...for a complete human being

Texas International college

Date: 05-10-2024

Lab 1: Write a program to demonstrate basic network communication between a client and server.

Objective: To demonstrate basic network communication between a client and server using Socket and ServerSocket in Java.

Steps:

1. Set up a ServerSocket in order to listen for client connections.
2. Connect to server creating a socket on the client side.
3. Populate a message from the client to the server, which the server receives and displays.
4. Close the connections after communication is over.

Code Snippet

Server

```
import java.io.*;
import java.net.*;

public class Lab1Server {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(9001);
            System.out.println("Server started to listen on port 9001");
            Socket clientSocket = ss.accept();
```

```
            System.out.println("Connection is established with the client");
            DataInputStream dis = new DataInputStream(clientSocket.getInputStream());
            String str = (String) dis.readUTF();
            System.out.println("The message is: " + str);
            ss.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Client

```
import java.io.*;
import java.net.*;
public class Lab1Client {
    public static void main(String[] args) {
        try {
            Socket s = new Socket("localhost", 9001);
            System.out.println("Successfully connected to the server");
            DataOutputStream dout = new DataOutputStream(s.getOutputStream());
            dout.writeUTF("Hello Server!");
            dout.flush();
            dout.close();
            s.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
Run    Lab1Server
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:
Server started to listen on port 9001
```

```
Run    Lab1Server    Lab1Client
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe"
Successfully connected to the server

Process finished with exit code 0
```

Date: 05-10-2024

2. Write a program to display the IP address of a given hostname and test its reachability.

Objective: To display the IP address of a given hostname and to test its reachability using the InetAddress class in Java.

Steps:

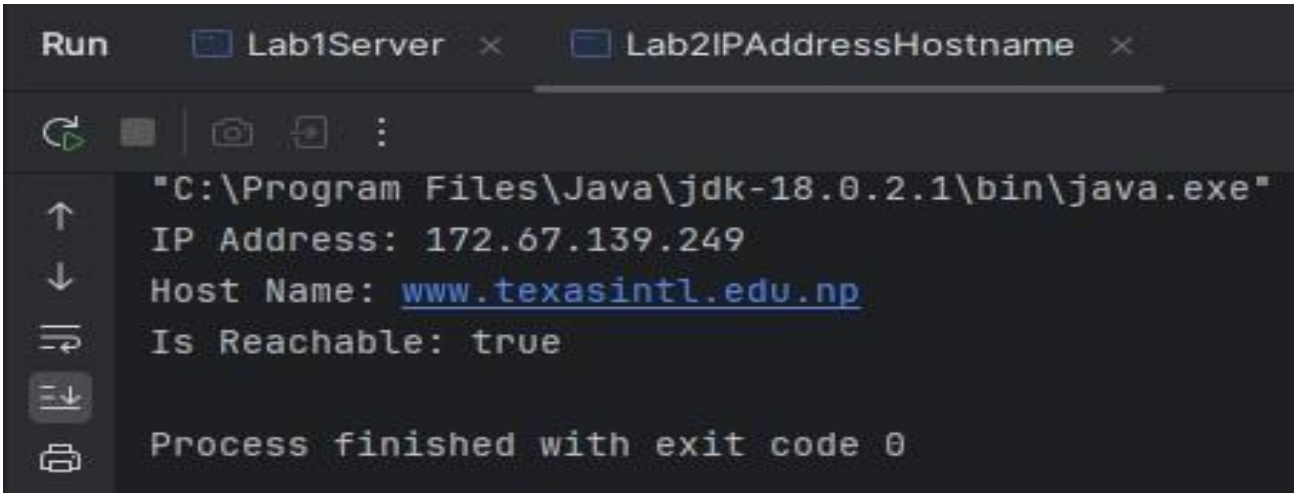
1. Retrieve and display the IP address of a given hostname using InetAddress.getByName().
2. Receive and display various details such as host name, canonical host name, and address.
3. Create and display InetAddress objects using different methods like getByAddress() and getLocalHost().

Code Snippet

```
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.io.IOException;

public class Lab2IPAddressHostname {
    public static void main(String[] args) {
        try {
            // Get the InetAddress object for the given hostname
            InetAddress inetAddress = InetAddress.getByName("www.texasintl.edu.np");
```

```
            // Display the IP address and hostname
            System.out.println("IP Address: " + inetAddress.getHostAddress());
            System.out.println("Host Name: " + inetAddress.getHostName());

            // Test reachability with a timeout of 3000 milliseconds (3 seconds)
            boolean reachable = inetAddress.isReachable(3000);
            System.out.println("Is Reachable: " + reachable);

        } catch (UnknownHostException e) {
            System.err.println("Unknown host: " + e.getMessage());
        } catch (IOException e) {
            System.err.println("Network error: " + e.getMessage());
        }
    }
}
```

Output:



Date: 05-10-2024

3. Write a program to list all network interfaces and their associated addresses.

Objective: To list all the network interfaces on the system and their associated IP addresses using the NetworkInterface and InetAddress classes in Java.

Steps:

1. Get all network interfaces using NetworkInterface.getNetworkInterfaces().
2. look over each interface and print name, display name, and associated IP addresses.
3. For error management, handle SocketException and UnknownHostException.

Code Snippet

```
import java.util.*;
import java.net.*;

public class Lab3ListNetworkInterfaces {
    public static void main(String[] args) {
        try {
            // List all network interfaces
            Enumeration<NetworkInterface> interfaces =
            NetworkInterface.getNetworkInterfaces();
            while (interfaces.hasMoreElements()) {
                NetworkInterface ni = interfaces.nextElement();
                System.out.println("Display Name: " + ni.getDisplayName());
                System.out.println("Name: " + ni.getName());
```

```
                // List all IP addresses associated with the network interface
                Enumeration<InetAddress> inetAddresses = ni.getInetAddresses();
                while (inetAddresses.hasMoreElements()) {
                    InetAddress address = inetAddresses.nextElement();
                    System.out.println("Inet Address: " + address);
                }
                System.out.println("-------------------------------------");
            }
        } catch (SocketException ex) {
            System.err.println("Could not list network interfaces.");
            ex.printStackTrace();
        }
    }
}
```

Output:

```
Run    Lab3NetworkInterfaces
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C:\Program Files
Display Name: Software Loopback Interface 1
Name: lo
Inet Address: /127.0.0.1
Inet Address: /0:0:0:0:0:0:0:1
-------------------------------------
Display Name: Microsoft 6to4 Adapter
Name: net0
-------------------------------------
Display Name: Bluetooth Device (Personal Area Network)
Name: eth0
Inet Address: /fe80:0:0:0:2307:40d3:d03b:8ecb%eth0
-------------------------------------
Display Name: WAN Miniport (L2TP)
Name: net1
-------------------------------------
Display Name: WAN Miniport (IP)
Name: eth1
```

Date: 05-10-2024

4. Write a program to display different static methods of INetAddress.

Objective: To display the various static methods provided by the InetAddress class in Java.

Steps:

1. Retrieve Local Host Address: Use InetAddress.getLocalHost() to get the address of the local machine.
2. Get InetAddress by Hostname: Use InetAddress.getByName() to get the IP address of a given hostname.
3. GRetrieve Loopback Address: Use InetAddress.getLoopbackAddress() to get the loopback IP address.
4. Get All IPs for a Hostname: Use InetAddress.getAllByName() to get all IP addresses associated with a given hostname.
5. Get InetAddress by IP Address: Use InetAddress.getByAddress() to create an InetAddress object from a byte array representing an IP address.

Code Snippet

