**Lab No. 1 Introducing to Network Programming**

**A. Familiarity with lab environment and Client-Server model, Unix basic commands.**

**Objective:**

To gain familiarity with the lab environment, the Client-Server model, and Unix basic commands, and to understand and use common networking utility commands.

**1. Lab Environment:**

Ensure you have access to a Unix/Linux machine or a terminal emulator (e.g., PuTTY for Windows).

Familiarize yourself with the terminal window where you will run commands.

Confirm that network services such as SSH and FTP are accessible.

**2. Client-Server Model:**

Client-Server Architecture: In this model, the client sends requests, and the server processes those requests and sends back responses. Examples of services include web browsing (HTTP), file transfers (FTP), and remote access (SSH, Telnet).

Client Examples: Browsers, FTP clients, SSH clients.

Server Examples: Web servers (Apache, Nginx), FTP servers, SSH servers.

**B. Understanding and using the following Networking Utility commands. Ifconfig, netstat, ping, arp, telnet, ftp, finter, whois etc.**

**1. Ifconfig**

Displays network interface configuration, including IP addresses, MAC addresses, and subnet masks.

**Lab2: InetAddress Class**

**A. Create and use InetAddress objects and display information of InetAddress.**

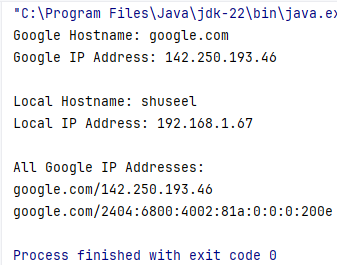
**Objectives:**

The InetAddress class in Java serves several key objectives, primarily related to working with IP addresses and hostnames. It represents IP Addresses, manage local and remote host.

**Source Code:**

import java.net.InetAddress;  
import java.net.UnknownHostException;  
  
public class InetAddressE {  
  
 public static void main(String[] args) {  
 try {  
 InetAddress google = InetAddress.*getByName*("google.com");  
 System.*out*.println("Google Hostname: " + google.getHostName());  
 System.*out*.println("Google IP Address: " + google.getHostAddress());  
 System.*out*.println();  
  
 InetAddress localHost = InetAddress.*getLocalHost*();  
 System.*out*.println("Local Hostname: " + localHost.getHostName());  
 System.*out*.println("Local IP Address: " + localHost.getHostAddress());  
 System.*out*.println();  
  
 InetAddress[] googleAddresses = InetAddress.*getAllByName*("google.com");  
 System.*out*.println("All Google IP Addresses:");  
 for (InetAddress address : googleAddresses) {  
 System.*out*.println(address);  
 }  
  
 } catch (UnknownHostException e) {  
 e.printStackTrace();  
 }  
 }  
}

Output:



**B. Create NetworkInterface and display the properties of a Network Interface.**

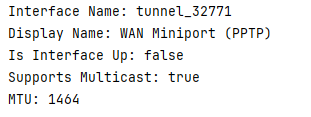
Objectives:

The NetworkInterface class in Java provides methods to represent, access, and retrieve detailed information about network interfaces, including their name, MAC address, IP addresses, and operational status, for network communication.

#Source Code:

import java.net.\*;  
import java.util.Enumeration;  
  
public class NetworkInterfaceE {  
 public static void main(String[] args) {  
 try {  
 Enumeration<NetworkInterface> interfaces = NetworkInterface.*getNetworkInterfaces*();  
  
 while (interfaces.hasMoreElements()) {  
 NetworkInterface networkInterface = interfaces.nextElement();  
  
 System.*out*.println("Interface Name: " + networkInterface.getName());  
 System.*out*.println("Display Name: " + networkInterface.getDisplayName());  
  
 byte[] mac = networkInterface.getHardwareAddress();  
 if (mac != null) {  
 System.*out*.print("MAC Address: ");  
 for (int i = 0; i < mac.length; i++) {  
 System.*out*.format("%02X%s", mac[i], (i < mac.length - 1) ? "-" : "");  
 }  
 System.*out*.println();  
 }  
  
 System.*out*.println("Is Interface Up: " + networkInterface.isUp());  
  
