Unit-6 (Socket for Client)

- A socket is a mechanism for allowing communication between processes, be it programs running on the same machine or different computers connected on a network.
- The socket provides bidirectional **FIFO** Communication facility over the network.
- Each socket has a specific address. This address is composed of an **IP address** and a **port number**.
- Socket are generally employed in **client server applications**.
- The server creates a socket, attaches it to a **network port addresses** then waits for the client to contact it.
- The client creates a **socket** and then attempts to connect to the **server socket**. When the connection is established, transfer of data takes place.

A socket is a connection between two hosts. It can perform seven basic operations:

• Connect to a remote machine

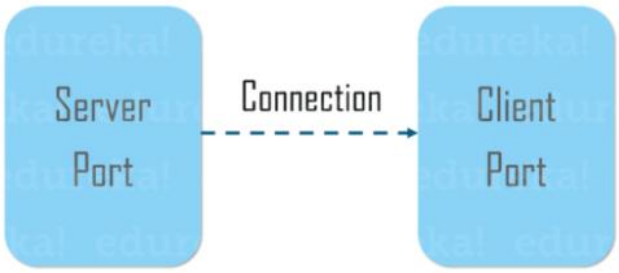
• Send data

• Receive data

• Close a connection

• Bind to a port Listen for incoming data

• Accept connections from remote machines on the bound port



Types of Sockets : There are two types of Sockets: the **datagram** socket and the **stream** socket.

1. **Datagram Socket :** A datagram socket uses the **User Datagram Protocol (UDP)** for sending messages. **UDP** is a much simpler protocol as it does not provide any of the delivery guarantees that TCP does. It is similar to mailbox. The letters (data) posted

into the box are collected and delivered (transmitted) to a letterbox (receiving socket).

2. **Stream Socket** A stream socket uses the **Transmission Control Protocol (TCP)** for sending messages. network socket which provides a **connection-oriented, sequenced, and unique flow of data without record.** It is similar to phone. A connection is established between the phones (two ends) and a conversation (transfer of data) takes place.

Investigating Protocols with Telnet

TELNET stands for **Tel**etype **Net**work. It is a type of protocol that enables one computer to connect to the local computer. It is used as a standard **TCP/IP protocol**.

Telnet is command line tool, no graphical user interface

Telnet is very fast

Developed in 1969

All command are sent in clear text

No Encryption

Telnet may be used in local network and working with older equipment

SSH(Secure Shell) is better option which has encryption and protect data from stolen

To enable telnet >go to control panel> click uninstall programs icon> select add or remove from features from left side tab> than check **enable telnet client**

In windows >go to cmd

You can type<telnet towel.blinkenlights.nl

Shows ascii version of star war movies

You can also play chess games using telnet

<telnet freechess.org 5000>

By default, Telnet attempts to connect to port 23. To connect to servers on different ports, specify the port you want to connect to like this:

telnet localhost 23

Reading from Servers with Sockets

Basic step to read data from servers with sockets

1. Open a socket.
2. Open an input stream and output stream to the socket.
3. Read from and write to the stream according to the server's protocol.
4. Close the streams.
5. Close the socket.

Reading from Server

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

public class App {
    public static void main(String[] args) throws IOException {
       Socket socket = new Socket("time.nist.gov", 13);//1.connect to remote
machine
      socket.setSoTimeout(15000);
      InputStream in = socket.getInputStream();//2. open stream
      StringBuilder time = new StringBuilder();
      InputStreamReader reader = new InputStreamReader(in, "ASCII");//3. read
data
      int c;
      while ((c = reader.read()) != -1)
        {
        time.append((char) c);
        }
      reader.close();//4. close the stream
      socket.close();//5. Close the socket
      System.out.println(time);


    }

}
```

Output:

60058 23-04-24 05:04:08 50 0 0 131.2 UTC(NIST) *

Another Example:

Lab: Write a program to read data from server using socket.

Reading from Server

Create MyClient.java class

```
import java.io.*;
import java.net.*;
public class MyClient {
    public static void main(String[] args) throws IOException {
        // Create a socket to connect to the server
        Socket clientSocket = new Socket("localhost", 1234);
        // Create input stream to read data from the server
        BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));//read data so InputStream
        // Read data from the server
        String serverResponse = in.readLine();
        System.out.println("Server says: " + serverResponse);
        // Close the connection
        clientSocket.close();
    }

