.toString();
}

// Method to perform bit unstuffing
public static String bitUnstuffing(String stuffedData) {
    StringBuilder unstuffedData = new StringBuilder();
    int consecutiveOnes = 0;

    for (int i = 0; i < stuffedData.length(); i++) {
        char bit = stuffedData.charAt(i);

        if (bit == '1') {
            consecutiveOnes++;
            unstuffedData.append(bit);

            if (consecutiveOnes == 5) {
                // Skip the next '0' after five consecutive '1's
                i++;  // Skip the stuffed '0'
                consecutiveOnes = 0;
            }
        } else {
            consecutiveOnes = 0;
            unstuffedData.append(bit);
        }
    }

    return unstuffedData.toString();
}
```

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

        // Input data frame
        System.out.print("Enter the binary data frame: ");
        String data = scanner.nextLine();

        // Perform bit stuffing
        String stuffedData = bitStuffing(data);
        System.out.println("Stuffed Data: " + stuffedData);

        // Perform bit unstuffing
        String unstuffedData = bitUnstuffing(stuffedData);
        System.out.println("Unstuffed Data: " + unstuffedData);

        scanner.close();
    }
}
```

How the Code Works:

1. **Bit Stuffing:**
    - The code iterates through the input binary string (data).
    - It counts the consecutive 1s. If it detects five consecutive 1s, it stuffs (inserts) a 0 bit after them to break the pattern.
    - The resulting string (stuffedData) is the transmitted data that includes these extra 0 bits.
2. **Bit Unstuffing:**
    - When the receiver gets the stuffedData, it checks for sequences of five consecutive 1s.
    - If it finds such a sequence, it skips the next 0 (the stuffed bit) to retrieve the original data frame.

Example Usage:

- **Input:**
    - Original Data: 011111101111110
- **Output:**
    - Stuffed Data: 0111110101111100
    - Unstuffed Data: 011111101111110

The stuffed data contains the extra 0s inserted after sequences of five consecutive 1s. The unstuffing process removes these 0s, returning the data to its original form.

Application:

This method is particularly useful in communication protocols to ensure that patterns in the data don't interfere with control sequences, such as frame delimiters.