**2**

import javax.swing.\*; // Import for GUI components (not used here but included)

import java.net.\*; // Import for networking (Socket, etc.)

import java.awt.image.\*; // Import for BufferedImage

import javax.imageio.\*; // Import for ImageIO (reading/writing images)

import java.io.\*; // Import for InputStream, OutputStream, File, etc.

public class Client {

public static void main(String args[]) throws Exception {

Socket soc; // Declare a socket to connect to the server

BufferedImage img = null; // Declare a BufferedImage to store the image

soc = new Socket("localhost", 4000); // Connect to server on localhost, port 4000

System.out.println("Client is running."); // Print client status

try {

System.out.println("Reading image from disk."); // Notify reading image

img = ImageIO.read(new File("digital\_image\_processing.jpg")); // Load image file into BufferedImage

ByteArrayOutputStream baos = new ByteArrayOutputStream(); // Create a byte stream to hold image data

ImageIO.write(img, "jpg", baos); // Write the image to the byte array in JPG format

baos.flush(); // Flush the stream to ensure all data is written

byte[] bytes = baos.toByteArray(); // Convert the stream into a byte array

baos.close(); // Close the ByteArrayOutputStream

System.out.println("Sending image to server."); // Notify sending image

OutputStream out = soc.getOutputStream(); // Get the socket's output stream

DataOutputStream dos = new DataOutputStream(out); // Wrap it in a DataOutputStream to send data easily

dos.writeInt(bytes.length); // First send the length of the byte array to the server

dos.write(bytes, 0, bytes.length); // Then send the image byte data

System.out.println("Image sent to server."); // Confirm image sent

dos.close(); // Close the DataOutputStream

out.close(); // Close the underlying OutputStream

} catch (Exception e) {

System.out.println("Exception: " + e.getMessage()); // Handle any exceptions by printing the message

}

soc.close(); // Close the socket connection

}

}

import java.net.\*; // Import for ServerSocket and Socket

import java.io.\*; // Import for Input/Output streams

import java.awt.image.\*; // Import for BufferedImage

import javax.imageio.\*; // Import for reading image from stream

import javax.swing.\*; // Import for GUI components to display the image

class Server {

public static void main(String args[]) throws Exception {

ServerSocket server = null; // Declare a ServerSocket

Socket socket; // Declare a Socket to accept the client connection

server = new ServerSocket(4000); // Bind server to port 4000

System.out.println("Server Waiting for image"); // Print waiting status

socket = server.accept(); // Wait and accept a client connection

System.out.println("Client connected."); // Print when client connects

InputStream in = socket.getInputStream(); // Get input stream from socket

DataInputStream dis = new DataInputStream(in); // Wrap it in a DataInputStream to read primitives

int len = dis.readInt(); // Read the length of the incoming image byte array

System.out.println("Image Size: " + len / 1024 + "KB"); // Print image size in KB

byte[] data = new byte[len]; // Create a byte array to hold image data

dis.readFully(data); // Read the full byte array from the client

dis.close(); // Close the DataInputStream

in.close(); // Close the InputStream

InputStream ian = new ByteArrayInputStream(data); // Create input stream from the byte array

BufferedImage bImage = ImageIO.read(ian); // Read the image from byte array into BufferedImage

JFrame f = new JFrame("Server"); // Create a JFrame window with title "Server"

ImageIcon icon = new ImageIcon(bImage); // Wrap the BufferedImage in an ImageIcon

JLabel l = new JLabel(); // Create a label to hold the image

l.setIcon(icon); // Set the icon (image) on the label

f.add(l); // Add the label to the frame

f.pack(); // Resize the window to fit the image

f.setVisible(true); // Make the window visible

}

}

**3a**

import java.io.\*; // Import for InputStream, OutputStream, DataInputStream, PrintStream

import java.net.\*; // Import for Socket

public class eclient {

public static void main(String args[]) {

Socket c = null; // Declare a socket for connecting to server

String line; // Declare a string to store user input

DataInputStream is, is1; // is for user input, is1 for server response

PrintStream os; // Output stream to send data to server

try {

c = new Socket("localhost", 8080); // Connect to server on localhost at port 8080

