//1 a: tcp udp

//1 b: rmi

//4 ring bully algorithm

//6 publisher subscriber

//7 Berkeley algorithm

//8 tic tac toe game

//1 a) to develop any distributed application through implementing client-server communication //programs based on Java Sockets.

//1 b) to develop any d. a. through impl. Client-server communication programs based on java rmi.

//2 to develop any d. a. using mpi message passing interface

//3 to develop any d. a. with CORBA program using javaide.

//4 to develop any d. algo. For leader election.

//5 to create web service and write any d. app. To consume the web service.

//6 to develop any d. app. Using messaging system in publisher subscriber paradigm.

//7 implement bekeley algorithm for clock synchronization.

//implement tic tac toe game

//1 to develop any distributed application through implementing client-server communication //programs based on Java Sockets.

//Client.java

import java.net.\*;

import java.io.\*;

public class Client {

    private Socket socket = null;

    private BufferedReader consoleInput = null;

    private DataOutputStream out = null;

    private DataInputStream in = null;

    public Client(String address, int port) {

        try {

            // Establish a connection

            socket = new Socket(address, port);

            System.out.println("Connected to server");

            // Input for terminal and server messages

            consoleInput = new BufferedReader(new InputStreamReader(System.in));

            in = new DataInputStream(socket.getInputStream());

            // Output to the server

            out = new DataOutputStream(socket.getOutputStream());

            String messageToSend = "", messageReceived = "";

            // Start communication loop

            while (!messageToSend.equalsIgnoreCase("Over")) {

                // Read message from console

                System.out.print("Enter message: ");

                messageToSend = consoleInput.readLine();

                out.writeUTF(messageToSend); // Send to server

                // Receive and print message from server

                messageReceived = in.readUTF();

                System.out.println("Server: " + messageReceived);

            }

        } catch (IOException i) {

            System.out.println("I/O error: " + i.getMessage());

        } finally {

            // Close connections

            try {

                if (consoleInput != null) consoleInput.close();

                if (out != null) out.close();

                if (in != null) in.close();

                if (socket != null) socket.close();

            } catch (IOException i) {

                System.out.println("Error closing connection: " + i.getMessage());

            }

        }

    }

    public static void main(String[] args) {

        new Client("127.0.0.1", 5000);

    }

}

//Server.java

import java.net.\*;

import java.io.\*;

public class Server {

    private Socket socket = null;

    private ServerSocket server = null;

    private DataInputStream in = null;

    private DataOutputStream out = null;

    private BufferedReader consoleInput = null;

    public Server(int port) {

        try {

            // Start server and wait for connection

            server = new ServerSocket(port);

            System.out.println("Server started");

            System.out.println("Waiting for a client ...");

            // Accept connection

            socket = server.accept();

            System.out.println("Client connected");

            // Input from client and terminal

            in = new DataInputStream(new BufferedInputStream(socket.getInputStream()));

            consoleInput = new BufferedReader(new InputStreamReader(System.in));

            // Output to client

            out = new DataOutputStream(socket.getOutputStream());

            String messageFromClient = "", messageToSend = "";

            // Start communication loop

            while (!messageFromClient.equalsIgnoreCase("Over")) {

                // Receive and print message from client

                messageFromClient = in.readUTF();

                System.out.println("Client: " + messageFromClient);

                // Send a response to client

                System.out.print("Enter message: ");

                messageToSend = consoleInput.readLine();

                out.writeUTF(messageToSend);

            }

        } catch (IOException i) {

            System.out.println("I/O error: " + i.getMessage());

        } finally {

            // Close connections

            try {

                if (in != null) in.close();

                if (out != null) out.close();

                if (consoleInput != null) consoleInput.close();

                if (socket != null) socket.close();

                if (server != null) server.close();

            } catch (IOException i) {

                System.out.println("Error closing connection: " + i.getMessage());

            }

        }

    }

    public static void main(String[] args) {

        new Server(5000);

    }

}

////////////////////////////////////////////////////////////////////////////////////////////

//1 a program to convert to uppercase

//Client.java

package Uppercase;

import java.net.\*;

import java.io.\*;

public class Client {

    // Initialize socket and input-output streams

    private Socket socket = null;

    private BufferedReader input = null;

    private DataOutputStream out = null;

    // Constructor to set IP address and port

    public Client(String address, int port) {

        // Establish a connection

        try {

            socket = new Socket(address, port);

            System.out.println("Connected");

            // Takes input from the terminal using BufferedReader

            input = new BufferedReader(new InputStreamReader(System.in));

            // Sends output to the socket

            out = new DataOutputStream(socket.getOutputStream());

