Distributed Programming in Java

Session: 8

Networking

Objectives

- Explain the basic concept of networking
- Explain communication using Transmission Control Protocol (TCP)
- Explain URLConnection class
- Explain Socket and ServerSocket class
- Explain Datagram
- Explain Network Interface
- Explain Cookie

Overview of Networking

- Networking refers to the concept of various systems working together across a network.
- Programming at the application layer allows this communication over a network.
- `java.net` supports two common network protocols.
  - Transmission Control Protocol (TCP)
  - User Datagram Protocol (UDP)

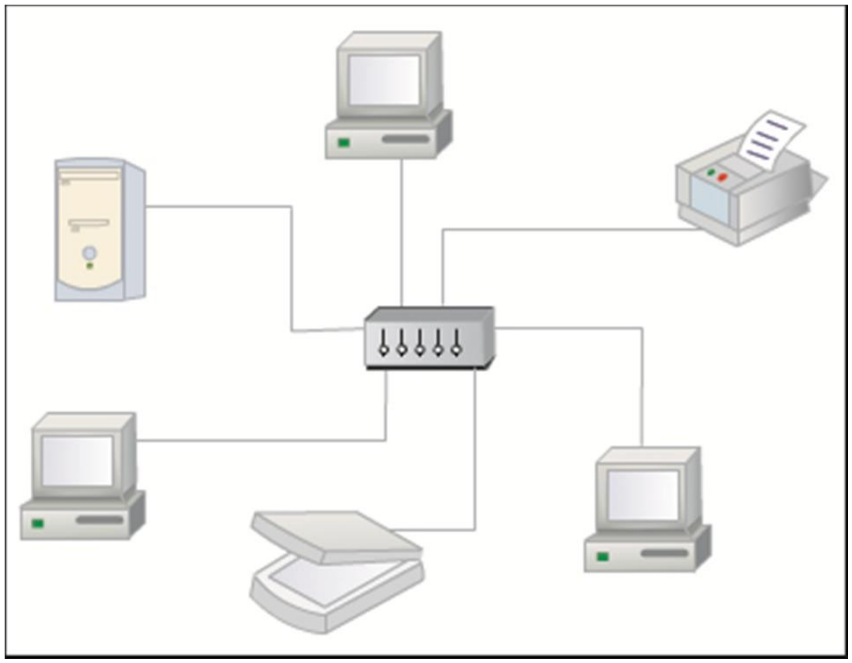
Introduction to Networking

- Transmission Control Protocol (TCP) is a connection-based protocol that provides a reliable flow of data between two computers.
- On the Internet, computers communicate with each other using either the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP).

Layer	Name	Functionality	Description
IV	Application	HTTP, FTP	This is where the 'high level' protocols such as File Transfer Protocol (FTP) and Hypertext Transfer Protocol (HTTP) operate.
III	Transport	TCP, UDP	This layer deals with opening and maintaining connections, and ensures that packets are transmitted and received.
II	Network	IP	This layer defines Internet Protocol (IP) addresses, and deals with packet transmission from one IP address to another.
I	Link	IP	This layer describes the physical equipment required for communications, such as twisted pair cables.

Communication on Computers

- When two applications on remote machines want to communicate with each other reliably, they establish a connection first.
- Once the connection is established, the applications can send data back and forth over that connection.
- TCP guarantees that data sent from one end of the connection is received at the other end.
- It also guarantees that the data received is in the same order as it was sent.
- Figure displays the communication between computers.



Transmission Control Protocol (TCP)

- TCP provides a point-to-point channel for applications that require reliable connections.
- HTTP, FTP, and Telnet are examples of applications that require reliable connections.
- Typically, you do not write programs at the lower level like transport, instead you generally program at the application level.
- Java provides the `java.net` package which contains all the necessary classes to perform system-independent network communications.

User Datagram Protocol (UDP)

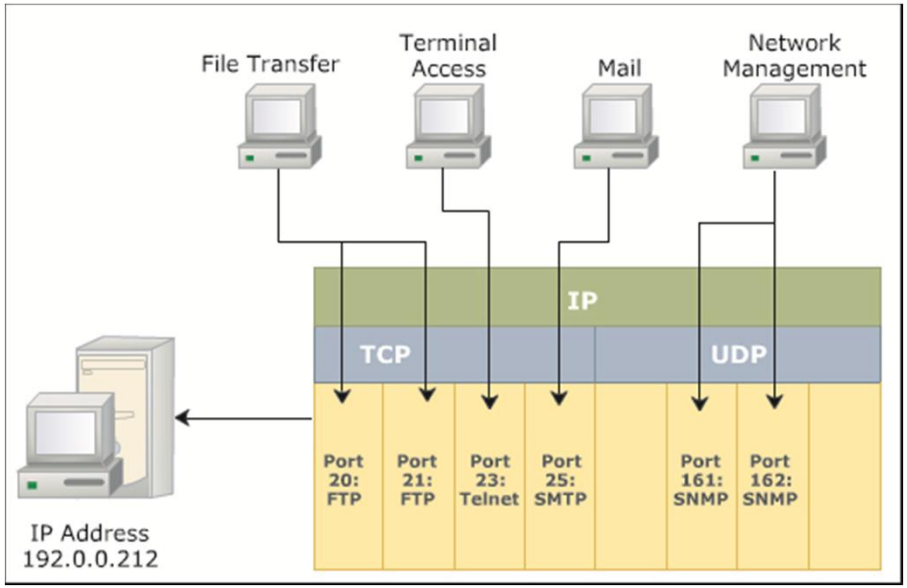
- A User Datagram Protocol (UDP) is a connectionless protocol.
- A connection between the sender and the receiver is not established before communication.
- A single socket connection can be used to send messages to multiple servers.
- The message is a datagram of fixed length termed as a record.