```
import java.net.InetAddress;
import java.net.UnknownHostException;

public class Lab4INetAddressStaticMethods {
   public static void main(String[] args) {
      try {
```

```
            // Get the local host address
            InetAddress localHost = InetAddress.getLocalHost();
            System.out.println("Local Host: " + localHost);

            // Get the loopback address (127.0.0.1)
            InetAddress loopback = InetAddress.getLoopbackAddress();
            System.out.println("Loopback Address: " + loopback);

            // Get an InetAddress object by hostname
            InetAddress byName = InetAddress.getByName("www.google.com");
            System.out.println("Google IP: " + byName);

            // Get all IP addresses associated with a hostname
            InetAddress[] allByName = InetAddress.getAllByName("www.google.com");
            System.out.println("All IP addresses for Google:");
            for (InetAddress address : allByName) {
                System.out.println("  " + address);
            }
            // Get InetAddress from an IP address (byte array)
            byte[] ip = {8, 8, 8, 8}; // IP address for Google's public DNS server
            InetAddress byAddress = InetAddress.getByAddress(ip);
            System.out.println("Google DNS by IP: " + byAddress);

        } catch (UnknownHostException e) {
            System.err.println("Unknown Host: " + e.getMessage());
        }
    }
}
```

Output:

```
Run    Lab4INetAddressStaticMethods

"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe"
Local Host: DESKTOP-8H78II0/192.168.254.180
Loopback Address: localhost/127.0.0.1
Google IP: www.google.com/142.250.77.228
All IP addresses for Google:
  www.google.com/142.250.77.228
  www.google.com/2404:6800:4002:814:0:0:0:2004
Google DNS by IP: /8.8.8.8

Process finished with exit code 0
```

Date: 05-10-2024

5. Write a program to create URLs, parse their components, and retrieve data from them.

Objective: To create URLs, parse their components, and to retrieve the data from them.

Steps:

1. Construct the URLs using strings, component parts, and relative paths.
2. Parse the components of the URL such as host, protocol, port, path, query, and fragment.
3. Retrieve and display the content from the URL using an InputStream.
4. Handle the exceptions related to malformed URLs and I/O operations.

Code Snippet

```
import java.io.*;
import java.net.*;

public class Lab5CreateParseRetrieveURL {
    public static void main(String[] args) {
        try {
            // Constructing a URL from a string
            URL url1 = new URL("https://www.google.com");
            System.out.println("Constructed URL: " + url1);

            // Constructing a URL from its component parts
            URL url2 = new URL("https", "www.google.com", "/search");
```

```
        System.out.println("Constructed URL from components: " + url2);

        // Constructing a relative URL
        URL baseURL = new URL("https://www.google.com/");
        URL relativeURL = new URL(baseURL, "intl/en/about/");
        System.out.println("Constructed relative URL: " + relativeURL);

        // Parsing URL components
        System.out.println("\nParsing URL Components:");
        System.out.println("Protocol: " + url1.getProtocol());
        System.out.println("Host: " + url1.getHost());
        System.out.println("Port: " + url1.getPort()); // -1 indicates no port specified
        System.out.println("Path: " + url1.getPath());
        System.out.println("Query: " + url1.getQuery());
        System.out.println("Fragment: " + url1.getRef());

        // Retrieving and displaying data from the URL
        System.out.println("\nRetrieving Data from URL:");
        try (InputStream in = new BufferedInputStream(url1.openStream());
             Reader reader = new InputStreamReader(in)) {
            int c;
            while ((c = reader.read()) != -1) {
                System.out.print((char) c);
            }
        }

    } catch (MalformedURLException ex) {
        System.err.println("Error: Invalid URL format.");
    } catch (IOException ex) {
        System.err.println("Error: " + ex.getMessage());
    }
```

```
        }
    }
```

Output:

```
Run    Lab5CreateParseRetrieveURL

Parsing URL Components:
Protocol: https
Host: www.google.com
Port: -1
Path:
Query: null
Fragment: null

Retrieving Data from URL:
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="ne"><head><meta content="te
var h=this||self;function l(){return window.google!==void 0&&window.google.kOPI!==void 0&&window.google.
function t(a,b,c,d,k){var e="";b.search("&ei=")===-1&&(e="&ei="+p(d),b.search("&lei=")===-1&&(d=q(d))&&(
document.documentElement.addEventListener("submit",function(b){var a;if(a=b.target){var c=a.getAttribute
</style><style>body,td,a,p,.h{font-family:arial,sans-serif}body{margin:0;overflow-y:scroll}#gog{padding:
var g=this||self;var k,l=(k=g.mei)!=null?k:1,n,p=(n=g.sdo)!=null?n:!0,q=0,r,t=google.erd,v=t.jsr;google.
```

Date: 05-10-2024

6. Develop a program that reads and displays the content of a web page given its URL.

Objective: To develop a program that reads and displays the content of a web page given its URL.

Steps:

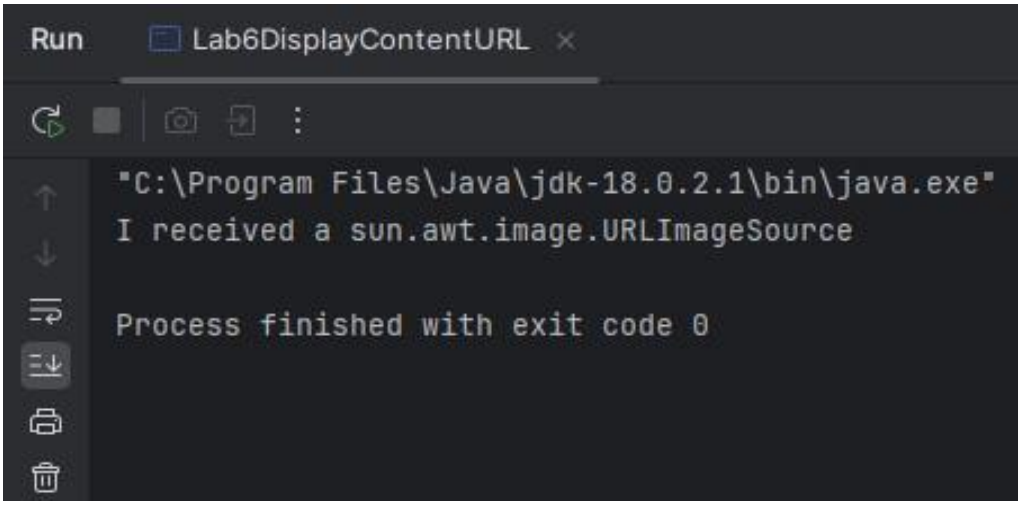
1. Construct URLs using strings, component parts, and relative paths.
2. Parse the components of the URL such as host, protocol, port, path, query, and fragment.
3. Retrieve and display the content from the URL using an InputStream.
4. Handle exceptions related to malformed URLs and I/O operations.

Code Snippet

```
import java.io.*;
import java.net.*;

public class Lab6DisplayContentURL {
    // https://logos-world.net/wp-content/uploads/2022/07/Java-Logo.png
    public static void main(String[] args) {
        // Open the URL for reading
        try {
            URL u = new URL("https://logos-world.net/wp-content/uploads/2022/07/Java-Logo.png");
            Object o = u.getContent();
```

```
            System.out.println("I got a " + o.getClass().getName());
        } catch (MalformedURLException ex) {
            System.err.println( "Provided URL is not a parseable URL");
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Output:



Date: 05-10-2024

7. Write a program to open a URL connection and read data from it.

Objective: To create a program that opens a URL connection and reads data from it.

