Department of Computer Applications MCA-3rd Sem CN LAB Programs	
S.NO.	**LIST OF PROGRAMS**
1	**Creating a client-server application in Java involves writing two separate programs: one for the server and one for the client.**
2	**Implement the data link layer framing methods such as One Time Pad using Java Program.**
3	**Implement the data link layer framing methods such as Find My PC IP Address using Java Program.**
4	**To write a Java program to implement the Leaky Bucket algorithm to smooth out bursty traffic and control network congestion.**
5	**To write a Java program to simulate the Stop-and-Wait Sliding Window protocol for flow control.**
6	**To write a Java program to implement the Two-Dimensional Redundancy Parity Check system to compute block check validation bits**
7	**a) Write Java Program For Bit Stuffing Using Java** **b) Write Java Program For Character Stuffing Using Java**
8	**Design And Develop A Java Program To Implement The CRC Polynomials Is Sender Side And Receiver Side.**
9	**Design And Develop A Java Program For Implement Of The Data Link Layer Framing Method (Character String)**
10	**Design And Develop A Java Program For Implementation Of The Parity Check Bit (Even And Odd)**
11	**Write A Java Program To Implement The RSA Algorithm For String Encryption.**
12	**Write and execute a program playfair Cipher Substitution technique using Java.**

1a. Aim: creating a client-server application in Java involves writing two separate programs: one for the server and one for the client.

Description:

Server Program

The server program listens on a specified port and waits for client connections. Once a client connects, it reads a message from the client and sends a response.

Client Program

The client program connects to the server on the specified port, sends a message, and waits for the server's response.

Procedure:

Steps to Run the Programs

Compile the Server and Client Programs:

Run the Server Program:

Run the Client Program (in a separate terminal or command prompt):

When you run these programs, the server will start and wait for a client connection. The client will connect to the server, send a message, and receive a response. Both messages will be printed to their respective consoles.

Conclusion:

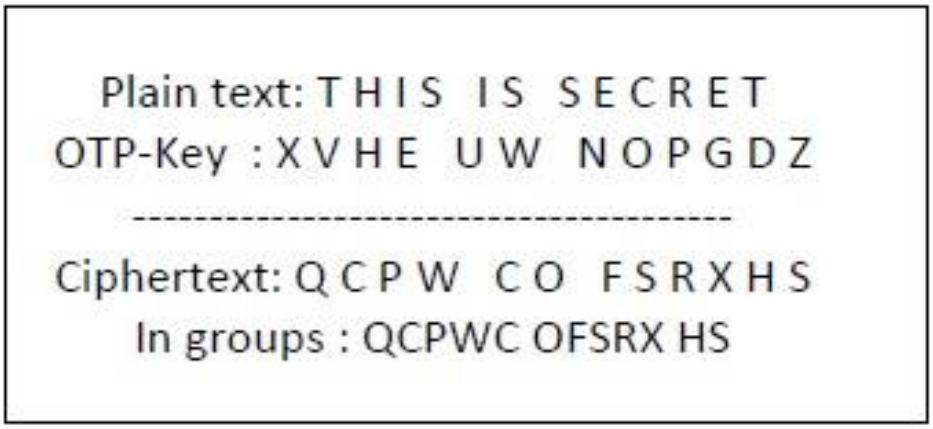
This is a simple example of a client-server application. For more complex applications, you might want to handle multiple clients, use more sophisticated communication protocols, or employ threading to handle concurrent connections.

2. AIM: Implement the data link layer framing methods such as One Time Pad using Java Program.

Description:

One-time pads are practical in situations where two parties in a secure environment must be able to depart from one another and communicate from two separate secure environments with perfect secrecy. The one-time-pad can be used in super encryption.

The primary use of padding with classical ciphers is to prevent the cryptanalyst from using that predictability to find known plaintext that aids in breaking the encryption. Random length padding also prevents an attacker from knowing the exact length of the plaintext message.