Networking Basics

- A computer generally has a single physical connection available for the network.
- The physical connections are the two serial ports COM1: and COM2:.
- One of these is usually used by a pointing device like the mouse.
- That leaves only a single physical connection for the network.
- Several applications running on a machine need to use this single physical connection for communication.
- If data arrives at these physical connections, there is no means to identify the application to which it should be forwarded.
- Hence, data being sent and received on the physical connection are based on the concept of ports.

Concept of a Port

- The physical connection is logically numbered within a range of `0` to `65535`. They are called as Ports.
- The port numbers ranging from `0` to `1023` are reserved.
- They are reserved for use by well known services such as HTTP, FTP, and other system services.
- Your applications should not use any of the port numbers in this range.
- Data transmitted over the Internet is accompanied with the destination address and the port number.
- The destination address identifies the computer, and the port number identifies the application.
- Figure displays the port.



Introduction to URL

- URL is an acronym for Uniform Resource Locator.
- It is a reference or an address to a resource on the Internet.
- The resource can be an HTML page, an image, or simply any file.
- You provide this URL to your Web browser so that it can locate the resource on the Internet.
- The concept of URL is similar to an address on a letter.
- The address is used by the post office to locate your house.
- A URL is a string that takes the form `protocol://resource`.
- It has two main parts:
  - Protocol Identifier
  - Resource Name

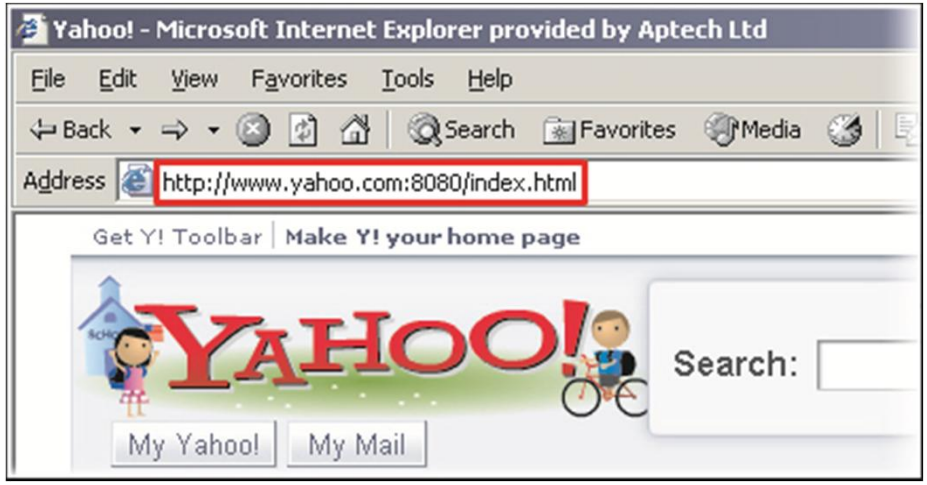
Components of URL

- Table lists the resource name contains one or more components.

Components	Description
Host Name	The name of the machine where the resource resides
File Name	The path to the file on the host
Port Name	The port number to which to connect (optional)

Retrieve a Resource

- To retrieve a resource `index.html` on a host `www.yahoo.com` using port number `80`, the URL would be as follows:
  - `http://www.yahoo.com:80/index.html`.
- Since the port number is optional, so the URL can be specified as follows:
  - `http://www.yahoo.com/index.html`
- Since the file name `index.html` is the default for most Web sites on the Internet, the URL can be further simplified as:
  - `http://www.yahoo.com`
- Figure displays the address to retrieve a resource.



Creating a URL

- The `java.net` package has a class `URL` which can be used to construct an `URL` object.
- The `URL` class has several constructors, all of which throw a `MalformedURLException` if no protocol is specified, or an unknown protocol is found.
- An instance of `URL` class is created using one of the following constructors as mentioned:
  - `URL(String url)`
  - `URL(String protocol, String host, String file)`
  - `URL(String protocol, String host, int port, String file)`
  - `URL(URL baseURL, String relativeURL)`

Methods of URL Class

- `public String getHost()`: This method returns the host name of the URL. This method returns the IP address enclosed in square brackets.
- `public String getFile()`: This method returns the file name of the URL. If the URL has a query string attached, then the query string is also concatenated and returned.
- `public String getPath()`: This method returns only the path of the URL. If the URL has a query string, they are not returned.
- `public final InputStream openStream()`: Opens a connection to a URL and returns an `InputStream` for reading from that connection. The `InputStream` returned can be wrapped within a `BufferedReader` to read directly from the URL.
- `public String getQuery()`: The `getQuery()` method is used to retrieve the query part of the URL.
- `public final Object getContent() throws IOException` : The `getContent()` method is used to retrieve the contents of the URL.
- `public int getDefaultPort()`: The `getDefaultPort()` method is used to retrieve the default port associated with the given URL.
- `public String getProtocol()` : The `getProtocol()` method is used to retrieve the protocol name of the URL.