} catch(IOException e) {

System.out.println(e); // Print exception if connection fails

}

try {

os = new PrintStream(c.getOutputStream()); // Get output stream to write to server

is = new DataInputStream(System.in); // Input stream to read from console

is1 = new DataInputStream(c.getInputStream()); // Input stream to read response from server

do {

System.out.println("client"); // Prompt client to enter input

line = is.readLine(); // Read input from user

os.println(line); // Send input to server

if (!line.equals("exit")) // If input is not "exit", read server response

System.out.println("server:" + is1.readLine()); // Print response from server

} while (!line.equals("exit")); // Repeat until "exit" is entered

} catch(IOException e) {

System.out.println("socket closed"); // Print if any I/O error occurs

}

}

}

import java.io.\*; // Import for I/O operations (DataInputStream, PrintStream, etc.)

import java.net.\*; // Import for networking classes (ServerSocket, Socket)

import java.lang.\*; // (Optional here) used for standard language classes like String, Exception (already imported by default)

public class eserver {

public static void main(String args[]) throws IOException {

ServerSocket s = null; // Declare ServerSocket to listen for clients

String line; // Variable to hold messages from the client

DataInputStream is; // Input stream to receive data from client

PrintStream ps; // Output stream to send data to client

Socket c = null; // Socket to communicate with client

try {

s = new ServerSocket(8080); // Start the server on port 8080

} catch(IOException e) {

System.out.println(e); // Print error if port is unavailable or other issue

}

try {

c = s.accept(); // Accept client connection

is = new DataInputStream(c.getInputStream()); // Create input stream from client

ps = new PrintStream(c.getOutputStream()); // Create output stream to client

while(true) {

line = is.readLine(); // Read message from client

System.out.println("msg received and sent back to client"); // Log to server console

ps.println(line); // Echo the message back to client

}

} catch(IOException e) {

System.out.println(e); // Handle I/O exceptions during communication

}

}

}

**3b**

import java.net.\*; // Import networking classes (ServerSocket, Socket)

import java.io.\*; // Import I/O classes (DataInputStream, PrintStream)

public class TCPserver1 {

public static void main(String arg[]) {

ServerSocket s = null; // ServerSocket to listen for incoming connections

String line; // Variable to store messages

DataInputStream is = null, is1 = null; // 'is' for client input, 'is1' for server console input

PrintStream os = null; // Output stream to send data to client

Socket c = null; // Socket for communication with the client

try {

s = new ServerSocket(9999); // Create a server socket on port 9999

} catch(IOException e) {

System.out.println(e); // Print exception if server fails to start

}

try {

c = s.accept(); // Wait for a client to connect

is = new DataInputStream(c.getInputStream()); // Input from client

is1 = new DataInputStream(System.in); // Input from server (keyboard)

os = new PrintStream(c.getOutputStream()); // Output to client

do {

line = is.readLine(); // Read message from client

System.out.println("Client:" + line); // Display client's message

System.out.println("Server:"); // Prompt for server's response

line = is1.readLine(); // Read server's response from keyboard

os.println(line); // Send server's response to client

}

while(line.equalsIgnoreCase("quit") == false); // Continue until "quit" is typed

is.close(); // Close input stream from client

os.close(); // Close output stream to client

} catch(IOException e) {

System.out.println(e); // Print I/O error if any

}

}

}

import java.net.\*; // Import networking classes (Socket)

import java.io.\*; // Import I/O classes (DataInputStream, PrintStream)

public class TCPclient1 {

public static void main(String arg[]) {

Socket c = null; // Declare client socket

String line; // Variable to hold input line

DataInputStream is, is1; // 'is' for keyboard input, 'is1' for server response

PrintStream os; // Output stream to send data to server

try {

c = new Socket("10.0.200.36", 9999); // Connect to server at given IP and port

} catch(IOException e) {

System.out.println(e); // Print if connection fails

}

try {

os = new PrintStream(c.getOutputStream()); // Get output stream to send data to server

is = new DataInputStream(System.in); // Get input stream from keyboard

is1 = new DataInputStream(c.getInputStream()); // Get input stream from server

do {

System.out.println("Client:"); // Prompt for input

line = is.readLine(); // Read user input

os.println(line); // Send input to server

System.out.println("Server:" + is1.readLine()); // Read and print server's response

}

while(line.equalsIgnoreCase("quit") == false); // Repeat until "quit" is typed

is1.close(); // Close input stream from server

os.close(); // Close output stream to server

} catch(IOException e) {

System.out.println("Socket Closed! Message Passing is over"); // Inform when connection ends