Steps:

1. Create a URL object with the target URL.
2. Open a connection with the URL using HttpURLConnection.
3. Read data from the connection using a BufferedReader.
4. Display the retrieved data by in console.

Code Snippet

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

public class Lab7ReadDataURL {
    public static void main(String[] args) {
        try {
            // Create a URL object
            URL url = new URL("https://jsonplaceholder.typicode.com/posts/1");

            // Open a connection to the URL
            HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
```

```
            // Set the request method to GET
            urlConnection.setRequestMethod("GET");

            // Read the response
            BufferedReader in = new BufferedReader(new
                    InputStreamReader(urlConnection.getInputStream()));
            String inputLine;
            StringBuilder content = new StringBuilder();
            while ((inputLine = in.readLine()) != null) {
                content.append(inputLine).append("\n");
            }

            // Close connections
            in.close();
            urlConnection.disconnect();

            // Print the retrieved content
            System.out.println(content.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
Run    Lab7ReadDataURL ×

"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community
{
  "userId": 1,
  "id": 1,
  "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas
}


Process finished with exit code 0
```

...for a complete human being
Texas
International college

Date: 05-10-2024

8. Develop a program to read and display HTTP header fields from a URL connection.

Objective: To develop a program that reads and displays HTTP header fields from a URL connection.

Steps:

1. First of all ,Create a URL object from the provided URL string.
2. Open a connection to the URL using the URLConnection class.
3. Retrieve and display the HTTP header fields by iterating through them using getHeaderFieldKey and getHeaderField.
4. Handle exceptions for malformed URLs and IO errors.

Code Snippet

```
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;

public class Lab8DisplayHTTPHeaderFields {

    public static void main(String[] args) {
        String urlString = "https://www.w3schools.com/java/";
        try {
```

```
            // Create a URL object
            URL url = new URL(urlString);

            // Open a connection to the URL
            URLConnection connection = url.openConnection();

            // Iterate through the HTTP headers and print each header field
            for (int i = 1; ; i++) {
                String headerKey = connection.getHeaderFieldKey(i);
                String headerValue = connection.getHeaderField(i);

                // Break if there are no more headers
                if (headerKey == null && headerValue == null) break;

                System.out.println(headerKey + ": " + headerValue);
            }
        } catch (MalformedURLException e) {
            System.err.println(urlString + " is not a valid URL.");
        } catch (IOException e) {
            System.err.println("Error in connection: " + e.getMessage());
        }
    }
}
```

Output:

```
Run    Lab8DisplayHTTPHeaderFields

Content-Security-Policy: frame-ancestors 'self' https://mycourses.w3schools.com https://pathfinder.w3schools.com;
Content-Type: text/html
Date: Tue, 05 Nov 2024 07:25:01 GMT
Expires: Tue, 05 Nov 2024 11:25:01 GMT
Last-Modified: Tue, 05 Nov 2024 03:41:46 GMT
Server: ECS (nag/99AF)
Vary: Accept-Encoding
X-Cache: HIT
X-Content-Security-Policy: frame-ancestors 'self' https://mycourses.w3schools.com https://pathfinder.w3schools.com;
X-Powered-By: ASP.NET
Content-Length: 374098

Process finished with exit code 0
```

Date: 05-10-2024

9. Write a program to create a socket, connect to a server, and send/receive messages.

Objective: To creates a socket, connects to a server, sends a message, and receives a response using java.

Steps:

1. Establish a Connection: The client establishes a socket connection to the server using server's IP address and port number.
2. Send a Message: The client sends a message to the server through the socket's output stream.
3. Receive a Response: The client reads the server's response using the socket's input stream.
4. Close the Socket: Finally, the client closes the socket to end the communication.

Code Snippet

```
import java.io.*;
import java.net.Socket;

public class Lab9ServerSocketMessage {
    public static final String SERVER = "dict.org";
    public static final int PORT = 2628;
    public static final int TIMEOUT = 15000;

    public static void main(String[] args) {
```

```
        Socket socket = null;
        try {
            // Create a socket connection to the server
            socket = new Socket(SERVER, PORT);
            socket.setSoTimeout(TIMEOUT);

            // Set up the output stream to send data to the server
            OutputStream out = socket.getOutputStream();
            Writer writer = new OutputStreamWriter(out, "UTF-8");
            BufferedWriter bufferedWriter = new BufferedWriter(writer);

            // Set up the input stream to receive data from the server
            InputStream in = socket.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(in, "UTF-
8"));
                // Send SHOW DB command to see available databases
                bufferedWriter.write("SHOW DB\r\n");
            bufferedWriter.flush();

            // Print available databases
            System.out.println("Available databases:");
            String response;
            while ((response = reader.readLine()) != null) {
                System.out.println(response);
                if (response.startsWith("250 ")) { // End of the list
                    break;
                }
            }
            // Use a valid database, such as WordNet (wn)
            String query = "DEFINE wn gold\r\n";
            bufferedWriter.write(query);
```

```
            bufferedWriter.flush();

            // Read and print the definition from the server
            System.out.println("\nDefinition of 'gold':");
            while ((response = reader.readLine()) != null) {
                if (response.startsWith("250 ") || response.startsWith("552 ")) {
                    break; // End of the response or no match
                }
                System.out.println(response);
            }

            // Send the "quit" command to terminate the connection
            bufferedWriter.write("quit\r\n");
            bufferedWriter.flush();

        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {
            if (socket != null) {
                try {
                    socket.close();
                } catch (IOException ex) {
                    // Ignore
                }
            }
        }
    }
}
```

Output:

```
Run    Lab9ServerSocketMessage
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community
Available databases:
220 dict.dict.org dictd 1.12.1/rf on Linux 4.19.0-10-amd64 <auth.mime> <384229884.6117.1730791678@dict.dict.org>
110 166 databases present
gcide "The Collaborative International Dictionary of English v.0.48"
wn "WordNet (r) 3.0 (2006)"
moby-thesaurus "Moby Thesaurus II by Grady Ward, 1.0"
elements "The Elements (07Nov00)"
vera "V.E.R.A. -- Virtual Entity of Relevant Acronyms (February 2016)"
jargon "The Jargon File (version 4.4.7, 29 Dec 2003)"
foldoc "The Free On-line Dictionary of Computing (30 December 2018)"
easton "Easton's 1897 Bible Dictionary"
hitchcock "Hitchcock's Bible Names Dictionary (late 1800's)"
bouvier "Bouvier's Law Dictionary, Revised 6th Ed (1856)"
devil "The Devil's Dictionary (1881-1906)"
```