Conclusion:

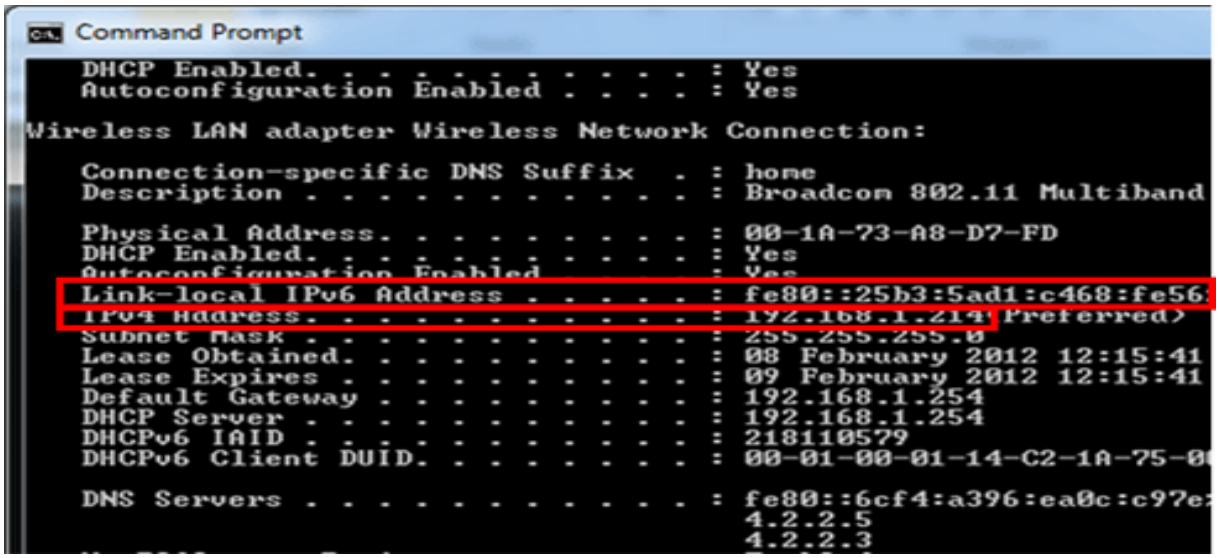
Finally the following One Time Pad program executed successfully using Java and output generated without any errors.

3. Aim: Implement the data link layer framing methods such as Find My PC IP Address using Java Program.

Description:

An IP(Internet Protocol) address is an identifier assigned to each computer and another device (e.g., router, mobile, etc) connected to a TCP/IP network that is used to locate and identify the node in communication with other nodes on the network. IP addresses are usually written and displayed in human-readable notation such as 192.168.1.35 in IPv4(32-bit IP address).

An IP address serves two principal functions: host or network interface identification and local addressing. Its role has been characterized as follows: “A name indicates what we seek. An address indicates where it is. A route indicates how to get there.”



Conclusion:

Finally the following Find my pc IP address program executed successfully using Java and output generated without any errors.

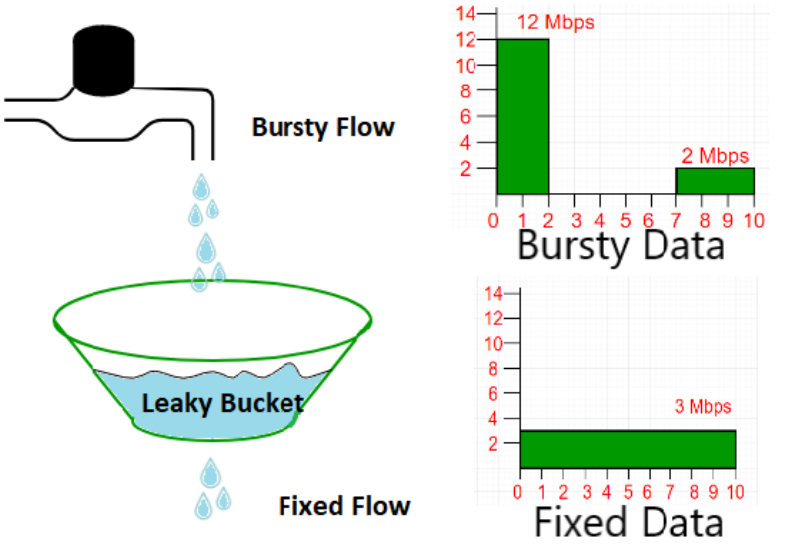
4.Aim: To write a Java program to implement the Leaky Bucket algorithm to smooth out bursty traffic and control network congestion.