 System.*out*.println("Supports Multicast: " + networkInterface.supportsMulticast());  
  
 System.*out*.println("MTU: " + networkInterface.getMTU());  
  
 Enumeration<InetAddress> inetAddresses = networkInterface.getInetAddresses();  
 while (inetAddresses.hasMoreElements()) {  
 InetAddress inetAddress = inetAddresses.nextElement();  
 System.*out*.println("IP Address: " + inetAddress.getHostAddress());  
 }  
  
 }  
 } catch (SocketException e) {  
 e.printStackTrace();  
 }  
 }  
}

Output:



**Lab 3 URLs and URIs**

**A. Write a Java Program that users all eight methods of URL class to split URLs entered on the command-line into their component parts.**

**#Source Code:**

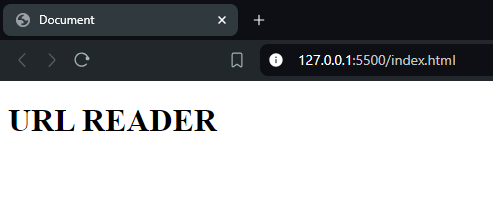
import java.net.URL;  
import java.net.MalformedURLException;  
  
  
public class UrlSplitE {  
 public static void main(String[] args) {  
 if (args.length != 1) {  
 System.*out*.println("Usage: java URLSplitter <URL>");  
 return;  
 }  
  
 try {  
 URL url = new URL(args[0]);  
  
 System.*out*.println("Full URL: " + url.toString());  
 System.*out*.println("Protocol: " + url.getProtocol());  
 System.*out*.println("Host: " + url.getHost());  
 System.*out*.println("Port: " + (url.getPort() == -1 ? "Default Port" : url.getPort()));  
 System.*out*.println("Path: " + url.getPath());  
 System.*out*.println("File: " + url.getFile());  
 System.*out*.println("Query: " + (url.getQuery() == null ? "No Query" : url.getQuery()));  
 System.*out*.println("Reference (Fragment): " + (url.getRef() == null ? "No Fragment" : url.getRef()));  
  
 } catch (MalformedURLException e) {  
 System.*out*.println("The URL provided is malformed: " + args[0]);  
 }  
 }  
  
}

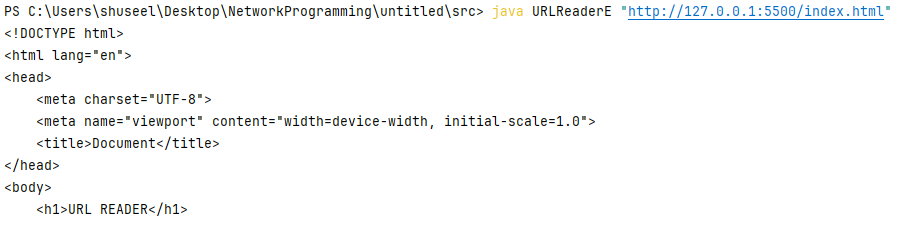
**b. Write a program that reads a URL from the command line, then prints the raw data located at the URL.**

**Source Code:**

import java.io.\*;  
import java.net.\*;  
  
  
public class URLReaderE {  
 public static void main(String[] args) {  
 if (args.length != 1) {  
 System.*out*.println("Usage: java URLReaderE <URL>");  
 return;  
 }  
  
 BufferedReader reader = null;  
  
 try {  
 String inputUrl = args[0];  
 if (!inputUrl.startsWith("http://") && !inputUrl.startsWith("https://")) {  
 inputUrl = "http://" + inputUrl;  
 }  
  
 URL url = new URL(inputUrl);  
  
 URLConnection connection = url.openConnection();  
 reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));  
  
 String line;  
 while ((line = reader.readLine()) != null) {  
 System.*out*.println(line);  
 }  
  
 } catch (MalformedURLException e) {  
 System.*out*.println("The URL provided is malformed: " + args[0]);  
 } catch (IOException e) {  
 System.*out*.println("Error reading from the URL: " + e.getMessage());  
 } finally {  
 if (reader != null) {  
 try {  
 reader.close();  
 } catch (IOException e) {  
 System.*out*.println("Error closing the reader: " + e.getMessage());  
 }  
 }  
 }  
 }  
}