```
Run    Lab9ServerSocketMessage
250 ok

Definition of 'gold':
150 1 definitions retrieved
151 "gold" wn "WordNet (r) 3.0 (2006)"
gold
    adj 1: made from or covered with gold; "gold coins"; "the gold
           dome of the Capitol"; "the golden calf"; "gilded icons"
           [syn: {gold}, {golden}, {gilded}]
    2: having the deep slightly brownish color of gold; "long
       aureate (or golden) hair"; "a gold carpet" [syn: {aureate},
       {gilded}, {gilt}, {gold}, {golden}]
```

...for a complete human being
Texas
International college

Date: 05-10-2024

10. Write a program to get information about sockets.

Objective: To write a Java program that retrieves and displays information about a socket, including the remote addresses , local addresses and ports.

Steps:

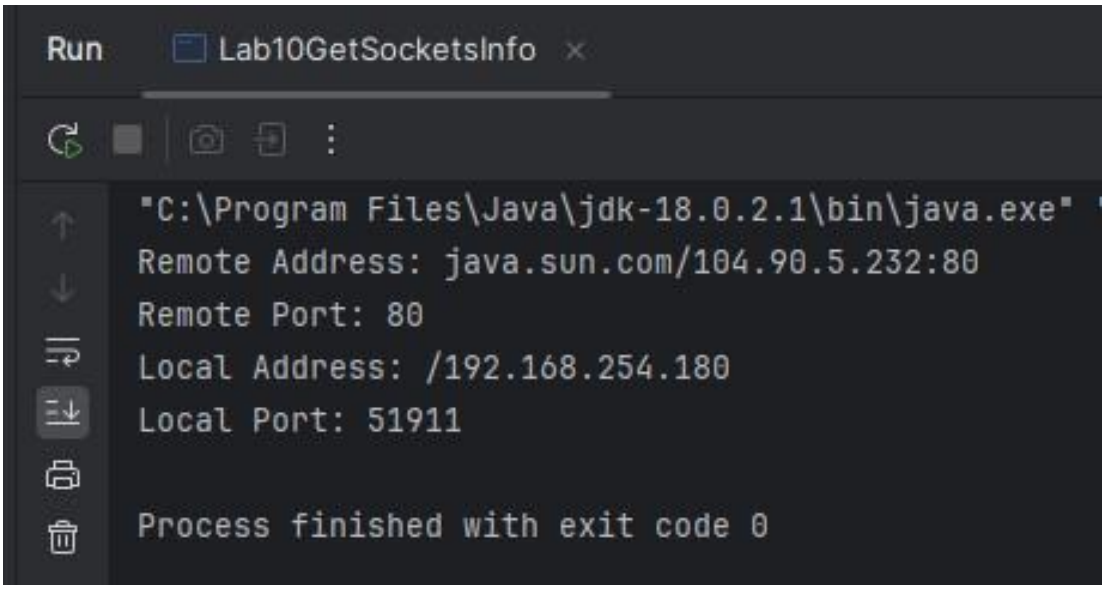
1. Create a Socket: Establish a socket connection to a specified host and port.
2. Retrieve Remote Information: Get and display the remote socket address and port using getRemoteSocketAddress() and getPort().
3. Retrieve Local Information: Get and display the local address and port using getLocalAddress() and getLocalPort().
4. Handle Exceptions: Implement exception handling for unknown host, socket, and I/O errors.

Code Snippet

```
import java.io.IOException;

import java.net.Socket;
import java.net.SocketException;
import java.net.UnknownHostException;

public class Lab10GetSocketsInfo {
    public static void main(String[] args) {
        String host = "java.sun.com";
```

```
        try {
            Socket theSocket = new Socket(host, 80);
            System.out.println("Remote Address: "+ theSocket.getRemoteSocketAddress());
            System.out.println("Remote Port: "+ theSocket.getPort());
            System.out.println("Local Address: "+ theSocket.getLocalAddress());
            System.out.println("Local Port: "+ theSocket.getLocalPort());
        } catch (UnknownHostException ex) {
            System.err.println("I can't find " + host);
        } catch (SocketException ex) {
            System.err.println("Could not connect to " + host);
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
}
```

Output:



...for a complete human being
Texas
International college

Date: 05-10-2024

11. Write a program to get information about server sockets.

Objective: To write a Java program that retrieves and displays information about a server socket, such as the local port and address where server is running.

Steps:

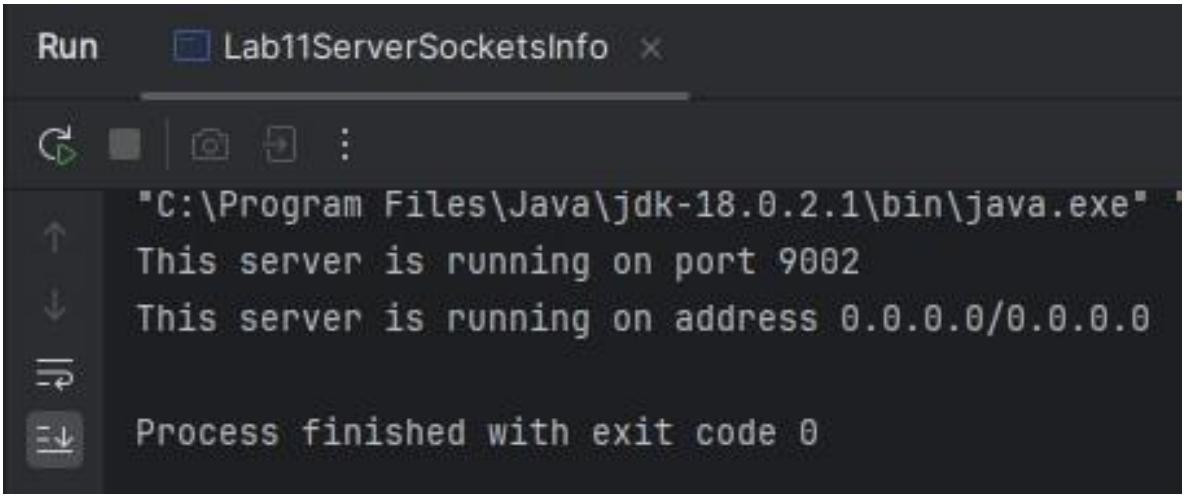
1. Create a Server Socket: Initialize a ServerSocket object on a unique port.
2. Retrieve Local Port: Simply use getLocalPort() to display the port number on which the server is running.
3. Retrieve Local Address: Use getInetAddress() to display the IP address the server is bound to.
4. Handle Exceptions: Implement exception handling for I/O errors during the server socket creation.

Code Snippet

```
import java.io.*;
import java.net.*;
public class Lab11ServerSocketsInfo {
    public static void main(String[] args) {
        try {
            ServerSocket server = new ServerSocket(9002);
            System.out.println("This server is running on port " + server.getLocalPort());
            System.out.println("This server is running on address " + server.getInetAddress());
```