Description:

The Leaky Bucket algorithm shapes bursty traffic into a steady, predictable output stream. The bucket represents a local buffer memory. Incoming data packets can arrive at random, erratic intervals (bursts). If the data flow exceeds the maximum buffer storage capacity, excess packets drop immediately. The bucket leaks data packets out at a fixed, constant transmission rate.

Algorithm:

- Define the bucket storage capacity (`bucket_size`) and the static transmission rate (`output_rate`).
- Input the number of execution steps or time intervals to simulate.
- At each time interval, generate a random number of incoming packets (bursty input).
- Check if the incoming packet size fits into the remaining storage of the bucket.
- If it fits, accept the packets into the bucket. Update the total storage counter.
- If it exceeds capacity, accept packets up to the maximum limit and drop the remaining excess traffic.
- Deduct the fixed `output_rate` from the bucket data to simulate transmission.
- If the bucket holds fewer packets than the output rate, empty the bucket completely.
- Repeat the cycle for all remaining time slices.



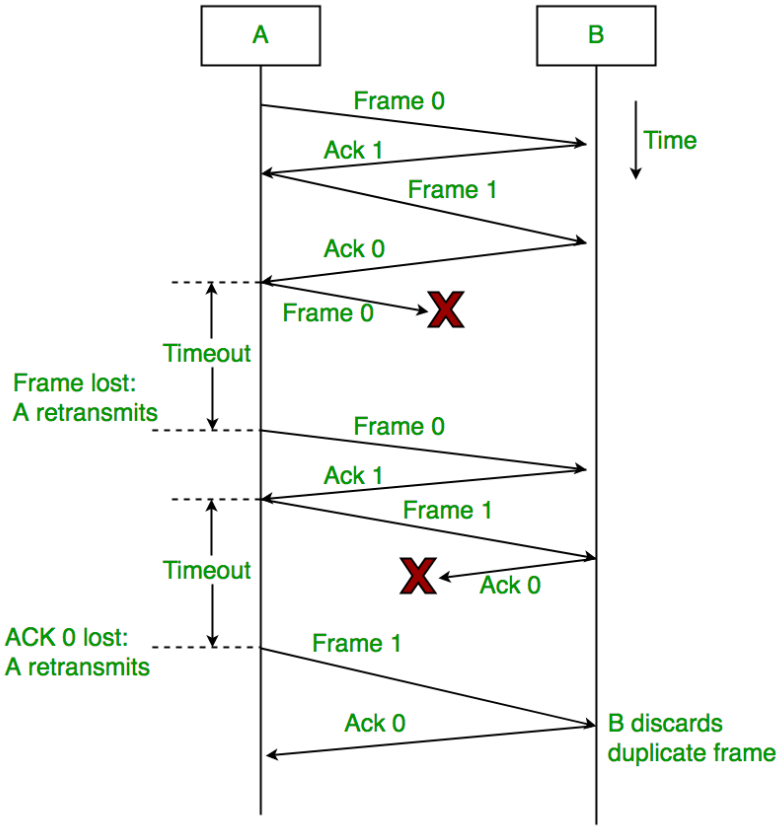
CONCLUSION

The Java program was successfully implemented to simulate the **Leaky Bucket algorithm** for controlling network congestion and smoothing bursty traffic. The program accepts incoming packets into the bucket according to its capacity and transmits them at a fixed rate. When the bucket becomes full, excess packets are discarded. Thus, the algorithm helps maintain a **steady and controlled data flow** in a network.

5. Aim: To write a Java program to simulate the Stop-and-Wait Sliding Window protocol for flow control.

Description:

Stop-and-Wait is a structural layer flow-control protocol. It ensures a high-speed sender does not overwhelm a slower destination node. The sender outputs a single numbered data unit (frame) and transitions into an idle state. It stays frozen until a positive acknowledgement packet (ACK) arrives. If a packet gets dropped by bad routing or ambient noise, an internal clock trigger fires a timeout, forcing a retransmission.