Methods of InetAddress Class [1-6]

- The `InetAddress` class is used to create an IP address.
- An IP address is either a 32 bit or 128 bit unsigned number used by IP which is a low level protocol.
- The `InetAddress` class is used for encapsulating the numerical IP address and the domain name for that address.
- The class offers numerous methods for dealing with hostnames and IP address.
- One of the factory methods are used to create an instance of `InetAddress` class as there are no visible constructors in this class.
- Factory methods are static methods in a class that returns an instance of that class.
- To create an instance of `InetAddress` class, the three factory methods are: `getLocalHost()`, `getByName()`, and `getAllByName()`.

Methods of InetAddress Class [2-6]

- The methods of the InetAddress class are:
  - `public String getHostAddress()`: This method returns the raw IP address in textual format.
    - Code Snippet shows how to retrieve the string representation of an IP address.

Code Snippet

```
InetAddress address = null;
// Retrieves the local host address try {
   address = InetAddress.getLocalHost();
} catch (UnknownHostException ex)
  { ex.printStackTrace(); }
// Retrieves the string representation of the local host address
     String strAddress = address.getHostAddress();
```

Methods of InetAddress Class [3-6]

- `public String getHostName()`: This method returns the host name of the IP address.
  - Code Snippet shows how to retrieve the textual name of host.

Code Snippet

```
InetAddress address = null;
// Retrieves the local host address try {
    address = InetAddress.getLocalHost();
catch (UnknownHostException ex)
   { ex.printStackTrace();  }
// Retrieves the textual name of the local host
   String strAddress = address.getHostName();
```

- `public static InetAddress getLocalHost()` : This method returns an IP address of the local host.
  - Code Snippet shows how to retrieve the `InetAddress` of local host.

Code Snippet

```
try {
address = InetAddress.getLocalHost();
     } catch (UnknownHostException ex) { ex.printStackTrace();  }
```

Methods of InetAddress Class [4-6]

- `public static InetAddress getByName(String host)` : This method returns an IP address of the specified host name.
  - Code Snippet shows how to retrieve the IP address of a given host.

Code Snippet

```
InetAddress address = null;
   try {
            address = InetAddress.getByName("www.yahoo.com");
        } catch (UnknownHostException ex) { ex.printStackTrace();   }
```

Methods of InetAddress Class [5-6]

- `static InetAddress [] getAllByName (String name) throws UnknownHostException`: The `getAllByName()` method returns an array of `InetAddress`.
  - Code Snippet shows the use of the `getAllByName()` method.

Code Snippet

```
try
{
    InetAddress address = InetAddress.getByName (args[0]);
     System.out.println ("Address: " + args[0] + " = " +
address);
      InetAddress[] add = InetAddress.getAllByName(args[0]);
      for(int i = 0; i < add.length; i++) {
      System.out.println ("Name of = " + add[i] );
      }
. . .}
} catch(UnknownHostException e) {
   System.out.println ("Unable to translate the address." + e);
}
```

Methods of InetAddress Class [6-6]

- `getContentLength()`: The `getContentLength()` method is used to retrieve the content length of the resource that is referenced by the URL connection. The method will return -1 if the content length is not known.
  - Code Snippet shows how to retrieve the content length of the URL resource.

Code Snippet

```
. . .
int contentLength; try {
// Creates and instantiates URL object
URL url = new URL("http://java.sun.com/docs/books/tutorial/index.
html");
URLConnection urlCon = url.openConnection();
// Retrieve the content length of the URL resource contentLength =
urlCon.getContentLength();
System.out.println("Content Length: " + contentLength);
} catch(MalformedURLException ex) { System.out.println("Exception :
" + ex.getMessage());
}
```

URLConnection Class

- `URLConnection` is an abstract class that represents the communication link between an application and a URL.
- Instances of this class can be used to read from and to write to the resource referenced by the URL.
- You cannot instantiate a `URLConnection` object directly; a valid URL instance is required to create an instance of `URLConnection` from the URL.
- An instance of `URLConnection` class can be used to:

1. Inspect and set the properties of connection
2. Obtain information such as content length, content type, and so on from the URL
3. Move data through the connection by obtaining an instance of the input and output streams

Methods of URLConnection Class [1-6]

- `openConnection()`
  - The `URL` class has a method `openConnection()` to create an `URLConnection` instance.
  - Code Snippet shows how to use the `openConnection()` method of the `URL` class.

Code Snippet

```
URL url = null;
try {
        url = new URL("http://www.yahoo.com/");
        URLConnection urlCon = url.openConnection();
    } catch (MalformedURLException ex) {
        ex.printStackTrace();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
```

Methods of URLConnection Class [2-6]

- `InputStream()`: Retrieve the `InputStream` from the connection.
  - Code Snippet shows how to retrieve the `InputStream` from an instance.

Code Snippet

```
try {
        InputStream in = urlCon.getInputStream();

        BufferedReader reader = new BufferedReader(new
InputStreamReader(in));
} catch (IOException ex) {
        ex.printStackTrace();
}
```

Methods of URLConnection Class [3-6]

- `OutputStream()`: Retrieve the `InputStream` from the connection.
  - Code Snippet shows how to retrieve the `OutputStream` from an instance.

Code Snippet

```
try {
        OutputStream out = urlCon.getOutputStream();
        BufferedWriter writer = new BufferedWriter(new
OutputStreamWrite r(out));
} catch (IOException ex) {
        ex.printStackTrace();
}
```

Methods of URLConnection Class [4-6]

- `getContentType()`: The `getContentType()` method is used to retrieve the content type of the resource that the URL references. The method will return null if it is not known.
  - Code Snippet shows how to retrieve the content type of the URL resource.