```
            } catch (IOException ex) {
                System.err.println(ex);
            }
        }
    }
```

Output:



...for a complete human being

Texas International college

Date: 05-10-2024

12. Write a program to set options like TCP_NODELAY and SO_TIMEOUT on a socket.

Objective: To write a Java program that demonstrates the process of setting and retrieving options like TCP_NODELAY and SO_TIMEOUT on a socket.

Steps:

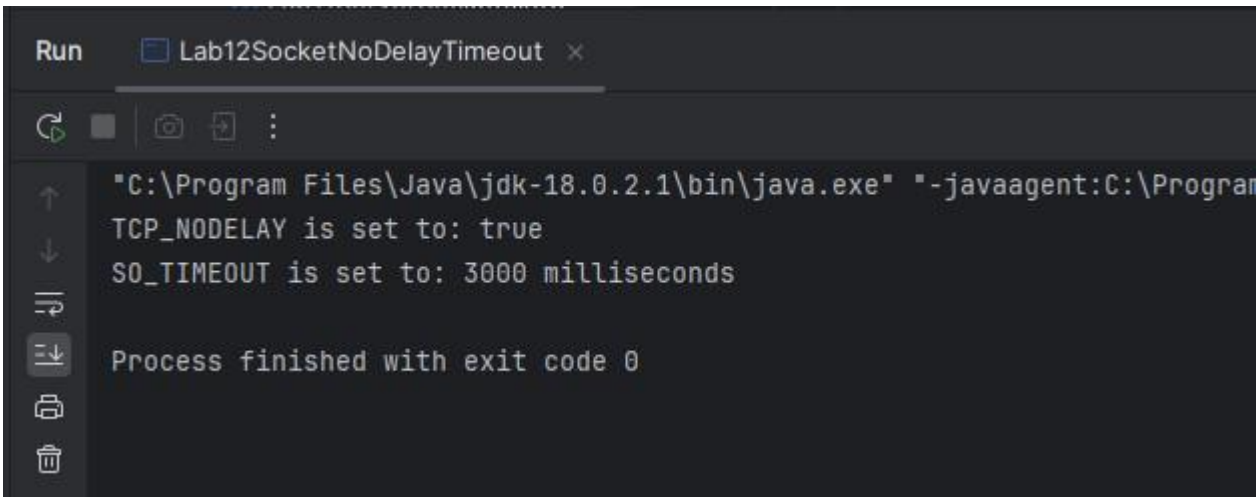
1. Create a Socket: Establish a socket connection to a server.
2. Set Options: Use setTcpNoDelay(true) to disable Nagle's algorithm and setSoTimeout() to set a timeout for read operations.
3. Retrieve and Display Options: Get the current state of TCP_NODELAY and SO_TIMEOUT using getTcpNoDelay() and getSoTimeout().
4. Close the Socket: Close the socket after the operation is complete.

Code Snippet

```
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;

public class Lab12SocketNoDelayTimeout {
    public static void main(String[] args) {
        String host = "www.example.com";
        int port = 80;
        try (Socket socket = new Socket(host, port)) {
            // Set TCP_NODELAY to true (disable Nagle's algorithm)
```

```
            socket.setTcpNoDelay(true);
            // Set SO_TIMEOUT to 3000 milliseconds (3 seconds)
            socket.setSoTimeout(3000);

            // Retrieve and display the current settings
            boolean tcpNoDelay = socket.getTcpNoDelay();
            int soTimeout = socket.getSoTimeout();
            System.out.println("TCP_NODELAY is set to: " + tcpNoDelay);
            System.out.println("SO_TIMEOUT is set to: " + soTimeout + " milliseconds");
        } catch (UnknownHostException ex) {
            System.err.println("Unknown host: " + host);
        } catch (IOException ex) {
            System.err.println("I/O error: " + ex.getMessage());
        }
    }
}
```

Output:



...for a complete human being

Texas International college

Date: 05-10-2024

13. Develop a server program that handles multiple client connections concurrently using threads.

Objective: To develop a server program that can handle multiple client connections concurrently using threads in java.

Steps:

1. Create a ServerSocket: Initialize a ServerSocket to listen for incoming client connections on a specified port.
2. Accept Client Connections: Continuously accept new client connections in a loop using server.accept().
3. Handle Connections with Threads: For each client connection, create a new thread to handle the client's requests.
4. Implement Client Handling Logic: Define a thread class that manages client communication, processes requests, and sends responses.

Code Snippet

<u>Server code</u>

```
import java.net.*;
import java.io.*;
import java.util.Date;

public class Lab13MultipleClientThread extends Thread {
    public final static int PORT = 9002;
```

```
public static void main(String[] args) {
    try (ServerSocket server = new ServerSocket(PORT)) {
        while (true) {
            try {
                Socket connection = server.accept();
                Thread task = new DaytimeThread(connection);
                task.start();
            } catch (IOException ex) {
            }
        }
    } catch (IOException ex) {
        System.err.println("Couldn't start server");
    }
}

private static class DaytimeThread extends Thread {
    private Socket connection;

    DaytimeThread(Socket connection) {
        this.connection = connection;
    }

    @Override
    public void run() {
        System.out.println("Thread started: " + this.getName());
        try {
            Writer out = new OutputStreamWriter(connection.getOutputStream());
            Date now = new Date();
            out.write(now.toString() + "\r\n");
            out.flush();
        } catch (IOException ex) {
```

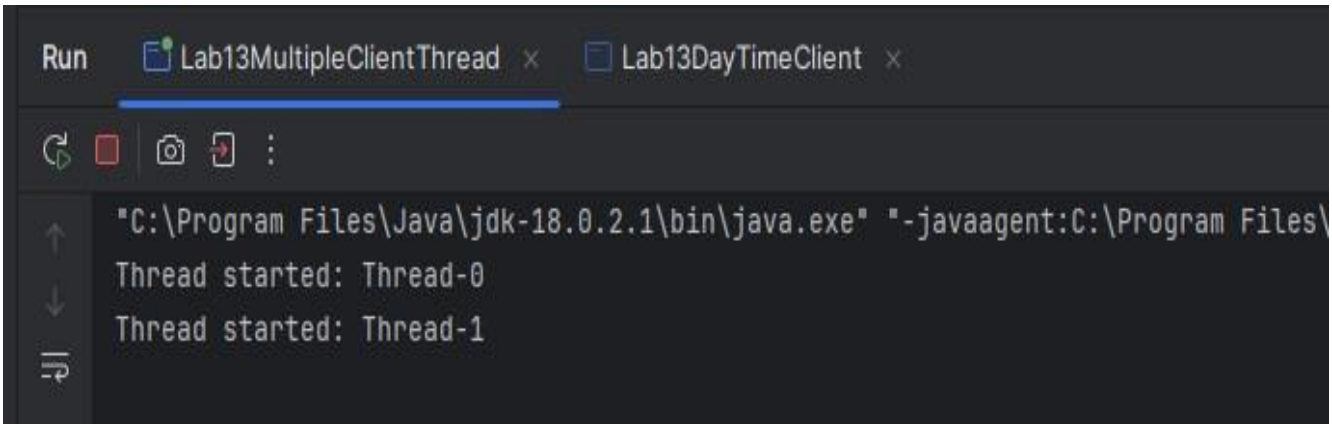
```
            System.err.println(ex);
        } finally {
            try {
                connection.close();
            } catch (IOException e) {
                // ignore;
            }
        }
    }
  }
}
```

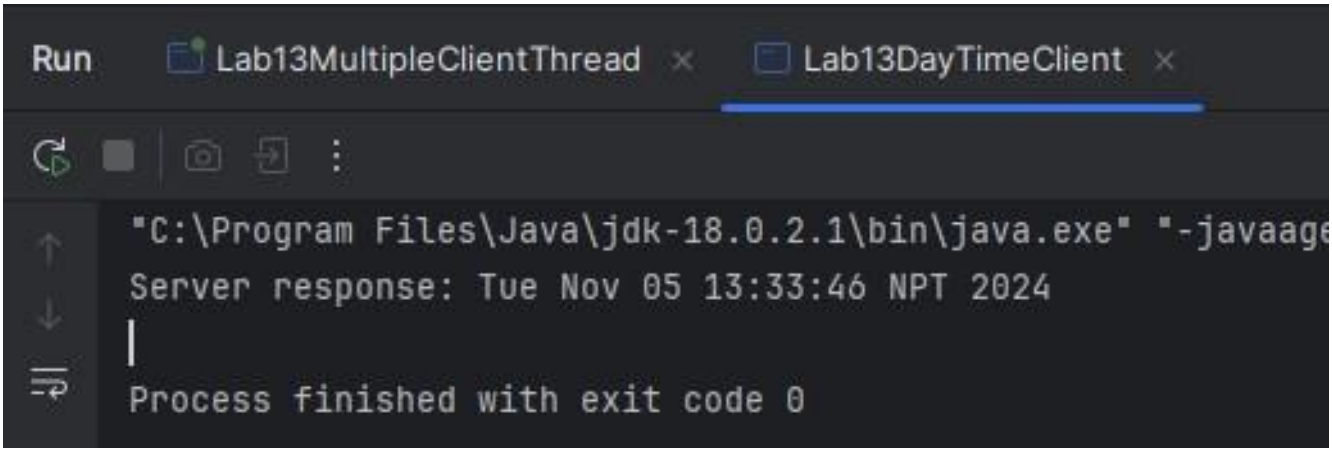
Client Code