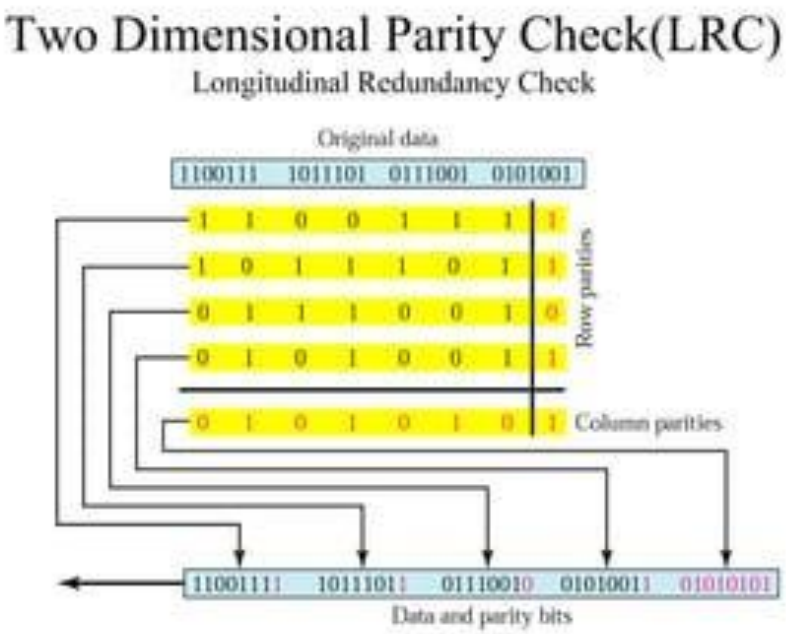
CONCLUSION

The Java program was successfully implemented to simulate the **Stop-and-Wait Sliding Window Protocol** for flow control. The protocol ensures that the sender transmits **one frame at a time** and waits for an acknowledgement from the receiver before sending the next frame. If an acknowledgement is not received within the specified time, the frame is retransmitted. Thus, the protocol provides **reliable data transmission and prevents the receiver from being overwhelmed**.

6.Aim: To write a Java program to implement the Two-Dimensional Redundancy Parity Check system to compute block check validation bits.

Description:

Standard 1D parity checks only count ones inside a single line slice, failing if an even number of bit shifts occur. A 2D structural approach organizes binary strings into blocks of specific rows and columns. Parity track equations are run horizontally across every individual row array row layer and vertically down every column structure column section. This two-axis checksum allows the receiver to detect burst failures and pinpoint the exact row-column intersection of a single-bit flip error for correction.



CONCLUSION

The Java program was successfully implemented to perform a **Two-Dimensional Redundancy Parity Check (2D Parity Check)**. The program calculates parity bits for each row and column of the binary data block and uses them to validate the received data. This method helps in **detecting and locating errors** in the transmitted data, particularly single-bit errors.

7. a) AIM: To write a Java program to perform Bit Stuffing and Bit Destuffing **on a given binary data.**

DESCRIPTION:

Bit Stuffing is a technique used in computer networks to insert an extra 0 bit after every sequence of **five consecutive 1 bits** in the data. This helps in identifying the beginning and ending of frames during data transmission.

In this program, the user enters binary data using the Scanner class. The program performs **bit stuffing** by inserting 0 after five consecutive 1s. It then performs **bit destuffing** by removing the inserted 0 and displays the stuffed and destuffed data.

ALGORITHM

1. Start the program.
2. Import the Scanner class for taking input from the user.
3. Read the binary data from the user.
4. Initialize a counter to count consecutive 1 bits.
5. Read each bit of the input data one by one.
6. If the bit is 1, increment the counter.
7. If the bit is 0, reset the counter to zero.
8. Append each bit to the stuffed data.
9. When five consecutive 1s are found, insert an additional 0 after them.
10. Display the stuffed data.
11. Read the stuffed data bit by bit for destuffing.
12. When five consecutive 1s followed by 0 are found, remove the extra 0.
13. Display the destuffed data.
14. Stop the program.

Conclusion:

The Java program was successfully executed to perform **Bit Stuffing and Bit Destuffing** on the given binary data.

7.b) AIM: To write a Java program to perform Character Stuffing **on a given string of data.**

DESCRIPTION

Character Stuffing is a technique used in computer networks to distinguish the actual data from the control characters used for framing. In this program, special characters are added to the data before transmission.

The program accepts a string from the user using the Scanner class. It adds the required control characters at the beginning and end of the data. If a special character sequence is found in the input, an additional escape character is inserted. Finally, the program displays the **original data** and the **transmitted data**.