Code Snippet

```
String strContentType; try {
// Creates and instantiates URL object
    URL url = new
URL("http://java.sun.com/docs/books/tutorial/index. html");
URLConnection urlCon = url.openConnection();
// Retrieve the content type of the URL
    strContentType =     urlCon.getContentType();
    System.out.println("Content Type : " + strContentType);
} catch(MalformedURLException ex) {
   System.out.println("Exception : " + ex.getMessage());
}
```

Methods of URLConnection Class [5-6]

- `getConnectTimeout()`: The `getConnectTimeout()` method is used to retrieve the settings of the connection timeout of the `URL` in milliseconds.
  - Code Snippet shows how to retrieve the connection timeout of the URL.

Code Snippet

```
. . .
int contentLength;
 try {
// Creates and instantiates URL object
URL url = new URL("http://java.sun.com/docs/books/tutorial/index.
html");
URLConnection urlCon = url.openConnection();
// Retrieve the content length of the URL resource
contentLength = urlCon.getContentLength();
System.out.println("Content Length: " + contentLength);
} catch(MalformedURLException ex) { System.out.println("Exception :
" + ex.getMessage());
}
. . .
```

Methods of URLConnection Class [6-6]

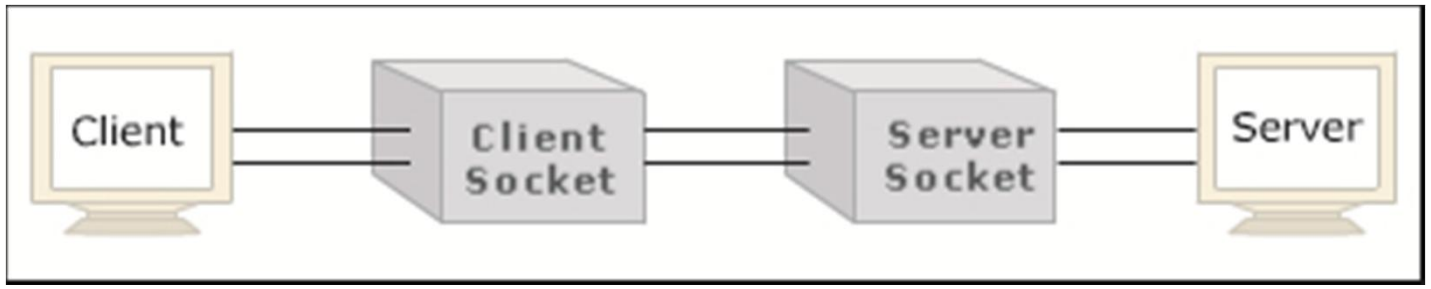
- `setConnectTimeout()`: The `setConnectTimeout()` method is used to set a specified timeout value in milliseconds. The timeout value is used when opening a communication link to the resource referenced by the `URLConnection`.
  - Code Snippet shows how to set the connection timeout of the URL.

Code Snippet

```
int timeOut;
try {
// Creates and instantiates URL object
URL url = new URL("http://java.sun.com/docs/books/tutorial/index. html");
URLConnection urlCon = url.openConnection();
// Assign a timeout value of 30 seconds
    timeOut = 30000;
// Sets the connection timeout of the URL
    urlCon.setConnectTimeout(timeOut);
    System.out.println("Connection timeout set to : " + timeOut+"
milliseconds");
} catch(MalformedURLException ex)
   { System.out.println("Exception : " + ex.getMessage());
}
. . .
```

Socket Class

- A `socket` is one end-point of a two-way communication link between two programs running on the network.
- A server application runs on a particular machine and has a socket bound to a specific port number.
- The server then waits, listening to a client socket to make a connection request.
- On the client-side, the client application needs to know the host name of the machine on which the server application is running and the port number on which the server is listening.
- To make a connection request, the client opens a socket with the server on the server's machine and port.
- Figure displays a socket.



Creating a Socket [1-2]

- The `java.net.Socket` class is used to represent the connection between a client program and a server program.
- The `Socket` class represents the client side of the connection.
- The `Socket` class has several constructors to create a socket.
- Following is most often used constructor to create a socket:

```
Socket(String host, int port)
```

- Syntax:

```
Socket(String host, int port) throws
UnknownHostException, IOException
```

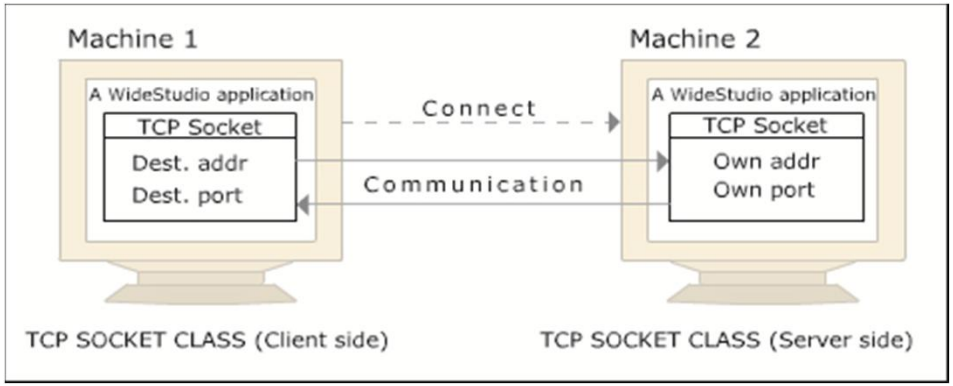
Creating a Socket [2-2]

- `Socket(String host, int port);`

Code Snippet

```
Socket socket;
String host;
final int PORT = 5000;
host = "localhost";
try {
          socket = new Socket(host, PORT);
} catch (UnknownHostException ex) {
          ex.printStackTrace();
} catch (IOException ex) {
          ex.printStackTrace();
}
```

- Figure displays the socket creation.



