Networking

With networking, it is possible to send or receive some data from a remote system. A group of interconnected **autonomous systems** is called a **network**. Every system on the network has a specific **32-bit IP address** with which we can identify the system. Data transmitted over the Internet can be identified by the **computer** and the **port** for which it is destined. Ports are identified by a **16-bit number**.

Port numbers range from **0 to 65,535**. The port numbers ranging from **0 - 1023** are restricted; they **are reserved** for use by well-known services such as HTTP and FTP and other system services. These ports are called **well-known ports**. Your applications should not attempt to bind to them.

URL is an acronym for **Uniform Resource Locator** and is a reference to a resource on the Internet. General form of a URL is as follows.

Protocol://hostname:port-number/File-name#Reference

e.g.

http://www.java.sun.com/index.html
http://localhost:8080/toc.html#top

Protocol	The type of protocol being used like HTTP, FTP, ...
Host Name	The name of the machine on which the resource lives.
Port Number	The port number to which to connect (optional).
Filename	The pathname to the file on the host machine.
Reference	A reference to a named anchor within a resource that usually identifies a specific location within a file (optional).

The package **java.net** contains classes and interfaces that can perform network related operations like getting hostnames, IP addresses, creating Sockets... Some of them are as follows:

InetAddress
URL
ServerSocket
Socket
DatagramSocket
DatagramPacket

The **URL** class provides several methods that let you query URL objects. You can get the protocol, authority, host name, port number, path, query, filename, and reference from a URL using these accessor methods:

getProtocol()	Returns the protocol identifier component of the URL.
getAuthority()	Returns the authority component of the URL.
getHost()	Returns the host name component of the URL.
getPort()	Returns the port number component of the URL. The getPort() method returns an integer that is the port number. If the port is not set, getPort() returns -1.
getPath()	Returns the path component of this URL.
getQuery()	Returns the query component of this URL.
getFile()	Returns the filename component of the URL. The getFile() method returns the same as getPath, plus the concatenation of the value of getQuery(), if any.
getRef()	Returns the reference component of the URL.

Methods of InetAddress Class

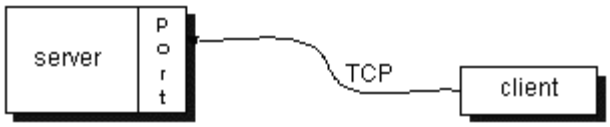
getLocalHost()	Returns the Local Host Name with IP Address
getHostName()	Returns the Host Name
getHostAddress()	Returns the IP Address of the Host.
hashCode()	Returns the hash code of the Host.
isMulticastAddress()	Returns true, if it is multicast address.
isLoopbackAddress()	Returns true, if it is loop back address

In client-server applications, **Server provides** some **service**, such as processing database queries or sending out current stock prices. The **Client uses** the **service** provided by the server, either displaying database query results to the user or making stock purchase recommendations to an investor.

Computers on the network can communicate in two ways. They are **Connection Oriented (or) Stream Based** communication and **Connectionless (or) Packet Based** communication.

In **connection-oriented** communication such as **TCP**, a server application binds a socket to a specific port number. This has the effect of registering the server with the system to receive all data destined for that port. A client can then talk with the server at the server's port.

A **socket** is one **end-point** of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program. A program can read from or write to a socket as simply as reading from a file or writing to a file.



TCP provides a **point-to-point** channel for applications that require **reliable communications**. The Hypertext Transfer Protocol (**HTTP**), File Transfer Protocol (**FTP**), and **Telnet** are all examples of applications that require a reliable communication channel. In this communication, data must be received in the order in which it was sent.

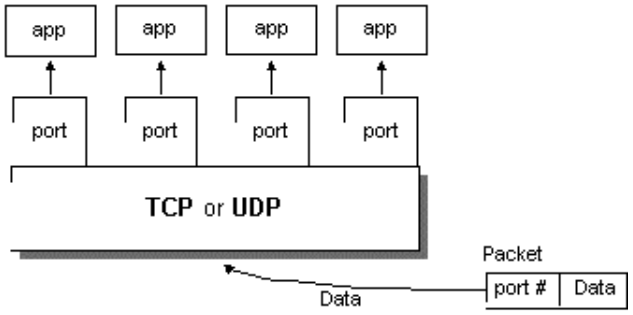
Connection oriented transmission is like the **telephone system**, you will be given a connection to the telephone of the person with whom you wish to communicate. The connection is maintained for the duration of your phone call, even when you are not talking.

The classes **Socket** and **ServerSocket** implements TCP based client side of the connection and the server side of the connection, respectively.

ServerSocket (int Port-number, int no-of-connections);
Socket (String Sever-name, int Port-number);

ServerSocket method **accept()** waits indefinitely for a connection from a client and returns a Socket object when a connection is established. **Socket** methods **getOutputStream()** and **getInputStream()** get references to a Socket's OutputStream and InputStream respectively.

In **connection-less** communication such as **UDP**, a **datagram packet** contains the port number of its destination and UDP routes the packet to the appropriate application. A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.



The UDP protocol provides for communication that is not guaranteed between two applications on the network. UDP is not connection-based like TCP. Rather, it sends independent packets of data, called **datagrams**, from one application to another. The order of delivery is not important and is not guaranteed, and each message is independent of any other.

Connection-less transmission with datagrams is similar to mail carried via the **postal service**. A large message that will not fit in a single envelope can be broken into separate message pieces. These are numbered and placed in separate envelopes. All these letters are then mailed. They could arrive in order, out of order or not at all. It is the duty of the receiver to arrange the envelopes in order with the help of **sequence numbers**.

Multi Threaded Server

A thread, sometimes called a light weight process, is a basic unit of CPU utilization. A traditional (or heavy weight) process has a single thread of control. If the process has multiple threads of control, it can do more than one task at a time. A web browser might have one thread display images or text while another thread retrieves data from the network. A word processor may have a thread for displaying graphics, another thread for reading keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.

In certain situations a single application may be required to perform several similar tasks. For example, a web server accepts client requests for web pages, images, sound. A busy web server may have several clients concurrently accessing it. If the web server ran as a traditional single-threaded process, it would be able to service only one client at a time.

In a multi threaded server, a Server program can provide service to many clients in terms of multiple threads. It should extend the Thread class. As each client request is received by the server, new instance of a thread will be created to process the request made by the client. The client then reads the data from the server's socket.

Methods of ServerSocket class:

ServerSocket(int port)

InetAddress getInetAddress()

int getLocalPort()

void close()

Socket accept()

Methods of Socket class:

Socket(String host, int port)

InetAddress getInetAddress()

InetAddress getLocalAddress()

int getPort()

InputStream getInputStream()

OutputStream getOutputStream()

void close()

Methods of DatagramSocket class:

DatagramSocket(int port, InetAddress)

InetAddress getInetAddress()

InetAddress getLocalAddress()

int getPort()

void send(DatagramPacket dp)

void receive(DatagramPacket dp)

void close()

Methods of Datagrampacket class:

DatagramPacket(byte data[ ], int length, InetAddress addr, int port)

DatagramPacket(byte data[ ], int length)

InetAddress getInetAddress()

byte[ ] getData()

int getPort()

void close()