ALGORITHM

1. Start the program.
2. Import the required Java packages.
3. Create a Scanner object to read input from the user.
4. Read the string from the user.
5. Find the length of the input string.
6. Display the original data.
7. Initialize the string with the starting control characters.
8. Read each character of the input string one by one.
9. Check whether the required special character sequence is present.
10. If the sequence is found, insert the escape sequence.
11. Append the current character to the transmitted data.
12. Add the ending control characters to the transmitted data.
13. Display the transmitted data.
14. Stop the program.

Conclusion:

The Java program was successfully executed to perform **Character Stuffing** on the given data and display the transmitted data.

8. AIM: To design and develop a Java program to implement **Cyclic Redundancy Check (CRC)** using CRC polynomials at the **sender side and receiver side** for detecting errors in transmitted data.

DESCRIPTION: Cyclic Redundancy Check (CRC) is an error-detection technique used in computer networks to detect errors in data transmission. In this program, the user enters a **generator polynomial** and the original data.

At the sender side, zeros are appended to the data according to the length of the generator polynomial. The program performs binary division using the generator and obtains the CRC remainder. This remainder is added to the original data to form the **transmitted code word**.

At the receiver side, the received code word is divided by the same generator polynomial. If the remainder is zero, the received data contains **no errors**. If the remainder is non-zero, the received data **contains errors**.

ALGORITHM

1. Start the program.
2. Read the generator polynomial from the user.
3. Read the data from the user.
4. Store the original data as the code.
5. Append zeros to the data until its length becomes equal to the data length plus the generator length minus one.
6. Perform binary division using the generator polynomial.
7. Obtain the CRC remainder.
8. Append the CRC remainder to the original data to form the transmitted code word.
9. Display the transmitted code word.
10. Read the received code word from the user.
11. Divide the received code word by the same generator polynomial.
12. Check the remainder obtained after division.
13. If the remainder is zero, display **"The received code word contains no errors."**
14. Otherwise, display **"The received code word contains errors."**
15. Stop the program.

Conclusion:

The Java program was successfully executed to implement **CRC error detection** at the sender and receiver sides. The program correctly identifies whether the received code word contains **errors or no errors**.

9. AIM: To design and develop a Java program to implement the **Data Link Layer Framing Method using Character String** by reading a string and displaying each character along with its position.

DESCRIPTION

The Data Link Layer is responsible for **framing the data** before it is transmitted over a network. In this program, a character string is taken as input from the user using the `Scanner` class.

The program reads the string character by character using the `charAt()` method. It checks each character and displays the **character and its position** in the string. Blank spaces are ignored. Thus, the program demonstrates the handling of characters in a string as part of a simple character-based framing method.

ALGORITHM

1. Start the program.
2. Import the `Scanner` class.
3. Create a `Scanner` object to read input from the user.
4. Read a string from the user using `nextLine()`.
5. Pass the string to the `PrintStringCharacters()` method.
6. Find the length of the string using the `length()` method.
7. Read each character using the `charAt()` method.
8. Check whether the character is a blank space.
9. If the character is not a blank space, display its position and character.
10. Repeat the process until all characters are processed.
11. Stop the program.

Conclusion:

The Java program was successfully executed to implement the **Data Link Layer Character String Framing Method** and display each character along with its position.

10. AIM: To design and develop a Java program to implement the **Parity Check Bit using Even Parity and Odd Parity** for detecting errors in transmitted binary data.

DESCRIPTION

Parity Check is an error-detection technique used in computer networks to detect errors during data transmission. In this method, an additional bit called the **parity bit** is added to the binary data.

In **Even Parity**, the parity bit is added so that the total number of `1`s in the transmitted data becomes even. In **Odd Parity**, the parity bit is added so that the total number of `1`s becomes odd.

In this program, the sender enters binary data and selects either even or odd parity. The program adds the appropriate parity bit to the data. At the receiver side, the received data is checked according to the selected parity. If the parity condition is satisfied, the program displays that there is **no error**; otherwise, it displays that an **error has been detected**.