        } catch (UnknownHostException u) {

            System.out.println(u);

        } catch (IOException i) {

            System.out.println(i);

        }

        String line = "";

        // Keep reading until "Over" is input

        while (!line.equals("Over")) {

            try {

                // Read message from input using BufferedReader

                line = input.readLine();

                out.writeUTF(line); // Send message to the server

            } catch (IOException i) {

                System.out.println(i);

            }

        }

        // Close the connection

        try {

            input.close();

            out.close();

            socket.close();

        } catch (IOException i) {

            System.out.println(i);

        }

    }

    public static void main(String args[]) {

        // Starts the client and initiates the connection

        new Client("127.0.0.1", 5000); // Connects to the server on port 5000

    }

}

//////////////////////////////////////////

//Server.java

package Uppercase;

import java.net.\*;

import java.io.\*;

public class Server {

    // Initialize socket and input stream

    private Socket socket = null;

    private ServerSocket server = null;

    private DataInputStream in = null;

    // Constructor with port

    public Server(int port) {

        // Starts server and waits for a connection

        try {

            server = new ServerSocket(port);

            System.out.println("Server started");

            System.out.println("Waiting for a client ...");

            socket = server.accept();

            System.out.println("Client accepted");

            // Takes input from the client socket

            in = new DataInputStream(new BufferedInputStream(socket.getInputStream()));

            String line = "";

            // Read messages from client until "Over" is sent

            while (!line.equals("Over")) {

                try {

                    line = in.readUTF();

                    System.out.println(line.toUpperCase()); // Prints the message in uppercase

                } catch (IOException i) {

                    System.out.println(i);

                }

            }

            System.out.println("Closing connection");

            // Close the connection

            socket.close();

            in.close();

        } catch (IOException i) {

            System.out.println(i);

        }

    }

    public static void main(String args[]) {

        // Starts the server on port 5000

        Server server = new Server(5000);

    }

}

//Output:

//Server started

//Waiting for a client ...

//Client accepted

//HI

//HELLO

//Connected

//hi

//hello

//////////////////////////////////////////////////////////////////////////////////

//2 to develop any distributed application using message passing interface(mpi).

import mpi.MPI;

public class mpj {

    public static void main(String args[]) {

        // Initialize MPI execution environment

        MPI.Init(args);

        // Get the id of the process

        int rank = MPI.COMM\_WORLD.Rank();

        // Total number of processes

        int size = MPI.COMM\_WORLD.Size();

        int root = 0; // Root process id

        int sendbuf[] = null; // Array to be scattered

        if (rank == root) {

            // Initialize the data to be scattered at the root process

            sendbuf = new int[size];

            sendbuf[0] = 10;

            sendbuf[1] = 20;

            sendbuf[2] = 30;

            sendbuf[3] = 40;

            System.out.print("Processor " + rank + " has data: ");

            for (int i = 0; i < size; i++) {

                System.out.print(sendbuf[i] + " ");

            }

            System.out.println();

        }

        // Buffer to receive scattered data

        int recvbuf[] = new int[1];

        // Scatter operation: distribute data from root to all processes

        MPI.COMM\_WORLD.Scatter(sendbuf, 0, 1, MPI.INT, recvbuf, 0, 1, MPI.INT, root);

        // Each process displays its received data

        System.out.println("Processor " + rank + " received data: " + recvbuf[0]);

        // Each process doubles its received data

        System.out.println("Processor " + rank + " is doubling the data.");

        recvbuf[0] \*= 2;

        // Gather operation: collect the modified data at the root process

        MPI.COMM\_WORLD.Gather(recvbuf, 0, 1, MPI.INT, sendbuf, 0, 1, MPI.INT, root);

        // Root process displays the gathered data

        if (rank == root) {

            System.out.println("Processor " + rank + " has gathered data: ");

            for (int i = 0; i < size; i++) {

                System.out.print(sendbuf[i] + " ");

            }

            System.out.println();

        }

        // Terminate MPI execution environment

        MPI.Finalize();

    }

}

//output:

/\* PS C:\Users\LENOVO> & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\LENOVO\AppData\Local\Temp\vscodesws\_ca47e\jdt\_ws\jdt.ls-java-project\bin' 'mpj'

Exception in thread "main" java.lang.Error: Unresolved compilation problems:

MPI cannot be resolved

MPI cannot be resolved

MPI cannot be resolved

MPI cannot be resolved

MPI cannot be resolved to a variable

MPI cannot be resolved to a variable

MPI cannot be resolved

MPI cannot be resolved to a variable

MPI cannot be resolved to a variable

MPI cannot be resolved

at mpj.main(mpj.java:6) \*/