**Output:**





**Lab 4: URL Connection**

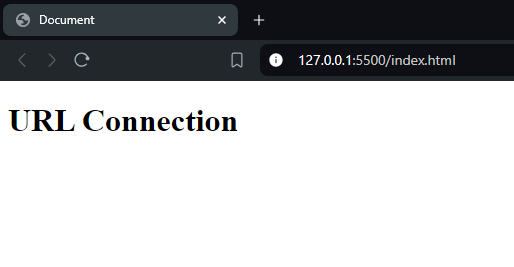
**A. Write a java program to create a URLConnection using openConnection() method of URL object and then use it to examine the document's properties and content.**

**Source Code:**

import java.net.\*;  
import java.io.\*;  
  
  
public class URLConnectionE {  
  
 public static void main(String[] args) {  
 if (args.length != 1) {  
 System.*out*.println("Usage: java URLConnectionExample <URL>");  
 return;  
 }  
  
 BufferedReader reader = null;  
  
 try {  
 URL url = new URL(args[0]);  
  
 URLConnection connection = url.openConnection();  
  
 connection.connect();  
  
 System.*out*.println("Document Properties:");  
 System.*out*.println("Content Type: " + connection.getContentType());  
 System.*out*.println("Content Length: " + connection.getContentLength());  
 System.*out*.println("Content Encoding: " + connection.getContentEncoding());  
 System.*out*.println("Date: " + connection.getDate());  
 System.*out*.println("Last Modified: " + connection.getLastModified());  
 System.*out*.println("Expiration: " + connection.getExpiration());  
  
 System.*out*.println("\nDocument Content:");  
  
 reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));  
  
 String line;  
 while ((line = reader.readLine()) != null) {  
 System.*out*.println(line);  
 }  
  
 } catch (MalformedURLException e) {  
 System.*out*.println("The URL provided is malformed: " + args[0]);  
 } catch (IOException e) {  
 System.*out*.println("Error reading from the URL: " + e.getMessage());  
 } finally {  
 if (reader != null) {  
 try {  
 reader.close();  
 } catch (IOException e) {  
 System.*out*.println("Error closing the reader: " + e.getMessage());  
 }  
 }  
 }  
 }  
}

**Output:**

****

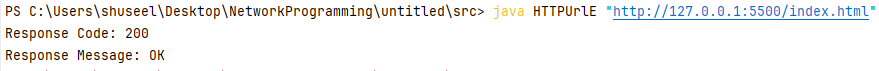
****

**b. Write a java program to create an URLConnection and display response code and message.**

**Source Code**

import java.net.\*;  
import java.io.\*;  
public class HTTPUrlE {  
  
 public static void main(String[] args) {  
 if (args.length != 1) {  
 System.*out*.println("Usage: java HttpURLConnectionExample <URL>");  
 return;  
 }  
  
 try {  
 URL url = new URL(args[0]);  
  
 HttpURLConnection connection = (HttpURLConnection) url.openConnection();  
  
 connection.setRequestMethod("GET");  
  
 connection.connect();  
  
 int responseCode = connection.getResponseCode();  
 String responseMessage = connection.getResponseMessage();  
  
 System.*out*.println("Response Code: " + responseCode);  
 System.*out*.println("Response Message: " + responseMessage);  
  
 connection.disconnect();  
  
 } catch (MalformedURLException e) {  
 System.*out*.println("The URL provided is malformed: " + args[0]);  
 } catch (IOException e) {  
 System.*out*.println("Error during connection: " + e.getMessage());  
 }  
 }  
 }

Output:

**Lab 5 Socket Programming**

**a. Create a simple client and server sockets that implements Daytime service in Java.**

**Source Code**

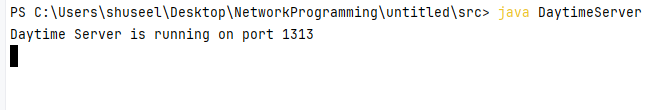
**#DaytimeServer.java**

import java.io.\*;  
import java.net.\*;  
import java.util.Date;  
  
public class DaytimeServer {  
 public static void main(String[] args) {  
 int port = 1313;  
  
 try (ServerSocket serverSocket = new ServerSocket(port)) {  
 System.*out*.println("Daytime Server is running on port " + port);  
  
 while (true) {  
 Socket clientSocket = serverSocket.accept();  
  
 String currentDateTime = new Date().toString();  
  
 PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);  
 out.println(currentDateTime);  
  
 clientSocket.close();  
 }  
  
 } catch (IOException e) {  
 System.*out*.println("Error on the server: " + e.getMessage());  
 e.printStackTrace();  
 }  
 }  
}

**#DaytimeClient.java**

import java.io.\*;  
import java.net.\*;  
  
public class DaytimeClient {  
 public static void main(String[] args) {  
 if (args.length != 1) {  
 System.*out*.println("Usage: java DaytimeClient <server\_ip>");  
 return;  
 }  
  
 String serverAddress = args[0];  
 int port = 1313;  
  
 try (Socket socket = new Socket(serverAddress, port)) {  
 BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
  
 String serverResponse = in.readLine();  
 System.*out*.println("Current date and time: " + serverResponse);  
  
 } catch (IOException e) {  
 System.*out*.println("Error on the client: " + e.getMessage());  
 e.printStackTrace();  
 }  
 }  
}

**Output:**

****

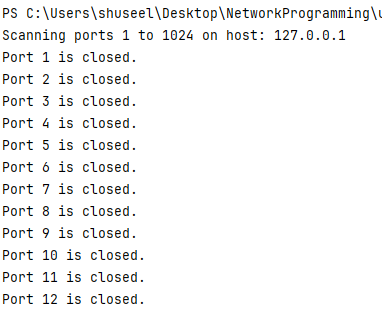
**b. Create a java based simple Low Port Scanner program**

Source Code:

*#LowPortScanner.java*

import java.io.IOException;  
import java.net.Socket;  
  
public class LowPortScanner {  
 public static void main(String[] args) {  
 if (args.length != 1) {  
 System.*out*.println("Usage: java LowPortScanner <host>");  
 return;  
 }  
  
 String host = args[0];  
 int startPort = 1;  
 int endPort = 1024;  
  
 System.*out*.println("Scanning ports " + startPort + " to " + endPort + " on host: " + host);  
  
 for (int port = startPort; port <= endPort; port++) {  
 try {  
 Socket socket = new Socket(host, port);  
  
 System.*out*.println("Port " + port + " is open.");  
  
 socket.close();  
 } catch (IOException e) {  
 System.*out*.println("Port " + port + " is closed.");  
 }  
 }  
  
 System.*out*.println("Port scan completed.");  
 }  
}

Output:



Lab 6: Advance Socket Programming

a. Write a simple program to implement whois as a simple directory service protocol using socket.

Source Code

WhoClient.java

import java.io.\*;  
import java.net.\*;  
  
public class WhoClient {  
 public static void main(String[] args) {  
 if (args.length != 1) {  
 System.*out*.println("Usage: java WhoisClient <domain>");  
 return;  
 }  
  
 String domain = args[0];  
 String whoisServer = "whois.iana.org";  
  
 try {  
 Socket socket = new Socket(whoisServer, 43);  
  
 OutputStream out = socket.getOutputStream();  
 out.write((domain + "\r\n").getBytes());  
 out.flush();  
  
 InputStream in = socket.getInputStream();  
 BufferedReader reader = new BufferedReader(new InputStreamReader(in));  
  
 String line;  
 System.*out*.println("Whois information for: " + domain);  
 while ((line = reader.readLine()) != null) {  
 System.*out*.println(line);  
 }  
  
 socket.close();  
 } catch (IOException e) {  
 System.*err*.println("Error connecting to the Whois server: " + e.getMessage());  
 e.printStackTrace();  
 }  
 }  
}