}
```

Create MyServer.java

```
import java.io.*;
import java.net.*;
public class MyServer {
    public static void main(String[] args) throws IOException {
        // Create a server socket
        ServerSocket serverSocket = new ServerSocket(1234);
        // Wait for a client to connect
        Socket clientSocket = serverSocket.accept();
        System.out.println("Connected to Client");
        // Create output stream to send data to the client
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
true);//send data so output Stream
        // Send the current date and time to the client
        out.println("Hello from server");
        // Close the connection
        clientSocket.close();
        serverSocket.close();
    }
}

    }
```

```
    }
}
```

Output:
Connected to localhost on port 1234
Server says: Hello from server

Writing to Server with Socket

Lab: Write a java program to write data to server using socket.

MyClient.java

```
import java.io.*;
import java.net.*;
public class MyClient {
    public static void main(String[] args) throws IOException {
            // Create a socket to connect to the server
            Socket client = new Socket("localhost", 1234);
            // Create output stream to send data to the server
            PrintWriter out = new PrintWriter(client.getOutputStream(),
true);//send data so output stream
            // Send data to the server
            out.println("Hello from client");
            // Close the connection
            client.close();
    }
}
```

MyServer.java

```
import java.io.*;
import java.net.*;
public class MyServer {
    public static void main(String[] args) throws IOException {
        // Create a server socket
        ServerSocket serverSocket = new ServerSocket(1234);
        System.out.println("Server started");
        // Wait for a client to connect
        Socket server = serverSocket.accept();
        // Create input stream to read data from the client
        BufferedReader in = new BufferedReader(new
InputStreamReader(server.getInputStream()));//read data so Inputstream
        // Read and print the message from the client
        String clientMessage = in.readLine();
        System.out.println("Client says: " + clientMessage);
        // Close the connection
        server.close();
        serverSocket.close();
```

```
    }
}
```

Output:

Exam Question Lab: write a server side program for daytime service using socket.

MyServer.java

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

public class MyServer {
    public static void main(String[] args) throws IOException {
        // Create a server socket
        ServerSocket serverSocket = new ServerSocket(1234);
        // Wait for a client to connect
        Socket clientSocket = serverSocket.accept();
        System.out.println("Connected to Client");
        // Get the current date and time
        String currentDateTime = new Date().toString();
        // Create output stream to send data to the client
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
        // Send the current date and time to the client
        out.println(currentDateTime);
        // Close the connection
        clientSocket.close();
        serverSocket.close();
    }
}
```

MyClient.java

```
import java.io.*;
import java.net.*;
public class MyClient {
    public static void main(String[] args) throws IOException {
        // Create a socket to connect to the server
        Socket clientSocket = new Socket("localhost", 1234);
        // Create input stream to read data from the server
        BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

        // Read data from the server
```

```
        String serverResponse = in.readLine();
        System.out.println("Server says: " + serverResponse);

        // Close the connection
        clientSocket.close();
    }

}
```

Constructing and Connecting Sockets

The **java.net.Socket** class is Java's fundamental class for performing client-side TCP operations.

TCP network connections such as **URL**, **URLConnection**, **Applet**, and **JEditorPane** all ultimately end up invoking the methods of this class.

Basic Constructor

- Each socket constructor specifies the **host** and the **port** to connect
- Host may be specified as an **InetAddress** or a string
- Remote ports are specified as int values from **1 to 65535**:
- Public **Socket**(string host, int port) throws **UnknownHostExcetion**,**IOException**
- Public **Socket**(InetAddress host, int port) **throws IOException**

Picking a Local Interface to Connect From:

Two constructors specify both the **host** and **port** to **connect to** and the **interface** and **port** to **connect from**:

- `public Socket(String host, int port, InetAddress interface, int localPort) throws IOException, UnknownHostException`

Used to create a socket and connect it to a specified remote host and port, while binding it to a specific local network interface and local port.

- **String host**: The hostname or IP address of the remote server you want to connect to.

- **int port**: The port number on the remote server to which you want to connect.
- **InetAddress localAddr**: The local network interface's IP address that the socket should bind to. This specifies which local network interface to use for the connection.
- **int localPort**: The port number on the local machine to bind the socket to. If this value is `0`, the system will choose any free port.

- `public Socket(InetAddress host, int port, InetAddress interface, int localPort) throws IOException`

Used to create a socket and connect it to a specified remote address and port, while binding it to a specific local network interface and local port.