Reading from and Writing to a Socket [1-6]

- Once a `Socket` instance is created successfully, perform the following steps to use the socket:
  - Retrieve the `InputStream` from the socket
  - Retrieve the `OutputStream` from the socket
  - Read from the `InputStream`
  - Write to the `OutputStream`

Reading from and Writing to a Socket [2-6]

- `getInputStream()`
  - The `Socket` class has a method `getInputStream()` which returns an `InputStream` object associated with a host's socket.
  - This object is normally wrapped in a `BufferedReader` object, for directly reading data from the socket.
  - Code Snippet shows how to retrieve the `InputStream` of a socket.

Code Snippet

```
Socket socket;
String host; final int PORT = 5000;
BufferedReader reader;
try {
     host = "localhost";
     socket = new Socket(host, PORT); // Creates a Socket object
     // Retrieves the input stream of the socket
     reader = new BufferedReader(new
     InputStreamReader(socket.getInputStream()));
} catch (UnknownHostException ex) {
          ex.printStackTrace();
} catch (IOException ex) {
          ex.printStackTrace(); }
```

Reading from and Writing to a Socket [3-6]

- `getOutputStream()`
  - The `Socket` class has a method `getOutputStream()` which returns an `OutputStream` object associated with a host's socket.
  - This object is normally wrapped in a `PrintStream` or `PrintWriter` object for writing data to the socket.
  - Code Snippet shows how to retrieve the `OutputStream` of the socket.

Code Snippet

```
Socket socket;
String host;
final int PORT = 5000;
PrintWriter writer;
host = "localhost";
try {
     socket = new Socket(host, PORT); // Creates a Socket object
     // Retrieves the output stream of the socket
     writer = new PrintWriter(socket.getOutputStream());
} catch (UnknownHostException ex) {
          ex.printStackTrace();
} catch (IOException ex) {
          ex.printStackTrace();}
```

Reading from and Writing to a Socket [4-6]

- `readLine()`
  - Once a `BufferedReader` object is created, you use the `readLine()` method to read from the socket until a null is encountered which indicates that all data sent from the host has been read.
  - Code Snippet shows how to read from a socket.

Code Snippet

```
Socket socket;
String host = "localhost";
String strInfo = "";
final int PORT = 5000;
BufferedReader reader;
try {
    // Creates a Socket object
    socket = new Socket(host, PORT);
    // Retrieves the input stream of the socket
    reader = new BufferedReader(new InputStreamReader(socket.
    getInputStream()));
    // Reads from a socket
    while((strInfo = reader.readLine()) != null) {
```

Reading from and Writing to a Socket [5-6]

```
    reader = new BufferedReader(new InputStreamReader(socket.
    getInputStream()));
    // Reads from a socket
    while((strInfo = reader.readLine()) != null)
    {
        System.out.println(strInfo);
    }
}
catch (UnknownHostException ex) {
        ex.printStackTrace();
}
catch (IOException ex) {
        ex.printStackTrace();
}
```

Reading from and Writing to a Socket [6-6]

- `pritnln()`
  - Once a `PrintStream` or `PrintWriter` object is created, you use the `println()` method to write data to the host.
  - Code Snippet shows how to write to a socket.

Code Snippet

```
Socket socket;
String host = "localhost";
String strInfo = "";
final int PORT = 5000;
PrintWriter writer = null;
try {
    socket = new Socket(host, PORT); // Creates a Socket object
    // Retrieves the output stream of the socket
    writer = new PrintWriter(socket.getOutputStream());
    writer.println("Hello World"); // Writes to a socket
} catch (UnknownHostException ex) {
        ex.printStackTrace();
} catch (IOException ex) {
        ex.printStackTrace();
}
```

ServerSocket Class [1-2]

- The `ServerSocket` class is used to represent the server side of the two-way communication.
- The `ServerSocket` has to bind to a specific port which should be free and available.
- If the port is being used by any other application, an exception is thrown.
- The ServerSocket class has several constructors to create a server socket.
- However, the most commonly used constructor is:

```
public ServerSocket(int port) throws IOException
```

- This constructor takes a port number as an argument.
- The specified port should be available to use, failing which an exception is thrown.
- The `ServerSocket` class binds to this specified port, and if successful, can later listen for client requests.

ServerSocket Class [2-2]

- Code Snippet shows how to bind server socket to the port number 5000.

Code Snippet

```
ServerSocket server;
final int PORT = 5000;
try {
    server = new ServerSocket(PORT);
}
catch (IOException ex) {
    System.out.println("Failed to bind");
}
```

Creating ServerSocket Instance [1-2]

- The `ServerSocket` class has a method `accept()` for listening to client request.
- Once a `ServerSocket` instance is created, you can invoke the `accept()` method to listen for client request.
- The `accept()` method is a blocking method that is once invoked it will wait till a client requests for a connection.
- When a client requests for a connection, the `accept()` method creates a socket object of class `Socket` and returns it.
- This returned object represents a proxy of a client.
- To communicate with the client, you retrieve the `InputStream` and `OutputStream` of this proxy socket object.