```
import java.io.*;
import java.net.*;

public class Lab13DayTimeClient {
   public static final String SERVER_ADDRESS = "localhost";
   public static final int SERVER_PORT = 9002;

   public static void main(String[] args) {
      try {
         // Create a socket to connect to the server
         Socket socket = new Socket(SERVER_ADDRESS, SERVER_PORT);

         // Obtain input stream to read server response
         BufferedReader in = new BufferedReader(new
               InputStreamReader(socket.getInputStream()));

         // Read the response from the server
         String response = in.readLine();
```

```
            System.out.println("Server response: " + response);

            // Close the connection
            in.close();
            socket.close();
        } catch (IOException ex) {
            System.err.println("Error: " + ex.getMessage());
        }
    }
}
```

Output:





...for a complete human being
Texas
International college

Date: 05-10-2024

14. Write a program to create SSL client and server sockets for secure communication.

Objective: To create SSL client and server sockets for secure communication between a client and a server using java.

Steps:

1. First of all ,set up the keystore and truststore for the server and client.
2. Initialize SSLContext using the keystore and truststore.
3. Create SSL server and client sockets.
4. Establish a connection and perform exchange of messages securely.
5. Close the sockets after communication.

Code Snippet

```
import javax.net.ssl.*;
import java.io.*;
import java.security.KeyStore;

public class Lab14SSLServer {
    private static final int PORT = 8443;

    public static void main(String[] args) {
        try {
            // Load server keystore
            KeyStore keyStore = KeyStore.getInstance("JKS");
```

```
                keyStore.load(new FileInputStream("serverkeystore.jks"),
"password".toCharArray());

                // Set up KeyManagerFactory and SSLContext
                KeyManagerFactory keyManagerFactory =
KeyManagerFactory.getInstance("SunX509");
                keyManagerFactory.init(keyStore, "password".toCharArray());
                SSLContext sslContext = SSLContext.getInstance("TLS");
                sslContext.init(keyManagerFactory.getKeyManagers(), null, null);

                // Create SSLServerSocket
                SSLServerSocketFactory serverSocketFactory =
sslContext.getServerSocketFactory();
                SSLServerSocket serverSocket = (SSLServerSocket)
serverSocketFactory.createServerSocket(PORT);

                System.out.println("SSL Server started. Waiting for clients...");

                while (true) {
                   try (SSLSocket clientSocket = (SSLSocket) serverSocket.accept()) {
                  BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
                      PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

                      // Read and respond to client
                      String clientMessage = in.readLine();
                      System.out.println("Client: " + clientMessage);
                      out.println("Hello, client! Your message has been received.");
                   }
                }
            } catch (Exception e) {
```

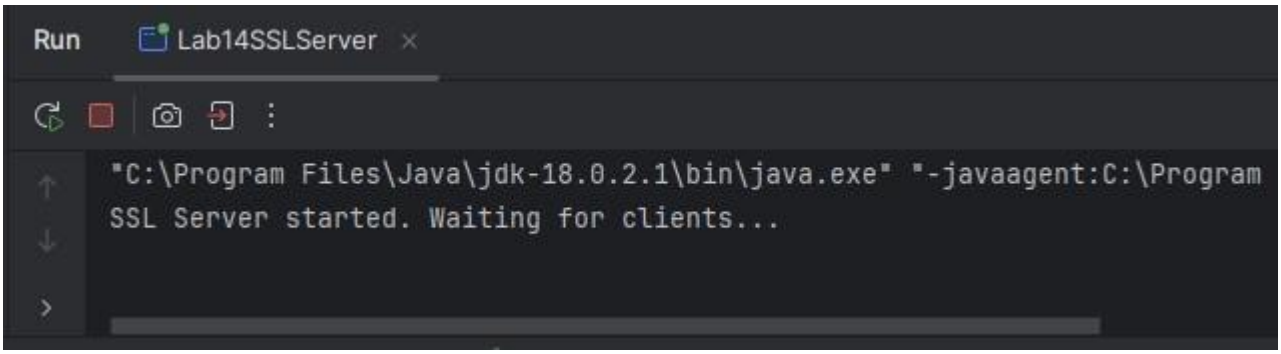
```
                e.printStackTrace();
            }
        }
    }
```

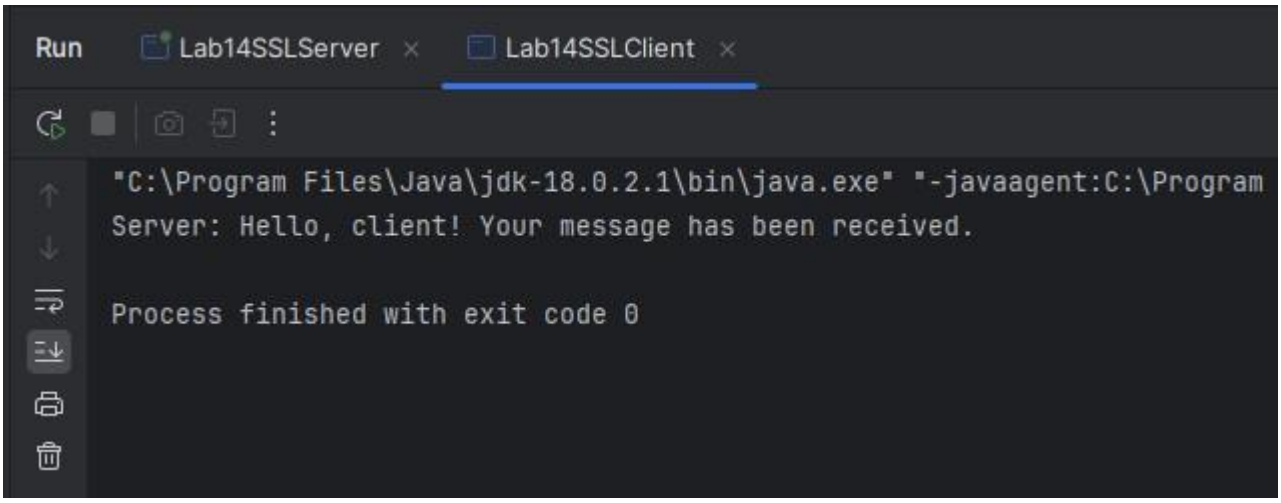
Client Code

```
import javax.net.ssl.*;
import java.io.*;
import java.security.KeyStore;

public class Lab14SSLClient {
    private static final String HOST = "localhost";
    private static final int PORT = 8443;

    public static void main(String[] args) {
        try {
            // Load client truststore
            KeyStore trustStore = KeyStore.getInstance("JKS");
            trustStore.load(new FileInputStream("clienttruststore.jks"),
"password".toCharArray());

            // Set up TrustManagerFactory and SSLContext
            TrustManagerFactory trustManagerFactory =
TrustManagerFactory.getInstance("SunX509");
            trustManagerFactory.init(trustStore);
            SSLContext sslContext = SSLContext.getInstance("TLS");
            sslContext.init(null, trustManagerFactory.getTrustManagers(), null);

            // Create SSLSocket
            SSLSocketFactory socketFactory = sslContext.getSocketFactory();
            try (SSLSocket socket = (SSLSocket) socketFactory.createSocket(HOST, PORT)) {
```

```
                PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
                BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

                // Send a message to the server
                out.println("Hello, server!");
                System.out.println("Server: " + in.readLine());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output:





Date: 05-10-2024

15. Write a program using NIO buffers and channels for non blocking data transfer.

Objective: To write a Java program using NIO SocketChannel and Selector for non-blocking network data transfer.