Example:

```
     String host="www.oreilly.com";
            int port = 80;
            // Local interface details
            InetAddress localAddr = InetAddress.getByName("192.168.1.2");
            int localPort = 0; // Automatically assign an available port

            // Create and connect the socket
            Socket socket = new Socket(host, port, localAddr, localPort);
            System.out.println("Connected to remote host " + host + " from local
address " + localAddr + ":" + socket.getLocalPort());
```

Example:

```
InetAddress host = InetAddress.getByName("www.example.com");
            int port = 80;

            // Local interface details
            InetAddress localAddr = InetAddress.getByName("192.168.1.100");
            int localPort = 0; // Automatically assign an available port

            // Create and connect the socket
            Socket socket = new Socket(host, port, localAddr, localPort);
            System.out.println("Connected to remote host " + host + " from local
address " + localAddr + ":" + socket.getLocalPort());
```

Constructing Without Connecting:

- All the constructors we have talked about so far both create the socket object and open a network connection to a remote host.
- If you give no arguments to the Socket constructor, it has nowhere to connect to

public Socket();

- you can connect later by passing a **SocketAddress** to one of the **connect()** method.

Eample:

```
try {
     Socket socket = new Socket();
     // fill in socket options
     SocketAddress address = new InetSocketAddress("time.nist.gov", 13);
     socket.connect(address);//connect() Method
     // work with the sockets...
     } catch (IOException ex) {
     System.err.println(ex);
     }
```

Socket Address Class

The `SocketAddress` class represents a connection endpoint.
SocketAddress class is to provide a convenient store for socket connection information such as the **IP address** and **port** that can be reused to create new sockets.
Socket class offers two methods:

```
public SocketAddress getRemoteSocketAddress()
returns the address of the system being connected to i.e. server address
public SocketAddress getLocalSocketAddress()
return the address from which the connection is made i.e. client address
```

Example:
Lab:WRITE a java program to get socket information using SocketAddress class.

```
Socket socket = new Socket("www.yahoo.com", 80);
SocketAddress a = socket.getRemoteSocketAddress();//get hostname and ip
SocketAddress b = socket.getLocalSocketAddress();//Fetch Local ip and Port
socket.close();
```

Getting information about a Socket

Socket objects have several properties that are accessible through getter methods:

Remote address: IP and Web Address

public InetAddress getInetAddress()

Remote port

public int getPort()//server port 13

Local address

public InetAddress GetLocalAddress()://Ip Address

Local Port

public int getLocalPort();//eg.577788

isClosed: method return true if the socket is closed, false if it isn't

if(socket.isClosed()

{

}

IsConnected(): tell you if the socket is currently connected to remote host

If(socket.isConnected())

{

}

toString()

the socket class overrides only one of the standard methods from

java.lang.Object.toString()

The toString() method produces a string that looks like this:

```
Socket[addr=www.oreilly.com/198.112.208.11,port=80,localport=500
55]
```

Lab: Write a program to find WebAddress, IPAddress,Server Port, Local Port, Local IPAddress from socket with host address www.oreilly.com

Example:

```
public static void main(String[] args) throws Exception {
     String host="www.oreilly.com";
       try {

       Socket theSocket = new Socket(host, 80);
       System.out.println("Connected to " + theSocket.getInetAddress()
       + " on port " + theSocket.getPort() + " from port "
       + theSocket.getLocalPort() + " of "
       + theSocket.getLocalAddress());
       } 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);
```

```
        }

  }
```

Lab: Write a program to perform basic two way operation between client and server socket using java

MyClient.java

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

public class MyClient {
    public static void main(String[] args) throws IOException {
         Socket clientSocket = new Socket("localhost", 1234);
          // Set up input and output streams
         PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
         BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        int a=5;
        int b=6;
        out.println(a + " " + b);
        String serverResponse = in.readLine();
        System.out.println("Sum from server: " + serverResponse);
        clientSocket.close();
    }
}
```

MyServer.java

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

public class MyServer {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(1234);
        Socket clientSocket = serverSocket.accept();
        System.out.println("Connected to Client");
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
true);
        BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
       // Read the numbers from the client
       String inputLine = in.readLine();
       System.out.println("Received from client: " + inputLine);
```

```
        String[] numbers = inputLine.split(" ");
        int a = Integer.parseInt(numbers[0]);
        int b = Integer.parseInt(numbers[1]);
        int sum = a + b;
         System.out.println("Calculated sum: " + sum);
         out.println(sum);
         clientSocket.close();
         serverSocket.close();
    }

}
```

Setting Socket Options:

This **java.net.SocketOption** class allows end-user to generate the **socketOptions** objects that are useful in **invoking all basics socket operations** We can do networking operations such as **sending, reading data and closing to connections**.
SocketOption class is invoking a **stream SocketOption** and which is connected to the **simplified port number** and **port addressed**.