**Output:\**

****

****

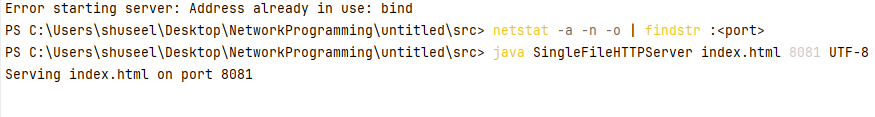
**b. Write a java program to create a SingleFileHTTPServer program using ServerSocket that is supplied filename, local port and content encoding from command line.**

**Source Code**

**SingleFileHTTPServer.java**

import java.io.\*;  
import java.net.\*;  
import java.nio.file.\*;  
import java.util.Date;  
  
public class SingleFileHTTPServer {  
  
 private File file;  
 private int port;  
 private String encoding;  
  
 public SingleFileHTTPServer(File file, int port, String encoding) {  
 this.file = file;  
 this.port = port;  
 this.encoding = encoding;  
 }  
  
 public void start() throws IOException {  
 ServerSocket serverSocket = new ServerSocket(port);  
 System.*out*.println("Serving " + file.getName() + " on port " + port);  
  
 while (true) {  
 try (Socket clientSocket = serverSocket.accept()) {  
 serveFile(clientSocket);  
 } catch (IOException e) {  
 System.*err*.println("Connection error: " + e.getMessage());  
 }  
 }  
 }  
  
 private void serveFile(Socket clientSocket) throws IOException {  
 OutputStream rawOutput = clientSocket.getOutputStream();  
 Writer out = new OutputStreamWriter(rawOutput, encoding);  
 BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));  
  
 String requestLine = in.readLine();  
 if (requestLine != null && requestLine.startsWith("GET")) {  
 sendHeader(out, "HTTP/1.0 200 OK", "text/html; charset=" + encoding, file.length());  
  
 Files.*copy*(file.toPath(), rawOutput);  
 rawOutput.flush();  
 } else {  
 sendHeader(out, "HTTP/1.0 400 Bad Request", "text/plain", 0);  
 out.write("Bad Request");  
 out.flush();  
 }  
  
 out.close();  
 in.close();  
 }  
  
 private void sendHeader(Writer out, String responseCode, String contentType, long contentLength) throws IOException {  
 out.write(responseCode + "\r\n");  
 out.write("Date: " + new Date() + "\r\n");  
 out.write("Server: SingleFileHTTPServer 1.0\r\n");  
 out.write("Content-length: " + contentLength + "\r\n");  
 out.write("Content-type: " + contentType + "\r\n\r\n");  
 out.flush();  
 }  
  
 public static void main(String[] args) {  
 if (args.length < 3) {  
 System.*out*.println("Usage: java SingleFileHTTPServer <file> <port> <encoding>");  
 return;  
 }  
  
 File file = new File(args[0]);  
 int port = Integer.*parseInt*(args[1]);  
 String encoding = args[2];  
  
 if (!file.exists()) {  
 System.*err*.println("File not found: " + file.getAbsolutePath());  
 return;  
 }  
  
 try {  
 SingleFileHTTPServer server = new SingleFileHTTPServer(file, port, encoding);  
 server.start();  
 } catch (IOException e) {  
 System.*err*.println("Error starting server: " + e.getMessage());  
 }  
 }  
}

**Output:**

****

**Lab 7 Building a UDP Client/Server.**

**Create and run java based echo client and echo server application and display its working.**

**Source Code**

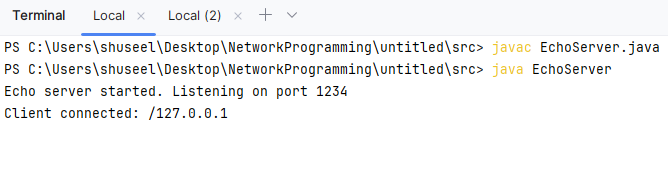
**EchoServer.java**

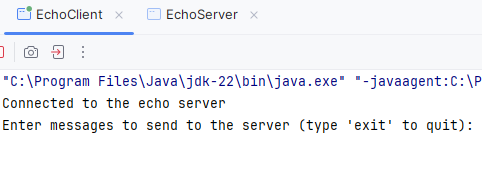
import java.io.\*;  
import java.net.\*;  
  
public class EchoServer {  
 public static void main(String[] args) {  
 int port = 1234;  
  
 try (ServerSocket serverSocket = new ServerSocket(port)) {  
 System.*out*.println("Echo server started. Listening on port " + port);  
  
 while (true) {  
 try (Socket clientSocket = serverSocket.accept();  
 BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));  
 PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true)) {  
  