Creating ServerSocket Instance [2-2]

- Code Snippet shows how to use an instance of a `ServerSocket` class to retrieve a proxy of the client and its associated input and output streams.

Code Snippet

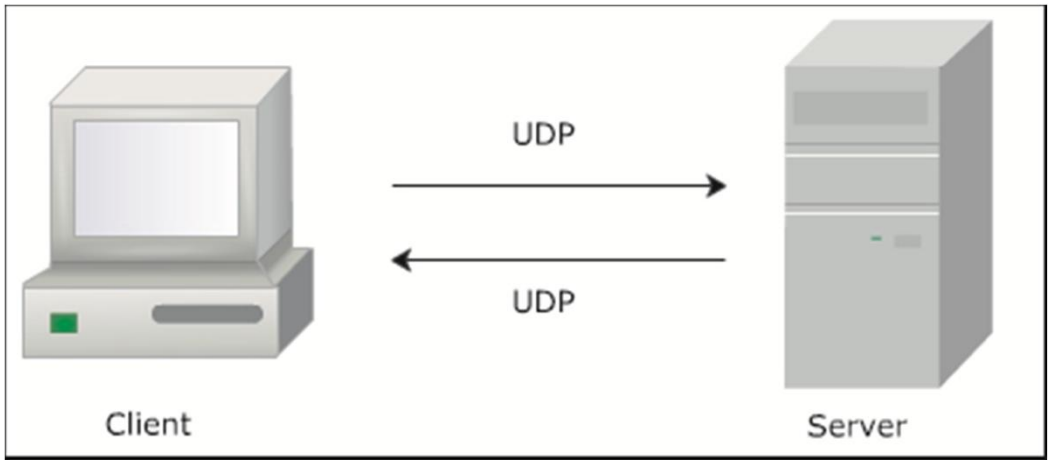
```
ServerSocket server;
Socket client; // Proxy of a client
final int PORT = 5000;
PrintStream ps = null;
BufferedReader reader = null;
try {
     // Creates a ServerSocket object
     server = new ServerSocket(PORT);
     // Retrieve a proxy of the client
     client = server.accept();
     // Retrieves the input stream of the client
     socket reader = new BufferedReader(new InputStreamReader(client.
          getInputStream()));
     // Retrieves the output stream of the socket
     ps = new PrintStream(client.getOutputStream());
} catch (IOException ex) {
     System.out.println("Failed to bind");
}
```

Methods of ServerSocket Class

- `getInetAddress()`: The `getInetAddress()` method returns the local address of the server socket to which it is bound. The local address is returned as an Internet Protocol (IP) address.
- `public void close() throws IOException`: The `close()` method is used to close the server socket.
- `public boolean isBound()`: The `isBound()` method is used to retrieve the binding state of the server socket.
- `public int getLocalPort()`: The `getLocalPort()` method is used to retrieve the local port on which the server socket is listening.

Datagram Class

- A datagram is an independent, self-contained message sent over the network whose arrival time and order of content are not guaranteed.
- `ServerSocket` and `Socket` are used to establish a reliable communication that is data transmitted from one end will reach the other end.
- Datagram packets are used to employ a connectionless packet delivery service.
- Each message is routed based solely on information contained within that packet.
- Multiple packets sent from one machine to another might be routed differently and might arrive in any order.
- Packets delivered by datagrams are not guaranteed.
- Figure displays a datagram.



Creating a Datagram Packet [1-2]

- The `java.net` package provides the `DatagramPacket` class to create a datagram packet.
- The `DatagramPacket` has several constructors to create datagram packet.
- The most commonly used constructor of `DatagramPacket` are:
  - `public DatagramPacket(byte[] buf, int length)`
    - Code Snippet shows how to create a datagram packet with a buffer size of 256 bytes to receive a packet of data.

Code Snippet

```
// Creates buffer of size 256
     bytes byte[] buf = new byte[256];
// Creates an object of DatagramPacket of size same as buffer
    DatagramPacket packet = new DatagramPacket(buf, buf.length);
```

Creating a Datagram Packet [2-2]

- `public DatagramPacket(byte[] buf, int length, InetAddress address, int port)`
  - Code Snippet shows how to create a datagram packet to send a packet of data to a host named Joe1.

Code Snippet

```
// Creates buffer of size 256 bytes
   byte[] buf = new byte[256];
   final int PORT = 5000;
   String host = "Joe1";
   InetAddress address = null;
// Get the IP address of the host
try {
      address = InetAddress.getByName(host);
    } catch (UnknownHostException ex) {
    ex.printStackTrace();
}
// Creates a datagram packet to send to the host DatagramPacket
packet = new DatagramPacket(buf, buf.length, address, PORT);
```

Creating Datagram Socket [1-2]

- `DatagramSocket` class represents a socket for sending and receiving datagram packets.
- Each packet of data sent or received on a datagram socket is individually addressed and routed.
- The most commonly used constructor of `DatagramSocket` are:
  - `public DatagramSocket(int port) throws SocketException`
    - Code Snippet shows how to create a datagram socket and bind it to the port number 5000.

Code Snippet

```
final int PORT = 5000;
DatagramSocket socket = null;
try {
     socket = new DatagramSocket(PORT);
} catch(SocketException ex) {
     System.out.println("Could not create a datagram
     socket.");
}
```

Creating Datagram Socket [2-2]

- `public DatagramSocket(int port, InetAddress iaddr) throws SocketException`
  - Code Snippet shows how to create a datagram socket listening on port number 5000 on a server named Yasmin1.