Java supports nine options for client-side sockets:

- TCP_NODELAY
- SO_BINDADDR
- SO_TIMEOUT
- SO_LINGER
- SO_SNDBUF
- SO_RCVBUF
- SO_KEEPALIVE
- OOBINLINE
- IP_TOS

TCP_NODELAY

```
    public void setTcpNoDelay(boolean on) throws SocketException
```

```
public boolean getTcpNoDelay() throws SocketException
```

Setting TCP_NODELAY to true ensures that packets are sent **as quickly** as possible regardless of their size.

Example:

```
if (!s.getTcpNoDelay())
      s.setTcpNoDelay(true);
```

SO_LINGER

- Specify a linger-on-close(Stay Long than necessary) **timeout**.
- Valid for (client) Sockets.

```
public void setSoLinger(boolean on, int seconds) throws SocketException
public int getSoLinger() throws SocketException
```

Example:

```
if (s.getTcpSoLinger() == -1) s.setSoLinger(true, 240);
```

SO_TIMEOUT

```
public void setSoTimeout(int milliseconds) throws SocketException
public int getSoTimeout() throws SocketException
```

- Specify a **timeout** on blocking socket operations.
- Valid for all sockets: Socket, **ServerSocket**, **DatagramSocket**

Example:

```
if (s.getSoTimeout() == 0) s.setSoTimeout(180000);
```

SO_RCVBUF and SO_SNDBUF

SO_RCVBUF

- **Set** the size of the underlying buffers for outgoing network I/O.
- Valid for all sockets: Socket, ServerSocket, DatagramSocket.

SO_SNDBUF

- **Get** the size of the buffer actually used by the platform when receiving in data on this socket.
- Valid for all sockets: Socket, **ServerSocket**, **DatagramSocket**.

```
public void setReceiveBufferSize(int size) throws SocketException,
IllegalArgumentException
public int getReceiveBufferSize() throws SocketException
public void setSendBufferSize(int size) throws SocketException,
IllegalArgumentException
public int getSendBufferSize() throws SocketException
```

SO_KEEPALIVE

- Turn on socket keepalive.
- Valid for Socket

```
public void setKeepAlive(boolean on) throws SocketException
public boolean getKeepAlive() throws SocketException
```

Example:

```
if (s.getKeepAlive()) s.setKeepAlive(false);
```

OOBINLINE

- Once OOBINLINE is turned on, any urgent data that arrives will be place on the socket's input stream to be read in the usual way. Java doesnot distinguish it from nonurgent data.
- If you want to receive **urgent data** inline with regular data, you need to set the OOBINLINE option to true using these methods:

Methods:

```
public void setOOBInline(boolean on) throws SocketException
public boolean getOOBInline() throws SocketException
```

Example:

```
if (!s.getOOBInline()) s.setOOBInline(true);
```

SO_REUSEADDR

When a socket is closed, it may not immediately release the local port, especially if a connection was open when the socket was closed.

If the SO_REUSEADDER is turned on (it's turned off by default), another socket is allowed to bind to the port even while data may be remaining for the previous socket.

Methods:

```
public void setReuseAddress(boolean on) throws SocketException
public boolean getReuseAddress() throws SocketException
```

IP_TOS Class of Service

This option sets the **type-of-service** or **traffic** class **field in the IP header** for a TCP or UDP socket.

```
public int getTrafficClass() throws SocketException
public void setTrafficClass(int trafficClass) throws SocketException
```

WHOIS Protocol.
Whois can be used to gather information about specific hosts and domains.

Step to connect client to to a whois server.