}

}

}

**3c**

import java.net.\*; // For Socket

import java.io.\*; // For I/O operations

public class FileClient {

public static void main (String [] args) throws IOException {

int filesize = 6022386; // Approximate file size in bytes (just a limit buffer)

long start = System.currentTimeMillis(); // Capture start time

int bytesRead;

int current = 0;

Socket sock = new Socket("127.0.0.1", 13267); // Connect to server at port 13267

System.out.println("Connecting...");

byte [] mybytearray = new byte [filesize]; // Buffer to hold file data

InputStream is = sock.getInputStream(); // Get input stream from socket

FileOutputStream fos = new FileOutputStream("source-copy.pdf"); // Create file to save received data

BufferedOutputStream bos = new BufferedOutputStream(fos); // Use buffered stream for performance

bytesRead = is.read(mybytearray, 0, mybytearray.length); // Read first chunk

current = bytesRead;

// Keep reading till the end of the stream

do {

bytesRead = is.read(mybytearray, current, (mybytearray.length - current));

if (bytesRead >= 0) current += bytesRead;

} while (bytesRead > -1);

bos.write(mybytearray, 0 , current); // Write all received bytes to file

bos.flush(); // Ensure all data is written

long end = System.currentTimeMillis(); // Capture end time

System.out.println(end - start); // Print time taken in ms

bos.close(); // Close streams

sock.close(); // Close socket

}

}

import java.net.\*; // For networking classes like ServerSocket and Socket

import java.io.\*; // For input/output classes

public class FileServer {

public static void main (String [] args) throws IOException {

// Create a server socket listening on port 13267

ServerSocket servsock = new ServerSocket(13267);

while (true) { // Infinite loop to keep accepting new connections

System.out.println("Waiting...");

Socket sock = servsock.accept(); // Accept client connection

System.out.println("Accepted connection : " + sock);

// Load the file you want to send

File myFile = new File("source.pdf");

// Create a byte array the size of the file

byte [] mybytearray = new byte[(int) myFile.length()];

// Read file data into the byte array

FileInputStream fis = new FileInputStream(myFile);

BufferedInputStream bis = new BufferedInputStream(fis);

bis.read(mybytearray, 0, mybytearray.length);

// Get the output stream of the socket to send data

OutputStream os = sock.getOutputStream();

System.out.println("Sending...");

// Send the byte array to the client

os.write(mybytearray, 0, mybytearray.length);

os.flush(); // Ensure all data is sent

// Close the connection with this client

sock.close();

}

}

}

**4**

import java.io.\*; // For I/O classes

import java.net.\*; // For networking classes

public class udpdnsserver {

// Utility method to find index of a string in an array

private static int indexOf(String[] array, String str) {

str = str.trim(); // Remove leading/trailing whitespace

for (int i = 0; i < array.length; i++) {

if (array[i].equals(str))

return i; // Return index if match found

}

return -1; // Return -1 if not found

}

public static void main(String arg[]) throws IOException {

// List of predefined hostnames

String[] hosts = {"yahoo.com", "gmail.com", "cricinfo.com", "facebook.com"};

// Corresponding IP addresses

String[] ip = {"68.180.206.184", "209.85.148.19", "80.168.92.140", "69.63.189.16"};

System.out.println("Press Ctrl + C to Quit");

// Infinite loop to keep the server running

while (true) {

// Create a new UDP socket bound to port 1362

DatagramSocket serversocket = new DatagramSocket(1362);

// Buffers to store incoming and outgoing data

byte[] senddata = new byte[1021];

byte[] receivedata = new byte[1021];

// Prepare packet for receiving data

DatagramPacket recvpack = new DatagramPacket(receivedata, receivedata.length);

serversocket.receive(recvpack); // Receive client request

// Extract the requested hostname from the received data

String sen = new String(recvpack.getData()).trim();

// Get client IP address and port number

InetAddress ipaddress = recvpack.getAddress();

int port = recvpack.getPort();

String capsent; // Response to send back to client

System.out.println("Request for host " + sen);

// Lookup host in predefined array

if (indexOf(hosts, sen) != -1)

capsent = ip[indexOf(hosts, sen)]; // Send corresponding IP

else

capsent = "Host Not Found"; // If not found

// Convert response to bytes

senddata = capsent.getBytes();

// Create a response packet with IP and port info of client

DatagramPacket pack = new DatagramPacket(senddata, senddata.length, ipaddress, port);

// Send the response back to client

serversocket.send(pack);

// Close the server socket after handling one request

serversocket.close();