Code Snippet

```
final int PORT = 5000;
DatagramSocket socket = null;
InetAddress address = null;
try {
        address = InetAddress.getByName("Yasmin1");
    // Create the datagram socket
        socket = new DatagramSocket(PORT,address);
} catch(UnknownHostException ex) {
     ex.printStackTrace();
} catch(SocketException ex) {
     System.out.println("Could not create a datagram socket.");
}
```

Methods of DatagramSocket Class [1-3]

- `public void send(DatagramPacket p)`: The `send()` method is used to send a datagram packet using a `DatagramSocket` instance.
  - Code Snippet shows how to send a datagram packet using a datagram socket.

Code Snippet

```
String host = "Yasmin1";
byte[] buf = new byte[256];
try {
     InetAddress address = InetAddress.getByName(host);
     DatagramPacket packet = new DatagramPacket(buf, buf.length,
          address, 4445);
     DatagramSocket socket = new DatagramSocket();
     socket.send(packet); // Send the packet
} catch (SocketException ex) {
     ex.printStackTrace();
} catch (UnknownHostException ex) {
     ex.printStackTrace();
} catch (IOException ex) {
     ex.printStackTrace();
}
```

Methods of DatagramSocket Class [2-3]

- `public void receive(DatagramPacket p)`: The `receive()` method is used to receive a datagram packet using a `DatagramSocket` instance.
  - Code Snippet shows how to receive a datagram packet using a datagram socket.

Code Snippet

```
// Create a buffer to store data
byte[] buf = new byte[256];
// Create a datagram socket to receive the packet
DatagramSocket socket = new DatagramSocket();
// Create a datagram packet to store data sent by host
DatagramPacket packet = new DatagramPacket(buf,
    buf.length);
// Receive the packet
socket.receive(packet);
```

Methods of DatagramSocket Class [3-3]

- `public InetAddress getAddress()`: The method returns the IP address of the machine to which the datagram packet is being sent or received.
- `public int getPort()`: The method returns the port number of the remote host to which the datagram packet is being sent or received.
- `public byte[] getData()`: The method returns the data buffer. The data received or the data to be sent starts from the offset in the buffer, and continues till the specified length. By default the offset is zero.
- `public void setData(byte[] data)`: The method sets the data buffer for the packet. The offset is set to zero, and the length of data is set to the length of the data array.
- `public void (int port)`: The method sets the port number on the remote host to which this datagram packet is being sent.

Broadcasting to Multiple Recipients [1-3]

- A MulticastSocket is a `java.net` utility that is used on the client-side to listen for messages that the server broadcasts to multiple clients.
- A client may or may not send any message to the server.
- Code Snippet to send messages from a server to multiple clients.

Code Snippet

```
import java.io.*;
public class MulticastServer {
 public static void main(String[] args) throws IOException
 {
          new MulticastServerThread().start();
 }
}
```

Broadcasting to Multiple Recipients [2-3]

- Code Snippet shows the implementation of MulticastServerThread in the `run()` method.

Code Snippet

```
public void run() {
  while (moreQuotes) {
         try {
               byte[] buf = new byte[256];
            // don't wait for request...just send a quote
              String dString = null;
                 if (in == null)
                    dString = new Date().toString();
                    else
                    dString = getNextQuote();
                    buf = dString.getBytes();
                    InetAddress group = InetAddress.getByName("204.0.115.0");
              DatagramPacket packet;
                    packet = new DatagramPacket(buf, buf.length, group, 4346);
                    socket.send(packet);
                    try {
                         sleep((long)Math.random() * FIVE_SECONDS);
                       } catch (InterruptedException e) { }
                       } catch (IOException e) { e.printStackTrace();
                                                 moreQuotes = false;
                     }
  }
```

Broadcasting to Multiple Recipients [3-3]

- Code Snippet shows how the client program receives Datagram packets.

Code Snippet

```
MulticastSocket socket = new MulticastSocket(4346);
InetAddress group = InetAddress.getByName("204.0.115.0");
socket.joinGroup(group);
DatagramPacket packet;
for (int i = 0; i < 10; i++) {
byte[] buf = new byte[256];
packet = new DatagramPacket(buf, buf.length);
socket.receive(packet);
String received = new String(packet.getData());
System.out.println("Quote of the Moment: " + received);
}socket.leaveGroup(group);
socket.close();
```

Multicast Socket [1-2]

- MulticastSocket has some additional capabilities that let it form groups of other multicast hosts on the Internet.
- A multicast group is specified by a class D IP address and by a standard UDP port number.
  - Class D IP addresses are in the range 224.0.0.0 to 239.255.255.255. The address 224.0.0.0 is reserved and should not be used.
- Code Snippet shows how to join a multicast group by creating a `MulticastSockets` with the desired port number and invoking the `joinGroup()` method.