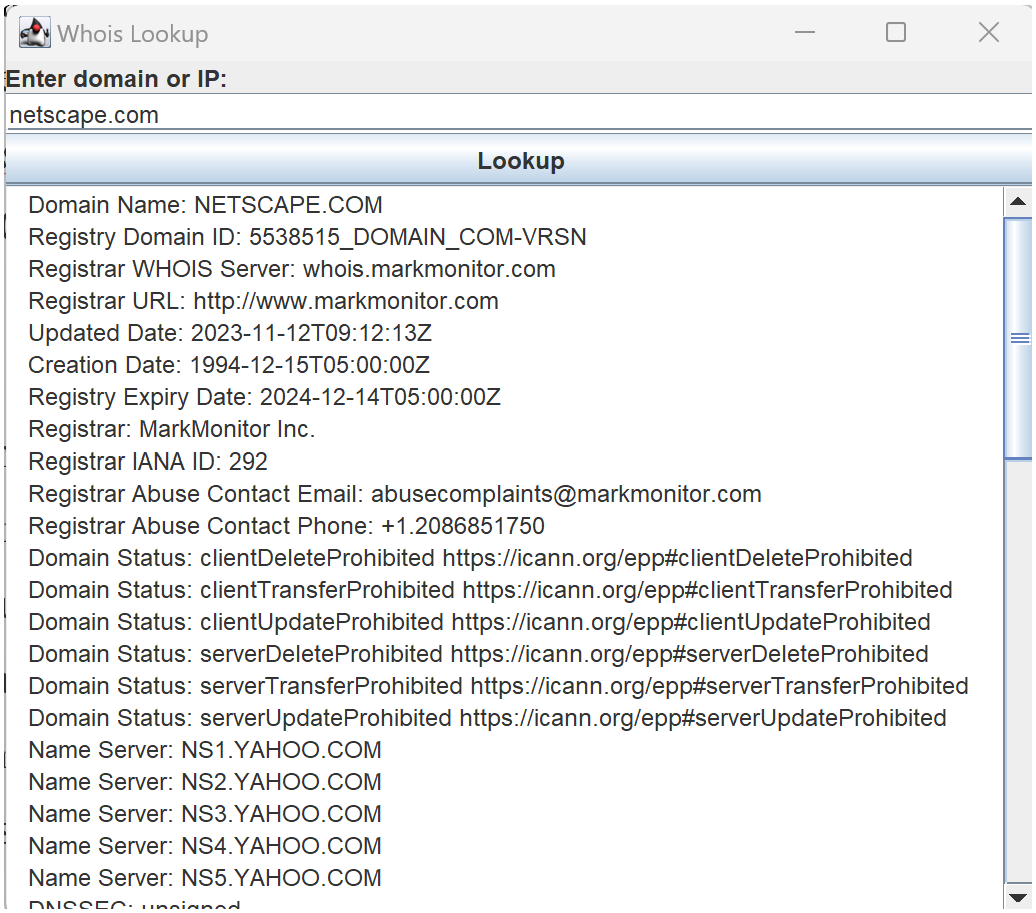
1. The client opens a TCP socket to port 43 on the server.

   2. The client sends a search string terminated by a carriage return/linefeed pair (\r\n).

   The search string can be a name, a list of names, or a special command. You can also search for domain names, like www.oreilly.com or netscape.com, which give you information about a network.

   3. The server sends an unspecified amount of human-readable information in response to the command and closes the connection.

   4. The client displays this information to the user.



WHOIS Perfix

Prefix	Meaning
Domain	Find only domain records.
Gateway	Find only gateway records.
Group	Find only group records.
Host	Find only host records.
Network	Find only network records.
Organization	Find only organization records.
Person	Find only person records.
ASN	Find only autonomous system number records.
Handle or !	Search only for matching handles.
Mailbox or @	Search only for matching email addresses.
Name or :	Search only for matching names.
Expand or *	Search only for group records and show all individuals in that group.
Full or =	Show complete record for each match.
Partial or suffix	Match records that start with the given string.
Summary or $	Show just the summary, even if there's only one match.
SUBdisplay or %	Show the users of the specified host, the hosts on the specified network, etc.

A network Client Libraries:

Java client library for WHOIS API: *https://github.com/whois-api-llc/whois-api-java*

```
import java.io.IOException;
import org.apache.commons.net.whois.WhoisClient;

public class JavaWhoisClient {

    public static void main(String[] args) {
        WhoisClient whois;

        whois = new WhoisClient();

        try {
            whois.connect(WhoisClient.DEFAULT_HOST);
            System.out.println(whois.query("=google.com"));
            whois.disconnect();
        } catch (IOException e) {
            System.err.println("Error I/O exception: " + e.getMessage());
            return;
        }
    }

}
```