}

}

}

import java.io.\*; // For BufferedReader, InputStreamReader

import java.net.\*; // For networking classes

public class udpdnsclient {

public static void main(String args[]) throws IOException {

// To read input from user via keyboard

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

// Create a UDP socket for the client

DatagramSocket clientsocket = new DatagramSocket();

InetAddress ipaddress;

// Determine server IP address: localhost by default, or from command line

if (args.length == 0)

ipaddress = InetAddress.getLocalHost();

else

ipaddress = InetAddress.getByName(args[0]);

// Create byte arrays for sending and receiving data

byte[] senddata = new byte[1024];

byte[] receivedata = new byte[1024];

// Set the port number where the server is listening

int portaddr = 1362;

// Get the hostname input from the user

System.out.print("Enter the hostname : ");

String sentence = br.readLine();

// Convert hostname string to byte array

senddata = sentence.getBytes(); // FIXED: corrected typo from 'Senddata' to 'senddata'

// Create a packet to send the data to the server

DatagramPacket pack = new DatagramPacket(senddata, senddata.length, ipaddress, portaddr);

// Send the packet

clientsocket.send(pack);

// Prepare a packet to receive the response from the server

DatagramPacket recvpack = new DatagramPacket(receivedata, receivedata.length);

// Receive the server's response

clientsocket.receive(recvpack);

// Convert the response from byte array to string

String modified = new String(recvpack.getData()).trim();

// Print the received IP address or "Host Not Found"

System.out.println("IP Address: " + modified);

// Close the socket

clientsocket.close();

}

}

**5a**

import java.io.\*; // For InputStream, BufferedReader, etc.

import java.net.\*; // For Socket and networking classes

class ArpClient {

public static void main(String args[]) throws IOException {

try {

// Create a socket to connect to the ARP server on localhost at port 1100

Socket ss = new Socket(InetAddress.getLocalHost(), 1100);

// To send data to server

PrintStream ps = new PrintStream(ss.getOutputStream());

// To read input from the user

BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); // FIXED: added space between 'new' and 'InputStreamReader'

String ip;

System.out.println("Enter the IPADDRESS:");

ip = br.readLine(); // Read IP address input from user

ps.println(ip); // Send IP address to the server

String str;

// To receive data from the server

BufferedReader br2 = new BufferedReader(new InputStreamReader(ss.getInputStream())); // FIXED: added space between 'new' and 'InputStreamReader'

System.out.println("ARP From Server::");

// Print the server's response line by line until "end" is received

do {

str = br2.readLine();

System.out.println(str);

} while (!(str.equalsIgnoreCase("end")));

} catch (IOException e) {

System.out.println("Error: " + e); // Print any IO error

}

}

}

import java.io.\*; // For I/O streams and readers

import java.net.\*; // For networking classes

class ArpServer {

public static void main(String args[]) throws IOException {

try {

// Create a ServerSocket on port 1100

ServerSocket ss = new ServerSocket(1100);

// Accept incoming client connection

Socket s = ss.accept();

// Output stream to send data back to client

PrintStream ps = new PrintStream(s.getOutputStream());

// Input stream to read IP address sent by client

BufferedReader br1 = new BufferedReader(new InputStreamReader(s.getInputStream())); // FIXED: added space between 'new' and 'InputStreamReader'

String ip;

ip = br1.readLine(); // Read the IP address from the client

// Get the runtime environment to execute system commands

Runtime r = Runtime.getRuntime();

// Execute the arp command to get MAC address mapping

Process p = r.exec("arp -a " + ip);

// Read the output of the executed command

BufferedReader br2 = new BufferedReader(new InputStreamReader(p.getInputStream())); // FIXED: added space between 'new' and 'InputStreamReader'

String str;

while ((str = br2.readLine()) != null) {

// Send each line of arp output to the client

ps.println(str);

}

// Optionally, send end indicator

ps.println("end");

} catch (IOException e) {

// Print any exceptions

System.out.println("Error: " + e); // FIXED: added space in "Error: "

}

}

}

**5b**

import java.io.\*; // For input/output streams

import java.net.\*; // For DatagramSocket, DatagramPacket, InetAddress

import java.util.\*; // For utility classes (not directly used here)

class Clientrarp12 {

public static void main(String args[]) {

try {

// Create a UDP socket for the client

DatagramSocket client = new DatagramSocket();

// Set server address to localhost

InetAddress addr = InetAddress.getByName("127.0.0.1");