 System.*out*.println("Client connected: " + clientSocket.getInetAddress());  
  
 String receivedMessage;  
 while ((receivedMessage = in.readLine()) != null) {  
 System.*out*.println("Received: " + receivedMessage);  
 out.println(receivedMessage);  
 }  
 } catch (IOException e) {  
 System.*err*.println("Connection error: " + e.getMessage());  
 }  
 }  
 } catch (IOException e) {  
 System.*err*.println("Error starting server: " + e.getMessage());  
 }  
 }  
}

EchoClient.java

import java.io.\*;  
import java.net.\*;  
  
public class EchoClient {  
 public static void main(String[] args) {  
 String hostname = "localhost";  
 int port = 1234;  
  
 try (Socket socket = new Socket(hostname, port);  
 BufferedReader userInput = new BufferedReader(new InputStreamReader(System.*in*));  
 BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 PrintWriter out = new PrintWriter(socket.getOutputStream(), true)) {  
  
 System.*out*.println("Connected to the echo server");  
  
 String messageToSend;  
 System.*out*.println("Enter messages to send to the server (type 'exit' to quit):");  
  
 while ((messageToSend = userInput.readLine()) != null) {  
 out.println(messageToSend);  
  
 if ("exit".equalsIgnoreCase(messageToSend)) {  
 System.*out*.println("Closing connection...");  
 break;  
 }  
  
 String serverResponse = in.readLine();  
 System.*out*.println("Echoed from server: " + serverResponse);  
 }  
 } catch (UnknownHostException e) {  
 System.*err*.println("Unknown host: " + e.getMessage());  
 } catch (IOException e) {  
 System.*err*.println("I/O error: " + e.getMessage());  
 }  
 }  
}

Output:





Source Code:

RemoteInterface.java

import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface RemoteInterface extends Remote {  
 String sayHello() throws RemoteException;  
}

Remote Server.java

import java.rmi.RemoteException;  
import java.rmi.server.UnicastRemoteObject;  
  
public class RemoteServer extends UnicastRemoteObject implements RemoteInterface {  
  
 protected RemoteServer() throws RemoteException {  
 super();  
 }  
  
 @Override  
 public String sayHello() throws RemoteException {  
 return "Hello from RMI Server!";  
 }  
}

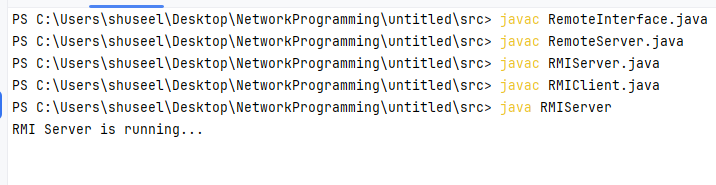
#RMIServer.java

import java.rmi.Naming;  
import java.rmi.RemoteException;  
import java.rmi.registry.LocateRegistry;  
  
public class RMIServer {  
 public static void main(String[] args) {  
 try {  
 LocateRegistry.*createRegistry*(1099);  
  
 RemoteServer server = new RemoteServer();  
  
 Naming.*rebind*("rmi://localhost/HelloService", server);  
  
 System.*out*.println("RMI Server is running...");  
 } catch (RemoteException | java.net.MalformedURLException e) {  
 e.printStackTrace();  
 }  
 }  
}

#RMIClient.java

import java.rmi.Naming;  
  
public class RMIClient {  
 public static void main(String[] args) {  
 try {  
 RemoteInterface remoteObject = (RemoteInterface) Naming.*lookup*("rmi://localhost/HelloService");  
  
 String response = remoteObject.sayHello();  
 System.*out*.println("Response from server: " + response);  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

**Output:**

****

****