**Client Class:**

We created this code to simulate a simple client program that sends a message to a server and measures the time it takes to receive a response.

Imports:

java.io - Provides classes for input and output operations, including reading and writing data streams.

java.net - Provides classes for networking functionalities, including sockets.

Client Class:

The main method is the program's entry point.

Connection Setup:

Creates a Socket object by connecting to "localhost" (usually the same machine) on port 4000. This establishes a communication channel with the server.

Creates a PrintWriter to send data (the message) through the socket's output stream.

Creates a BufferedReader to read data (the response) from the socket's input stream.

Sending Message:

Defines a string containing the message to be sent ("Amdahl's Law..." statement).

Uses out.println to send the message to the server via the socket.

Prints a confirmation message indicating what was sent.

Measuring Time:

Captures the starting time using System.nanoTime before any code to be timed is executed.

Places a placeholder for code that might take some time to execute (commented out).

Captures the ending time using System.nanoTime after the (potential) code execution.

Calculates the duration in nanoseconds by subtracting the starting time from the ending time.

Prints the duration in both nanoseconds and milliseconds.

Receiving Response:

Uses in.readLine to read a single line of response sent by the server.

Prints a confirmation message indicating what was received.

Closing Connection:

Closes the socket to terminate the communication.

Note: This program assumes a server is already running on port 4000 and listening for messages. The provided code snippet only focuses on the client-side communication.

**MatrixUtils.Java**

The code defines a utility class, MatrixUtils, that provides static methods for matrix operations. Important methods associated with this class include:

add Method: Adds two square matrices A and B element by element and returns the resulting matrix.

subtract Method: Subtracts one square matrix B from another square matrix A element by element and returns the resulting matrix.

split Method: Extracts a submatrix from a larger matrix matrix, starting at the specified row and col indices, and of the specified size. The extracted submatrix is returned.

join Method: Merges a smaller matrix C into a larger matrix P at the specified row and col position.

Notes:

These methods assume square matrices (equal number of rows and columns).

The methods use System.arraycopy for efficient array copying where applicable.

**Router.Java**

The code implements a simple router in Java that facilitates communication between a client and a server.

ServerSocket Initialization:

The router listens for client connections on port 4000 using a ServerSocket.

Main Loop:

The router operates in an infinite loop, accepting incoming client connections.

Client Communication:

Once a client connects, the router establishes input and output streams (BufferedReader and PrintWriter) to read messages from and send responses back to the client.

Server Communication:

The router establishes a connection to the server on port 5000 using a Socket, with corresponding input and output streams.

Message Forwarding:

The router:

Reads a message from the client.

Forwards the message to the server.

Receives the server's response.

Sends the server's response back to the client.

Connection Management:

After each interaction, the router closes both the client and server sockets.

Key Points:

Port Configuration: The router listens on port 4000 and forwards messages to a server on port 5000.

Middleware Role: Acts as an intermediary for message forwarding between a client and a server.

Synchronous Communication: Handles one client request-response cycle at a time (no threading or concurrency).

**Server.Java**

The code implements a simple server in Java that listens for client connections via a router and processes client messages.

Server Initialization:

The server listens for incoming connections on port 5000 using a ServerSocket.

Main Loop:

The server runs in an infinite loop, waiting to accept client connections.

Client Communication:

When a client (via the router) connects, the server:

Establishes input and output streams (BufferedReader and PrintWriter) for communication.

Reads a message sent by the client.

Message Processing:

The server:

Processes the client's message by converting it to uppercase.

Responds with a confirmation message, including the uppercase version of the client's input.

Connection Management:

After each interaction, the server closes the socket connection with the client.

Key Points:

Port Configuration: The server listens on port 5000 for messages relayed by the router.

Response Logic: The server acknowledges the client message and transforms it to uppercase before sending it back.

Synchronous Operation: Handles one client request at a time without concurrency or threading.

**Strassen.Java**

We created this code to implement Strassen's Algorithm for matrix multiplication, which is an efficient method to multiply two square matrices.

Base Case:

If the matrix size (n) is 1 (a single element), the matrices are directly multiplied.

Matrix Splitting:

For larger matrices, the algorithm divides each input matrix (A and B) into four equally-sized submatrices (A11, A12, A21, A22 for A and similarly for B) using the MatrixUtils.split method.

Strassen's Formula:

The algorithm computes seven intermediate matrices (M1 to M7) using a combination of matrix addition, subtraction, and recursive multiplication:

* 𝑀1=(𝐴11+𝐴22)⋅(𝐵11+𝐵22)
* 𝑀2=(𝐴21+𝐴22)⋅𝐵11
* M3=A11⋅(B12−B22)
* M4=A22⋅(B21−B11)
* M5=(A11+A12)⋅B22
* M6=(A21−A11)⋅(B11+B12)
* M7=(A12−A22)⋅(B21+B22)

Combining Results:

The final result matrix (result) is constructed by combining four submatrices (C11, C12, C21, C22) derived from M1 to M7:

* C11=M1+M4−M5+M7
* C12=M3+M5
* C21=M2+M4
* C22=M1+M3−M2+M6

The submatrices are joined into the final matrix using the MatrixUtils.join method.

Recursive Calls:

The method is applied recursively to submatrices until the base case is reached.

Key Points:

Efficient Algorithm: Strassen's algorithm reduces the complexity of matrix multiplication from

𝑂(𝑛^3) to approximately 𝑂(𝑛^2.81).

Recursive Design: The method breaks down larger matrices into smaller subproblems using divide-and-conquer.

Dependencies: Utilizes helper methods (split, join, add, and subtract) from the MatrixUtils class.

**StrassenMultiplication.Java**

The code implements a main class that demonstrates the use of Strassen's Algorithm to multiply two matrices.

Matrix Definition:

Two 4x4 matrices, matrixA and matrixB, are initialized with predefined values.

Matrix Multiplication:

The Strassen.multiply method is called with matrixA and matrixB as inputs. This performs the multiplication using Strassen's algorithm and returns the resulting matrix.

Result Display:

The resulting matrix is printed to the console in a formatted manner, row by row.

Key Points:

Input Matrices:

matrixA contains sequential integers from 1 to 16.

matrixB contains descending integers from 16 to 1.

Output:

The program computes and prints the product of matrixA and matrixB using the Strassen algorithm.

Purpose:

Serves as a test or demonstration of the Strassen class functionality for matrix multiplication.