// Buffers to send and receive data

byte[] sendbyte = new byte[1024];

byte[] receivebyte = new byte[1024];

// Reader to get input from user (physical address)

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

System.out.println("Enter the Physical address (MAC):");

String str = in.readLine(); // Read MAC address input

sendbyte = str.getBytes(); // Convert MAC address to byte array

// FIXED: missing space in "new DatagramPacket"

DatagramPacket sender = new DatagramPacket(sendbyte, sendbyte.length, addr, 1309);

client.send(sender); // Send packet to server

// Prepare to receive response from server

DatagramPacket receiver = new DatagramPacket(receivebyte, receivebyte.length);

client.receive(receiver); // Receive response

// Convert byte array to string and trim whitespace

String s = new String(receiver.getData());

System.out.println("The Logical Address is (IP): " + s.trim());

client.close(); // Close the socket

} catch (Exception e) {

System.out.println(e); // Print any exception

}

}

}

import java.io.\*; // For input/output streams (not directly used here)

import java.net.\*; // For DatagramSocket, DatagramPacket, InetAddress

import java.util.\*; // For utility classes (optional here)

class Serverrarp12 {

public static void main(String args[]) {

try {

// Create a UDP socket bound to port 1309

DatagramSocket server = new DatagramSocket(1309);

while (true) {

// Buffers for receiving and sending data

byte[] sendbyte = new byte[1024];

byte[] receivebyte = new byte[1024];

// Packet to receive incoming data

DatagramPacket receiver = new DatagramPacket(receivebyte, receivebyte.length);

server.receive(receiver); // Receive the client's request

// Convert received byte array to string and trim it

String str = new String(receiver.getData());

String s = str.trim(); // This is the MAC address from the client

// Get client's address and port

InetAddress addr = receiver.getAddress();

int port = receiver.getPort();

// Lookup table: IP addresses and corresponding MAC addresses

String ip[] = {"165.165.80.80", "165.165.79.1"};

String mac[] = {"6A:08:AA:C2", "8A:BC:E3:FA"};

// Check if received MAC matches any in the list

for (int i = 0; i < ip.length; i++) {

if (s.equals(mac[i])) {

sendbyte = ip[i].getBytes(); // Convert matching IP to bytes

// FIXED: missing space in "new DatagramPacket"

DatagramPacket sender = new DatagramPacket(sendbyte, sendbyte.length, addr, port);

server.send(sender); // Send IP address to client

break;

}

}

break; // Exit after first query (remove this if server should run continuously)

}

} catch (Exception e) {

System.out.println(e); // Print exception if any

}

}

}

**6**

import java.io.\*; // Import necessary classes for input

class CRC {

public static void main(String args[]) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); // Reader for user input

System.out.println("Enter Generator:");

String gen = br.readLine(); // Read the generator polynomial

System.out.println("Enter Data:");

String data = br.readLine(); // Read the input data

String code = data;

while(code.length() < (data.length() + gen.length() - 1)) // Append zeros to the data

code = code + "0";

code = data + div(code,gen); // Final codeword = data + CRC remainder

System.out.println("The transmitted Code Word is: " + code);

System.out.println("Please enter the received Code Word: ");

String rec = br.readLine(); // Read the received codeword

if(Integer.parseInt(div(rec,gen)) == 0) // If division result is 0, no error

System.out.println("The received code word contains no errors.");

else

System.out.println("The received code word contains errors.");

}

// Function to simulate binary division (mod-2) between dividend and divisor

static String div(String num1, String num2) {

int pointer = num2.length(); // Set pointer to length of generator

String result = num1.substring(0, pointer); // Take initial part of dividend

String remainder = "";

for(int i = 0; i < num2.length(); i++) { // XOR the result with generator

if(result.charAt(i) == num2.charAt(i))

remainder += "0";

else

remainder += "1";

}

while(pointer < num1.length()) { // Continue until end of dividend

if(remainder.charAt(0) == '0') { // If MSB is 0, discard it

remainder = remainder.substring(1, remainder.length());

remainder = remainder + String.valueOf(num1.charAt(pointer)); // Append next bit

pointer++;

}

result = remainder; // New result is current remainder

remainder = "";

for(int i = 0; i < num2.length(); i++) { // XOR again

if(result.charAt(i) == num2.charAt(i))

remainder += "0";

else

remainder += "1";

}

}

return remainder.substring(1, remainder.length()); // Return final remainder (CRC)

}

}