Code Snippet

```
// join a Multicast group and send the group salutations
String msg = "Hello";
InetAddress group = InetAddress.getByName("228.5.6.7");
MulticastSocket s = new MulticastSocket(6789);
s.joinGroup(group);
DatagramPacket hi = new DatagramPacket(msg.getBytes(), msg.length(), group,
6789);
s.send(hi);
// get their responses!
byte[] buf = new byte[1000];
DatagramPacket recv = new DatagramPacket(buf, buf.length);
s.receive(recv);
...
// leave the group...
s.leaveGroup(group);
```

Multicast Socket [2-2]

- Some of the methods used by `MulticastSockets` are as follows:
  - `getInterface()`
  - `getLoopbackMode()`
  - `getNetworkInterface()`
  - `getTimeToLive()`
  - `joiningGroup()`
  - `Send(Datagram Packet, byte)`
  - `setInterface(InetAddress)`
  - `setNetworkInterface(NetworkInterface netIf)`
  - `setTimeToLive()`

Network Interface

- The point of interconnection between a computer and a private or public network is termed as a network interface.
- It can be either a hardware device such as a Network Interface Card (NIC) or a software implementation such as IPv6.
- The `java.net.NetworkInterface` class represents both types of interfaces.

Use a Socket with a Network Interface

- To send or receive data from server, network interfaces need sockets.
- These sockets are created using the `soc.connect()` method.
- Code Snippet shows how to create a socket to send data to a server.

Code Snippet

```
Socket soc = new java.net.Socket();
soc.connect(new InetSocketAddress(address, port));
```

- Code Snippet shows how to create a socket bound to an address and sends data through the network.

Code Snippet

```
NetworkInterface nif = NetworkInterface.getByName("bge0");
Enumeration<InetAddress> nifAddresses =
nif.getInetAddresses();
Socket soc = new java.net.Socket();
soc.bind(new InetSocketAddress(nifAddresses.nextElement(),
0));
soc.connect(new InetSocketAddress(address, port));
```

NetworkInterface Class

- The `NetworkInterface` class has no public constructor.
- Thus, you cannot create a new instance of this class with the `new` operator.
- Code Snippet lists all the network interfaces and their addresses on a machine.

Code Snippet

```
. . .
public class ListNets {
   public static void main(String args[]) throws SocketException {
     Enumeration<NetworkInterface> nets =
     NetworkInterface.getNetworkInterfaces();
     for (NetworkInterface netint : Collections.list(nets))
         displayInterfaceInformation(netint);
 }
static void displayInterfaceInformation(NetworkInterface netint)
 throws SocketException {
     out.printf("Display name: %s\n", netint.getDisplayName());
     out.printf("Name: %s\n", netint.getName());
     Enumeration<InetAddress> inetAddresses = netint.getInetAddresses();
    for (InetAddress inetAddress : Collections.list(inetAddresses))
    {
          out.printf("InetAddress: %s\n", inetAddress);
    }
          out.printf("\n");
}
```

InterfaceAddress Class [1-2]

- Represents a Network Interface address.
- **When the address is IP4**
  - Is an IP address, a subnet mask, and a broadcast address.
- **When the address is IP6**
  - Is an IP address and a network prefix length in the case of IPv6 address.
- Code Snippet shows a sample using NetworkInterface address.

Code Snippet

```
import java.net.InterfaceAddress;
import java.net.NetworkInterface;
import java.net.SocketException;
import java.util.Enumeration;
import java.util.List;
public class InterfaceAddressTest {
  public static void main(String[] args) throws SocketException {
    Enumeration<NetworkInterface> en =
NetworkInterface.getNetworkInterfaces();
```

InterfaceAddress Class [2-2]

```
  while (en.hasMoreElements()) {
    NetworkInterface ni = en.nextElement();
    List<InterfaceAddress> inAdd = ni.getInterfaceAddresses();
    for (InterfaceAddress ia : inAdd) {
     //returns Inet Address
        System.out.println(ia.getAddress());
      //returns Inet Address for the broadcast address
        System.out.println(ia.getBroadcast());
     //returns network prefix length
        System.out.println(ia.getNetworkPrefixLength());
    }
   }
 }
}
```

Cookie

- Java Web applications can use cookies to store information on the client machine.
- It is usually stored in a browser's cache.
- A cookie which holds data for a single Web session, that is, until you close the browser is termed as a short term cookie
- A cookie which holds data for a week or a year is termed as a long term cookie.
- The `CookieManager` class is the main entry point for cookie management.
- An instance of the `CookieManager` class is created and a `CookiePolicy` is set.
- Cookies are retrieved from the underlying CookieStore by using the `getCookies` method.

HTTP State Management

- The HTTP state management mechanism specifies a way to create a stateful session with HTTP requests and responses.
- A session is created for exchange of information. Cookies are used to create and maintain the state information of the session.
- Code Snippet shows how to create and set a system-wide `CookieManager`.

Code Snippet

```
java.net.CookieManager cm = new java.net.CookieManager();
java.net.CookieHandler.setDefault(cm);
```

- `CookieManager` uses the default policy `CookiePolicy.ACCEPT_ORIGINAL_SERVER`.
- `CookieManager` forces path match rule when getting the cookies from the cookie store.
- `CookieManager` provides the framework for handling cookies and a default implementation for `CookieStore`.

Summary

- TCP is a connection-based protocol that provides a reliable flow of data between two computers.
- On the Internet, computers communicate with each other using either the TCP or the UDP.
- URL is an acronym for Uniform Resource Locator. It is a reference or an address to a resource on the Internet. The resource can be a HTML page, an image, or simply any file.
- The URLConnection is an abstract class that represents the communication link between an application and a URL.
- A socket is one end-point of a two-way communication link between two programs running on the network.
- The ServerSocket class is used to represent the server side of the two-way communication. The ServerSocket class has to bind to a specific port which should be free and available.
- A datagram is an independent, self-contained message sent over the network whose arrival time and order of content are not guaranteed.