Steps:

1. Firstly ,set up a SocketChannel in non-blocking mode.
2. Register the SocketChannel with a Selector.
3. Handle non-blocking read/write operations using buffers.
4. Process the data asynchronously.

Code Snippet

```
package Lab;

import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.SocketChannel;
import java.util.Iterator;
import java.util.Set;

public class Lab15NIOBufferNonBlockingDataTransfer{
```

```
public static void main(String[] args) {
    try {
        // Create a non-blocking socket channel
        SocketChannel socketChannel = SocketChannel.open();
        socketChannel.configureBlocking(false);

        // Create a selector
        Selector selector = Selector.open();

        // Register the channel with the selector for connection
        socketChannel.register(selector, SelectionKey.OP_CONNECT);
        socketChannel.connect(new InetSocketAddress("example.com", 80));

        ByteBuffer buffer = ByteBuffer.allocate(256);

        while (true) {
            selector.select();
            Set<SelectionKey> selectedKeys = selector.selectedKeys();
            Iterator<SelectionKey> iter = selectedKeys.iterator();

            while (iter.hasNext()) {
                SelectionKey key = iter.next();

                if (key.isConnectable()) {
                    handleConnect(key);
                } else if (key.isReadable()) {
                    handleRead(key, buffer);
                }

                iter.remove();
            }
```

```
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void handleConnect(SelectionKey key) throws IOException {
        SocketChannel channel = (SocketChannel) key.channel();
        if (channel.isConnectionPending()) {
            channel.finishConnect();
        }
        channel.configureBlocking(false);

        // Register the channel for reading
        channel.register(key.selector(), SelectionKey.OP_READ);

        // Sending an HTTP GET request
        String request = "GET / HTTP/1.1\r\nHost: example.com\r\n\r\n";
        ByteBuffer buffer = ByteBuffer.wrap(request.getBytes());
        channel.write(buffer);
    }

    private static void handleRead(SelectionKey key, ByteBuffer buffer) throws IOException
{
        SocketChannel channel = (SocketChannel) key.channel();
        buffer.clear();
        int bytesRead = channel.read(buffer);
        if (bytesRead == -1) {
            channel.close();
        } else {
            buffer.flip();
```

```
                while (buffer.hasRemaining()) {
                    System.out.print((char) buffer.get());
                }
            }
        }
    }
```

Output:

```
Run    Lab14SSLServer ×    Lab15NIOBufferNonBlockingDataTransfer ×

"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
HTTP/1.1 200 OK
Accept-Ranges: bytes
Age: 394478
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Tue, 05 Nov 2024 08:18:38 GMT
Etag: "3147526947"
Expires: Tue, 12 Nov 2024 08:18:38 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECAcc (lac/559D)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 1256

<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
```

```
<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
    <p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
```

Date: 05-10-2024

16. Write a program to perform asynchronous read and write operations using AsynchronousSocketChannel.

Objective: To perform asynchronous read and write operations using AsynchronousFileChannel in java.

Steps:

1. Create a ByteBuffer in order to hold data for writing.
2. Use AsynchronousSocketChannel with a CompletionHandler for asynchronous writing.
3. Perform asynchronous reading using AsynchronousClientChannel..
4. Check and process the completion status of asynchronous operations.
5. Display file contents after read and write operations are completed.

Code Snippet

```
import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.AsynchronousSocketChannel;
import java.nio.channels.CompletionHandler;
import java.nio.charset.StandardCharsets;
import java.util.concurrent.ExecutionException;

public class Lab16AsyncEchoClient {
```

```
        private static final String HOST = "localhost";
        private static final int PORT = 5000;

        public static void main(String[] args) throws IOException, InterruptedException,
ExecutionException {
            AsynchronousSocketChannel clientChannel = AsynchronousSocketChannel.open();

            // Connect to the server
            clientChannel.connect(new InetSocketAddress(HOST, PORT)).get();
            System.out.println("Connected to server");

            // Send a message to the server
            ByteBuffer buffer = ByteBuffer.wrap("Hello, Async Server!".getBytes
(StandardCharsets.UTF_8));
            clientChannel.write(buffer, buffer, new CompletionHandler<Integer, ByteBuffer>() {
                @Override
                public void completed(Integer result, ByteBuffer buffer) {
                    System.out.println("Message sent to server.");

                    // Prepare to receive the echoed message
                    buffer.clear();
                    clientChannel.read(buffer, buffer, new CompletionHandler<Integer, ByteBuffer>()
{
                        @Override
                        public void completed(Integer result, ByteBuffer buffer) {
                            buffer.flip();
                            String response = StandardCharsets.UTF_8.decode(buffer).toString();
                            System.out.println("Received from server: " + response);

                            try {
                                clientChannel.close();
```

```
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            @Override
            public void failed(Throwable exc, ByteBuffer buffer) {
                System.err.println("Failed to read from server.");
                exc.printStackTrace();
            }
        });
    }

    @Override
    public void failed(Throwable exc, ByteBuffer buffer) {
        System.err.println("Failed to send message to server.");
        exc.printStackTrace();
    }
});

// Keep the client running until the response is received
Thread.sleep(5000); // Simple wait to allow asynchronous completion
}
}
```

```
import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.AsynchronousServerSocketChannel;
import java.nio.channels.AsynchronousSocketChannel;
import java.nio.channels.CompletionHandler;
import java.nio.charset.StandardCharsets;

public class Lab16AsyncEchoServer {
    private static final int PORT = 5000;

    public static void main(String[] args) throws IOException {
        AsynchronousServerSocketChannel serverChannel =
AsynchronousServerSocketChannel.open();
        serverChannel.bind(new InetSocketAddress(PORT));

        System.out.println("Asynchronous Echo Server is listening on port " + PORT);

        serverChannel.accept(null, new CompletionHandler<AsynchronousSocketChannel,
Void>() {
            @Override
            public void completed(AsynchronousSocketChannel clientChannel, Void
attachment) {
                // Accept the next connection
                serverChannel.accept(null, this);

                // Allocate a buffer for reading data
                ByteBuffer buffer = ByteBuffer.allocate(1024);

                // Start reading data from the client asynchronously
```

```
                    clientChannel.read(buffer, buffer, new CompletionHandler<Integer, ByteBuffer>()
{
                        @Override
                        public void completed(Integer result, ByteBuffer buffer) {
                            buffer.flip();
                            String receivedMessage = StandardCharsets.UTF_8.decode(buffer)
.toString();
                            System.out.println("Received from client: " + receivedMessage);

                            // Echo the message back to the client
                            clientChannel.write(ByteBuffer.wrap(("Echo: " +  receivedMessage)
.getBytes()) , null, new CompletionHandler<Integer, Void>() {
                                @Override
                                public void completed(Integer result, Void attachment) {
                                    System.out.println("Echoed message back to client.");
                                    try {
                                        clientChannel.close();
                                    } catch (IOException e) {
                                        e.printStackTrace();
                                    }
                                }

                                @Override
                                public void failed(Throwable exc, Void attachment) {
                                    System.err.println("Failed to write to client.");
                                    exc.printStackTrace();
                                }
                            });
                        }

                        @Override
```

```
                public void failed(Throwable exc, ByteBuffer attachment) {
                    System.err.println("Failed to read from client.");
                    exc.printStackTrace();
                }
            });
        }

        @Override
        public void failed(Throwable exc, Void attachment) {
            System.err.println("Failed to accept connection.");
            exc.printStackTrace();
        }
    });

    // Keep the server running
    try {
        Thread.currentThread().join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
  }
}
```