ALGORITHM

1. Start the program.
2. Create an object of the `OddEvenParity` class.
3. At the sender side, read the binary data from the user.
4. Check whether the entered data contains only `0` and `1`.
5. Ask the user to select **Even Parity** or **Odd Parity**.
6. Count the number of `1`s in the binary data.
7. For Even Parity:
    - If the number of `1`s is even, append `0`.
    - If the number of `1`s is odd, append `1`.
8. For Odd Parity:
    - If the number of `1`s is odd, append `0`.
    - If the number of `1`s is even, append `1`.
9. Display the data with the parity bit.
10. At the receiver side, read the received binary data.
11. Count the number of `1`s in the received data.
12. For Even Parity, check whether the number of `1`s is even.
13. For Odd Parity, check whether the number of `1`s is odd.
14. If the required parity condition is satisfied, display **"Data received successfully with no error."**
15. Otherwise, display **"Error has been detected in received data."**
16. Stop the program.

Conclusion:

The Java program was successfully executed to implement **Even Parity and Odd Parity checking** and to detect errors in the received binary data.

11.AIM: To write a Java program to implement the **RSA algorithm for string encryption and decryption** using public and private keys.

DESCRIPTION

RSA is a **public-key cryptographic algorithm** used for encrypting and decrypting data. It uses two different keys: a **public key** for encryption and a **private key** for decryption.

In this program, two prime numbers p and q are entered by the user. The program calculates n and Euler's totient value φ. It then generates the public key e and private key d. The input string is converted into **ASCII values**, and each value is encrypted using the public key. The encrypted values are then decrypted using the private key and converted back into the original string.

ALGORITHM

1. Start the program.
2. Import the required Java packages.
3. Read two prime numbers `p` and `q` from the user.
4. Calculate `n = p × q`.
5. Calculate `φ = (p − 1) × (q − 1)`.
6. Find a value `e` such that `1 < e < φ` and `GCD(e, φ) = 1`.
7. Calculate the private key `d` such that `(d × e) mod φ = 1`.
8. Display the public key `e` and private key `d`.
9. Read the message as a string.
10. Convert each character of the string into its ASCII value.
11. Encrypt each ASCII value using the public key `(e, n)`.
12. Display the encrypted message.
13. Decrypt the encrypted values using the private key `(d, n)`.
14. Convert the decrypted ASCII values back into characters.
15. Display the original message after decryption.
16. Stop the program.

Conclusion:

The Java program was successfully executed to implement the **RSA algorithm for string encryption and decryption**, and the original message was successfully recovered after decryption.

12. AIM: To write and execute a Java program to implement the **Playfair Cipher Substitution Technique** for encrypting a given plaintext using a secret key.

DESCRIPTION:

The **Playfair Cipher** is a classical substitution cipher that encrypts plaintext in pairs of characters instead of encrypting individual characters. It uses a **5 × 5 matrix** generated from a given key. The letter `J` is usually combined with `I`, resulting in 25 characters in the matrix.

In this program, the user enters a key and plaintext. The program creates a 5 × 5 encryption matrix using the key and the remaining alphabet characters. The plaintext is divided into pairs of characters. If two consecutive characters are the same, an `X` is inserted between them. If the plaintext has an odd number of characters, an `X` is added at the end.

Each pair is encrypted according to the Playfair rules: characters in the same row are replaced by the characters to their right, characters in the same column are replaced by the characters below them, and characters forming a rectangle are replaced by the characters at the opposite corners. Finally, the encrypted text is displayed as the ciphertext.

ALGORITHM

1. Start the program.
2. Import the required Java packages.
3. Read the key from the user.
4. Convert the key to uppercase and replace `J` with `I`.
5. Read the plaintext from the user.
6. Convert the plaintext to uppercase and replace `J` with `I`.
7. Format the plaintext into pairs of characters.
8. Insert `X` between repeated characters in a pair.
9. If the plaintext has an odd number of characters, append `X`.
10. Create a 5 × 5 matrix using the characters of the key.
11. Fill the remaining positions with unused letters of the alphabet.
12. Display the encryption matrix.
13. Take the plaintext two characters at a time.
14. Find the positions of both characters in the matrix.
15. If both characters are in the same row, replace each with the character to its right.
16. If both characters are in the same column, replace each with the character below it.
17. Otherwise, treat the two characters as opposite corners of a rectangle and replace them with the characters at the other corners.
18. Repeat the process for all character pairs.
19. Display the resulting ciphertext.
20. Stop the program.

Conclusion:

The Java program was successfully executed to implement the **Playfair Cipher Substitution Technique** and generate the encrypted ciphertext for the given plaintext and key.