Output:

```
Run    Lab16AsyncEchoServer
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C:
Asynchronous Echo Server is listening on port 5000
```

```
Run    Lab16AsyncEchoServer    Lab16AsyncEchoClient
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C:
Connected to server
Message sent to server.
Received from server: Echo: Hello, Async S

Process finished with exit code 0
```

Date: 05-10-2024

17. Write a UDP client and server program to send and receive datagrams.

Objective: To implement a UDP client-server model in Java for sending and receiving datagrams in java.

Steps:

1. First,create a UDP server which listen on a specific port.
2. Receive a datagram packet and process it.
3. Send a response packet from the server to the client.
4. Create a UDP client that sends a request to the server.
5. Receive and display the response from the server.

Code Snippet

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class Lab17UDPServer {
    public static void main(String[] args) {
        int port = 8080;
        try {
            DatagramSocket socket = new DatagramSocket(port);
            DatagramPacket request = new DatagramPacket(new byte[1], 1);
            socket.receive(request);
```

```
            String quote = "This is the UDP server. And you are getting this message.";
            byte[] buffer = quote.getBytes();
            InetAddress clientAddress = request.getAddress();
            int clientPort = request.getPort();
            DatagramPacket response = new DatagramPacket(buffer, buffer.length,
                    clientAddress, clientPort);
            socket.send(response);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

UDP Client

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;


public class Lab17UDPClient {
    private final static int PORT = 8080;
    private static final String HOSTNAME = "localhost";

    public static void main(String[] args) {
        try {
            InetAddress address = InetAddress.getByName(HOSTNAME);
            DatagramSocket socket = new DatagramSocket();
            DatagramPacket request = new DatagramPacket(new byte[1], 1, address, PORT);
            socket.send(request);
```

```
                byte[] buffer = new byte[512];
                DatagramPacket response = new DatagramPacket(buffer, buffer.length);
                socket.receive(response);
                String quote = new String(buffer, 0, response.getLength());
                System.out.println(quote);
                System.out.println();
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
}
```

Output:

Date: 05-10-2024

18. Write a program to send and receive multicast packets using MulticastSocket.

Objective: To send and receive multicast packets over a network using MulticastSocket in java.

Steps:

1. Firstly,set up a multicast group by defining an IP address in the range 224.0.0.0 to 239.255.255.255 .
2. Create a MulticastSocket to send and receive data.
3. Join the multicast group to receive messages sent to that group.
4. Send and receive messages using DatagramPacket and display them on the console.

Code Snippet

```
package Lab;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;

public class Lab18MultiCastServer {
    public static void main(String[] args) {
        try {
            MulticastSocket socket = new MulticastSocket();
```

```
            InetAddress groupAddress = InetAddress.getByName("224.3.3.3");

            String message = "Hello, Multicast Group!";
            byte[] data = message.getBytes();
            DatagramPacket packet = new DatagramPacket(data, data.length, groupAddress,
                    4001);

            socket.setTimeToLive(64);
            for (int i = 0; i < 10; i++) {
                socket.send(packet);
                System.out.println("Sent message: " + message);
                Thread.sleep(1000); // Sleep for 1 second between messages
            }

            socket.close();

        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

MulticastClient

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;

public class Lab18MultiCastClient {
```

```
    public static void main(String[] args) {
        try {
            MulticastSocket socket = new MulticastSocket(4001);
            InetAddress groupAddress = InetAddress.getByName("224.3.3.3");
            socket.joinGroup(groupAddress);

            byte[] buffer = new byte[8192];
            while (true) {
                DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
                socket.receive(packet);
                String message = new String(packet.getData(), 0, packet.getLength(), "UTF-8");
                System.out.println("Received message: " + message);
            }

            // socket.leaveGroup(groupAddress);
            // socket.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
Run    Lab18MultiCastServer
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C
Sent message: Hello, Multicast Group!
Sent message: Hello, Multicast Group!
Sent message: Hello, Multicast Group!
Sent message: Hello, Multicast Group!
Sent message: Hello, Multicast Group!
Sent message: Hello, Multicast Group!
Sent message: Hello, Multicast Group!
Sent message: Hello, Multicast Group!
```

```
Run    Lab18MultiCastServer    Lab18MultiCastClient
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C
Received message: Hello, Multicast Group!
Received message: Hello, Multicast Group!
Received message: Hello, Multicast Group!
Received message: Hello, Multicast Group!
```

...for a complete human being
Texas
International college

Date: 05-10-2024

19. Develop an RMI server application and a corresponding client application to invoke remote methods.

Objective: To create a Java RMI server and client application where the client invokes a remote method on the server.

Steps:

1. First of all, define a remote interface that declares the methods which can be invoked remotely.
2. Implement the remote interface on the server-side by creating a class that defines the behavior of the remote methods.
3. Create an RMI server that registers the implementation of the remote interface with the RMI registry.
4. Develop an RMI client that looks up the remote object from the registry and invokes the remote methods.

Code Snippet

RMIServer

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;

public class Lab19RMIServer extends Lab19ImplementExample {
```

```
        public Lab19RMIServer() {}

        public static void main(String args[]) {
            try {
                // Instantiating the implementation class
                Lab19ImplementExample obj = new Lab19RMIServer();

                // Exporting the object of implementation class (stub)
                Lab19HelloInterface stub = (Lab19HelloInterface)
UnicastRemoteObject.exportObject (obj, 0);

                // Binding the remote object (stub) in the registry
                Registry registry = LocateRegistry.createRegistry(9000);
                registry.bind("Hello", stub);

                System.out.println("Server ready");
            } catch (Exception e) {
                System.err.println("Server exception: " + e.toString());
                e.printStackTrace();
            }
        }
    }
```

RMIClient

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Lab19RMIClient {
```

```
    private Lab19RMIClient() {}

    public static void main(String[] args) {
        try {
            // Getting the registry
            Registry registry = LocateRegistry.getRegistry(9000);

            // Looking up the registry for the remote object
            Lab19HelloInterface stub = (Lab19HelloInterface) registry.lookup("Hello");

            // Calling the remote method using the obtained object
            stub.printMsg();

            // Print confirmation on the client side
            System.out.println("Remote method invoked successfully.");

        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}

// Implementing the remote interface
public class Lab19ImplementExample implements Lab19HelloInterface {

    // Implementing the interface method
    public void printMsg() {
        System.out.println("This is an example RMI program");
    }
}
```

```
import java.rmi.Remote;
import java.rmi.RemoteException;

// Creating Remote interface for our application
public interface Lab19HelloInterface extends Remote {
    void printMsg() throws RemoteException;
}
```

Output: