Chapter 2 networking

Q Write a short note on various elements connected to network

• ***Network cards*** are hardware devices added to a computer to allow it to talk to a network. The most common network card in use today is the Ethernet card. Network cards usually connect to a network cable, which is the link to the network and the medium through which data is transmitted. However, other media exist, such as dial-up connections through a phone line, and wireless links.

• ***Routers*** are machines that act as switches. These machines direct packets of data to the next "hop" in their journey across a network.

• ***Hubs*** provide connections that allow multiple computers to access a network (for example, allowing two desktop machines to access a local area network).

• ***Gateways*** connect one network to another—for example, a local area network to the Internet. While routers and gateways are similar, a router does not have to bridge multiple networks. In some cases, routers are also gateways.

Q Explain the concept of addressing

Each node in a network is typically represented by an address, just as a street name and number,town or city, and zip code identifies individual homes and offices. The manufacturer of the network interface card (NIC) installed in such devices is responsible for ensuring that no two card addresses are alike, and chooses a suitable addressing scheme. Each card will have this address stored permanently, so that it remains fixed—it cannot be manually assigned or modified, although some operating systems will allow these addresses to be faked in the event of an accidental conflict with another card's address.

Because of the wide variety of NICs, many addressing schemes are used. For example, Ethernet network cards are assigned a unique 48-bit number to distinguish one card from another. Usually, a numerical number is assigned to each card, and manufacturers are allocated batches of numbers. The physical address is referred to by many names (some of which are specific to a certain type of card, while others are general terms), including:

• Hardware address
• Ethernet address
• Media Access Control (MAC) address
• NIC address

Q Explain Data Transmission Using Packets

Sending individual bits of data from node to node is not very cost effective, as a fair bit of overhead is involved in relaying the necessary address information every time a byte of data is transmitted. Most networks, instead, group data into packets. Packets consist of a header and data segment.. The header contains addressing information (such as the sender and the recipient), checksums to ensure that a packet has not been corrupted, as well as other useful information that is needed for transmission across the network. The data segment contains sequences of bytes, comprising the actual data being sent from one node to another. Since the header information is needed only for transmission, applications are interested only in the data segment. Ideally, as much data as possible would be combined into a packet, in order to minimize the overhead of the headers.

When a node on the network is ready to transmit a packet, a direct connection to the destination node is usually not available. Instead, intermediary nodes carry packets from one location to another, and this process is repeated indefinitely until the packet reaches its destination. Due to network conditions (such as congestion or network failures), packets may take arbitrary routes, and sometimes they may be lost in transit or arrive out of sequence.

Q Explain various Layers of Network

Layer 1—Physical Layer

The physical layer is networking communication at its most basic level. The physical layer governs the very lowest form of communication between net-work nodes. At this level, networking hardware, such as cards and cables, transmit a sequence of bits between two nodes. Java programmers do not work at this level—it is the domain of hardware driver developers and electrical engineers.

Layer 2—Data Link Layer

The data link layer is responsible for providing a more reliable transfer of data, and for grouping data together into frames. Frames are similar to data packets, but are blocks of data specific to a single type of hardware architecture (whereas data packets are used at a higher level and can move from one type of network to another). Frames have checksums to detect errors in transmission, and typically a "start" and "end" marker to alert hardware to the division between one frame and another. Sequences of frames are transmitted between network nodes, and if a frame is corrupted it will be discarded.

Layer 3—Network Layer

Moving up from the data link layer, which sends frames over a network, we reach the network layer. The network layer deals with data packets, rather than frames, and introduces several important concepts, such as the network address and routing. Packets are sent across the network, and in the case of the Internet, all around the world. Unless traveling to a node in an adjacent network where there is only one choice, these packets will often take alternative routes (the route is determined by routers).Network programmers are rarely required to write software services for this layer.

4 Layer 4—Transport Layer

The fourth layer, the transport layer, is concerned with controlling how data is transmitted. This layer deals with issues such as automatic error detection and correction, and flow control (limiting the amount of data sent to prevent overload).

Layer 5—Session Layer

The purpose of the session layer is to facilitate application-to-application data exchange, and the establishment and termination of communication sessions. Session management involves a variety of tasks, including establishing a session, synchronizing a session, and reestablishing a session that has been abruptly terminated.

Layer 6—Presentation Layer

The sixth layer deals with data representation and data conversion. Different machines use different types of data representation (an integer might be represented by 8 bits on one system and 16 bits on another). Some protocols may want to compress data, or encrypt it. Whenever data types are being converted from one format to another, the presentation layer handles these types of tasks.

Layer 7—Application Layer

The final OSI layer is the application layer, which is where the vast majority of programmers write code. Application layer protocols dictate the semantics of how requests for services are made, such as requesting a file or checking for e-mail. In Java, almost all network software written will be for the application layer, although the services of some lower layers may also be called upon.

Q Explain Internet Protocol (IP)

The Internet Protocol (IP) is a Layer 3 protocol (network layer) that is used to transmit data packets over the Internet. It is undoubtedly the most widely used networking protocol in the world. Regardless of what type of networking hardware is used, it will almost certainly support IP networking. IP acts as a bridge between networks of different types, forming a worldwide network of computers and smaller subnetworks (see Figure) Indeed, many organizations use the IP and related protocols within their local area networks, as it can be applied equally well internally as externally.

The Internet Protocol is a packet-switching network protocol. Information is exchanged between two hosts in the form of IP packets, also known as IP datagrams. Each datagram is treated as a discrete unit, unrelated to any other previously sent packet—there are no "connections" between machines at the network layer. Instead, a series of datagrams are sent and higher-level protocols at the transport layer provide connection services.

Q Explain IP Address

The addressing of IP datagrams is an important issue, as applications require a way to deliver packets to specific machines and to identify the sender. Each host machine under the Internet Protocol has a unique address, the IP address.

The IP address is a four-byte (32-bit) address, which is usually expressed in dotted decimal format (e.g., 192.168.0.6). Although a physical address will normally be issued to a machine, once outside the local network in which it resides, the physical address is not very useful. Even if somehow every machine could be located by its physical address, if the address changed for any reason , then the machine would no longer be locatable. Instead, a new type of address is introduced, that is not bound to a particular physical location. IP address is a numerical number that uniquely identifies a machine on the Internet. Typically, one machine has a single IP address, but it can have multiple addresses. A machine could, for example, have more than one network card, or could be assigned multiple IP addresses (known as virtual addresses) so that it can appear to the outside world as many different machines. Machines connected to the Internet can send data to that IP address, and routers and gateways ensure delivery of the message. In normal programming, only the IP address is needed—the physical address is neither useful nor accessible in Java.

Q Explain in short Host Name

While numerical address values serve the purposes of computers, they are not designed with people in mind. Users who can remember thousands of 32-bit IP addresses in dotted decimal format and store them in their head are few and far between. A much simpler addressing mechanism is to associate an easy-to-remember textual name with an IP address. This text name is known as the hostname. For example, companies on the Internet usually choose a .com address, such as www.microsoft.com, or java.sun.com.

Q Explain Transmission Control Protocol

The Transmission Control Protocol (TCP) is a Layer 4 protocol (transport layer) that provides guaranteed delivery and ordering of bytes. TCP uses the Internet Protocol to send TCP segments, which contain additional information that allows it to order packets and resend them if they go astray. TCP also adds an extra layer of abstraction, by using a communications port.
A communications port is a numerical value (usually in the range 0–65,535) that can be used to distinguish one application or service from another. An IP address can be thought of as the location of a block of apartments, and the port as the apartment number. One host machine can have many applications connected to one or more ports. An application could connect to a Web server running on a particular host, and also to an e-mail server to check for new mail. Ports make all of this possible.

TCP's main advantage is that it guarantees delivery and ordering of data, providing a simpler programming interface. However, this simplicity comes at a cost, reducing network performance. For faster communication, the User Datagram Protocol may be used.

Q Explain User Datagram Protocol

The User Datagram Protocol (UDP) is a Layer 4 protocol (transport layer) that applications can use to send packets of data across the Internet (as opposed to TCP, which sends a sequence of bytes). Raw access to IP datagrams is not very useful, as there is no easy way to determine which application a packet is for. Like TCP, UDP supports a port number, so it can be used to send datagrams to specific applications and services. Unlike TCP, UDP does not guarantee delivery of packets, or that they will arrive in the correct order.

In fact, UDP differs very little from IP datagrams, save for the introduction of a port number. It may seem puzzling why anyone would want to use an unreliable packet delivery system. The additional error checking of TCP adds overhead and delays, so UDP might be seen to offer better performance. It is sufficient to realize that error-free transmission comes at a cost, and UDP can be used as an alternative.

Q Explain various Internet Application Protocols

Telnet

Telnet is a service that allows users to open a remote-terminal session to a specific machine. This allows Unix users, for example, to access their account from terminal servers or desktop machines. Since Unix servers are intended to support multiple users, a telnet session is often used, as only one person can access the machine from the local terminal (using a keyboard and monitor). Telnet allows many users to connect over the network and to access their accounts as if they were doing so locally. Telnet services use TCP port 23.

File Transfer Protocol (FTP)

The ability to transfer files is extremely important. Even before the World Wide Web, people distributed images, documents, and software using the File Transfer Protocol (FTP). FTP allows a user to log in (using a special username and password), or to attempt an anonymous log-in (by using the username of "anonymous"). FTP servers will often grant different access permissions depending on the user. For example, an anonymous account might be unable to write a file to the server, but may be able to read all files. FTP uses two TCP ports for communication—port 21 is used to control sessions and port 20 is used for the actual transfer of file contents.

Post Office Protocol Version 3 (POP3)

E-mail has become a vital part of modern life. With the exception of Web-based e-mail or specialized accounts, the majority of people access their e-mail using the Post Office Protocol, version 3 (POP3), which uses TCP port 110. Messages are stored on a server, retrieved by an email client, and then deleted from the server. This allows users to read mail offline, without being connected to the Internet.

Internet Message Access Protocol (IMAP)

While many browsers and e-mail clients support only POP3, some also support the Internet Message Access Protocol (IMAP). This protocol is less popular, as it requires a continual connection to the mail server, and thus increases bandwidth consumption and disk usage since messages are not stored on the user's system. IMAP allows users to create folders on the mail server, and also allows online searching of mail. IMAP uses TCP port 143.

Simple Mail Transfer Protocol (SMTP)

The Simple Mail Transfer Protocol allows messages to be delivered over the Internet. The separation between retrieving mail and sending mail might be perceived as a bit strange. However, separation actually simplifies the process considerably, allowing different mail-retrieval protocols to be used and enabling custom mail accounts. SMTP uses TCP port 25.

HyperText Transfer Protocol (HTTP)

HTTP is one of the most popular protocols in use on the Internet today; it made the World Wide Web possible. HTTP is an extremely important protocol, and Java includes good HTTP support. HTTP uses TCP port 80.

Finger

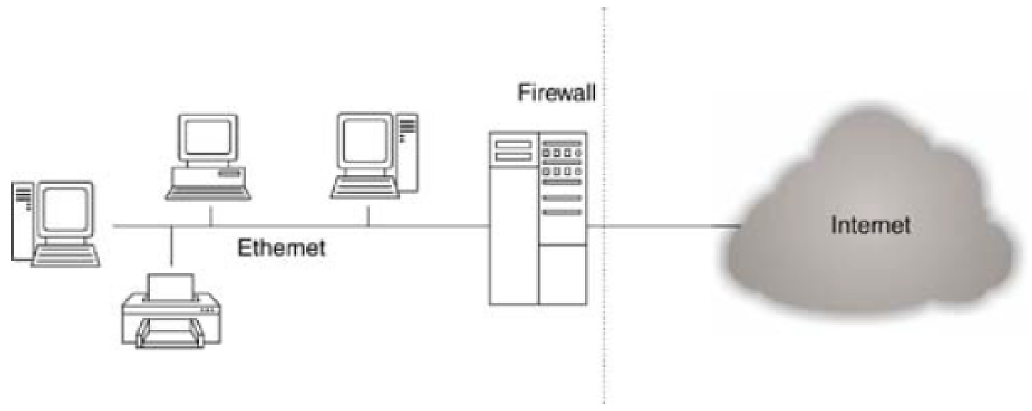
Finger is a handy protocol that allows someone to look up a person's account and find out certain information, such as when they last logged in and checked their mail. Typically, only Unix servers support finger. Unfortunately, many administrators disable finger access for security reasons, and so it is no longer as prevalent as it was. Finger uses TCP port 79.

Network News Transport Protocol (NNTP)

The Network News Transport Protocol allows users to access Usenet newsgroups. Usenet is a collection of discussion forums on a colorful and diverse number of topics, ranging from political and social commentary, to fan discussions about television programs, movies, and actors, to computing and business. NNTP uses TCP port 119.

Q Explain Firewalls

The firewall is a special machine that has been configured specifically to prohibit harmful (or as is sometimes the case in the business world, distracting) incoming or outgoing data. Usually, but not always, the firewall system will be a stripped-down computer, with all nonessential services removed to minimize the potential for cracking/hacking. The firewall is the first line of defense against intrusions from outside, and so any software that might assist in compromising the firewall should be removed, and all security patches for the operating system installed. There are many commercial firewall packages available, and some are even designed for use on desktop machines by individuals. Firewalls are most commonly separate machines except when used in companies and organizations where more than one or two individual machines are connected by, for example, a dial-up connection. The firewall works by intercepting incoming communication from machines on the Internet, and outgoing communication from machines within a local area network, as shown in Figure . It operates at the packet level, intercepting IP datagrams that reach it. By examining the header fields of these datagrams, the firewall can tell where the datagram is heading and from where it was sent. An outgoing datagram, for example, would have a source address from a machine within the firewall and a destination address from outside the firewall, whereas an incoming datagram would have a destination address of an internal machine and a source address external to the firewall. Firewalls can also help prevent a hacking technique known as masquerading, whereby an external host fakes the IP address of an internal machine to appear to be a legitimate machine, thus gaining access to resources. While there are legitimate uses for the masquerading of IP addresses (such as within an intranet), incoming datagrams from the Internet that use masquerading are suspect, and a firewall can make the distinction to filter them out.



Firewalls offer administrators very powerful security and fine-grained control of the network. Various permissions can be assigned to firewall filters. For example, outgoing data may be given greater access than incoming, and perhaps only certain machines within the network will be allowed Internet access, or only at certain times. Certain protocols, for example, might be allowed through (blocking UDP packets to certain machines and not others, or TCP access to certain port ranges). Many network administrators configure their firewalls to block all access by default, and then allow only limited network access through a proxy server.

Q Explain Proxy Servers

A proxy server is a machine that acts as a proxy for application protocols. The server accepts incoming connections from machines within a local network, and makes requests on their behalf to machines connected to the Internet. This has two advantages: direct access to internal machines is never established, and the proxy server can control the transaction. This means that popular protocols such as HTTP may be permitted or perhaps limited to certain Web sites. However, newer protocols such as RealAudio, or custom applications including games and application software, are not always permitted. Most proxy servers also log networking events, to allow network administrators to track unusual communications and their origin—in this way, employees visiting inappropriate sites or goofing off during work can be easily monitored. This might sound worrisome, and introduce some very serious legal and privacy issues, but there are legitimate security concerns addressed by logging, such as identifying disloyal employees who are visiting job-search sites or sending information to competitors.

Q What is the use of InetAddress class

A host on the Internet can be represented either in dotted decimal format as an IP address, or as a hostname such as yahoo.com. Under Java, such addresses are represented by the java.net.InetAddress class. This class can fill a variety of tasks, from resoling an IP address to looking up the hostname.

Q Explain the Methods of the java.net.InetAddress Class

The InetAddress class is used to represent IP addresses within a Java networking application. Unlike most other classes, there are no public constructors for that of InetAddress. Instead, there are two static methods that return InetAddress instances. Those and the other major methods of this class are covered in the list below.

• **byte[] getAddress()—** returns the IP address in byte format (as opposed to dotted decimal notation). The bytes are returned in network byte order, with the highest byte as bytearray[0].

• **static InetAddress[] getAllByName (String hostname) throws java.net.UnknownHostException, java.lang.SecurityException**—
returns, as a static method, an array of InetAddress instances representing the specified hostname. While most machines will have a single IP address, there are some situations in which one hostname can be mapped to many machines and/or a hostname can map to many addresses on one machine (virtual addresses). If the host cannot be resolved, or if resolving the host conflicts with the security manager, an exception will be thrown.

• **static InetAddress getByName (String hostname) throws java. net.UnknownHostException, java.lang.SecurityException**—
returns an InetAddress instance representing the specified hostname, which may be representedas either as a text hostname (e.g., davidreilly.com) or as an IP address in dotted decimal format. If the host cannot be resolved, or resolving the host conflicts with the security manager, an exception will be thrown.

• **String getHostAddress()**— returns the IP address of the InetAddress in dotted decimal format.

• **static InetAddress getLocalHost() throws java.net.UnknownHostException, java.lang.SecurityException**—
returns, as a static method, the IP address of the localhost machine. If the IP address cannot be determined, or doing so conflicts with the security manager, then an exception will be thrown.

• **String getHostName() throws java.lang.SecurityManager**— returns the hostname of the InetAddress.

Q Write a program that determine LocalHost Address

Code for LocalHostDemo

```
import java.net.*;
public class LocalHostDemo
{
public static void main(String args[])
{
System.out.println ("Looking up local host");
try
{
// Get the local host
InetAddress localAddress =
InetAddress.getLocalHost();
System.out.println ("IP address : " + localAddress.getHostAddress() );
}
catch (UnknownHostException uhe)
{
System.out.println ("Error - unable to resolve localhost");
}
}
}
```

Q Write a program to find the IP address of other pc connected in network

```
import java.net.*;
public class NetworkResolverDemo
{
public static void main(String args[])
{
if (args.length != 1)
{
System.err.println ("Syntax - NetworkResolverDemo host");
System.exit(0);
}
System.out.println ("Resolving " + args[0]);
try
{
```

```
// Resolve host and get InetAddress
InetAddress addr = InetAddress.getByName( args[0] );
System.out.println ("IP address : " +addr.getHostAddress() );
System.out.println ("Hostname : " +addr.getHostName() );
}
catch (UnknownHostException uhe)
{
System.out.println ("Error - unable to resolve hostname" );
}
}
}
```

Q Write a program to find the IP address and name of other pc connected in network through its host name

```
import java.net.*;
public class ExampleByName {
public static void main (String[] args) {
try {
InetAddress address = InetAddress.getByName("www.yahoo.com");
System.out.println(address);
}
catch (UnknownHostException e) {
System.out.println("Could not find www.yahoo.com");
}
}}
```

Q Write a program to find the IP address and name of other pc connected in network through its IP address

```
import java.net.*;
public class ExampleByAddress {
public static void main (String[] args) {
try {
InetAddress address = InetAddress.getByName("196.148.60.9");
System.out.println(address);
}
catch (UnknownHostException e) {
System.out.println("Could not find 196.148.60.9");
}
}
}
```

Q Write a program to find the all IP address of pc connected in network which have multiple IP address

```
import java.net.*;
public class AllAddressesOfMicrosoft {
public static void main (String[] args) {
```

```
try {
InetAddress[] addresses =
InetAddress.getAllByName("www.microsoft.com");
for (int i = 0; i < addresses.length; i++) {
System.out.println(addresses[i]);
}
}
catch (UnknownHostException e) {
System.out.println("Could not find www.microsoft.com");
}
}
}
```

Q Explain User Datagram Protocol (UDP) with their advantages

The User Datagram Protocol (UDP) is a commonly used transport protocol employed by many types of applications. UDP is a connectionless transport protocol, meaning that it doesn't guarantee either packet delivery or that packets arrive in sequential order. Rather than reading from, and writing to, an ordered sequence of bytes (using I/O streams), bytes of data are grouped together in discrete packets, which are sent over the network. Although control over the ultimate destination of a UDP packet rests with the computer that sends it, how it reaches that destination is an arbitrary process

The packets may travel along different paths, as selected by the various network routers that distribute traffic flow—depending on factors such as network congestion,priority of routes, and cost of transmission. (For example, for one packet a cheaper network route might be selected, even though it is slower, but another might travel along a superfast pipeline if the cheaper alternative becomes too congested.) This means that a packet can arrive out of sequence, if it encounters a faster route than the previous packet (or if the previous packet encounters some other form of delay). No two packets are guaranteed the same route, and if a particular route is heavily congested, the packet may be discarded entirely. Each packet has a time-to-live (TTL) counter, which is updated when the packet is routed along to the next point in the network. When the timer expires, it will be discarded, and the recipient of the packet will not be notified. If a packet does arrive, however, it will always arrive intact. Packets that are corrupt or only partially delivered are discarded.

Given the potential for loss of data packets, it may seem odd that anyone would even consider using such an unreliable, seemingly anarchical system. In fact, there are many advantages to using UDP that may not be apparent at first glance.

Advantages of UDP

• UDP communication can be more efficient than guaranteed-delivery data streams. If the amount of data is small and the data is sent frequently (such as in the case of a counter whose previous value is irrelevant), it may make sense to avoid the overhead of guaranteed delivery.

• Unlike TCP streams, which establish a connection, UDP causes fewer overheads. If the amount of data being sent is small and the data is sent infrequently, the overhead of establishing a connection might not be worth it. UDP may be preferable in this case, particularly if data is being sent from a large number of machines to one central one, in which case the sum total of all these connections might cause significant overload.

• Real-time applications that demand up-to-the-second or better performance may be candidates for UDP, as there are fewer delays due to the error checking and flow control of TCP. UDP packets can be used to saturate available network bandwidth to deliver large amounts of data (such as streaming video/audio, or telemetry data for a multiplayer network game). In addition, if some data is lost, it can be replaced by the next set of packets with updated information, eliminating the need to resend old data that is now out of date.

• UDP sockets can receive data from more than one host machine. If several machines must be communicated with, then UDP may be more convenient than other mechanisms such as TCP • Some network protocols specify UDP as the transport mechanism, requiring its use.
Java supports the User Datagram Protocol in the form of two classes:
• java.net.DatagramPacket
• java.net.DatagramSocket

Q Expain DatagramPacket Class

The DatagramPacket class represents a data packet intended for transmission using the User Datagram Protocol . Packets are containers for a small sequence of bytes, and include addressing information such as an IP address and a port.

When a **DatagramPacket** has been read from a UDP socket, the IP address of the packet represents the address of the sender (likewise with the port number). However, when a **DatagramPacket** is used to send a UDP packet, the IP address stored in **DatagramPacket** represents the address of the recipient (likewise with the port number). This reversal of meaning is important to remember—one wouldn't want to send a packet back to oneself!

Constructors

The choice of which **DatagramPacket** constructor to use is determined by its intended purpose.Either constructor requires the specification of a byte array, which will be used to store the UDP packet contents, and the length of the data packet.

To create a **DatagramPacket** for receiving incoming UDP packets, the following constructor should be used:
DatagramPacket(byte[] buffer, int length). For example:

DatagramPacket packet = new DatagramPacket(new byte[256], 256);

To send a **DatagramPacket** to a remote machine, it is preferable to use the following constructor:
DatagramPacket(byte[] buffer, int length, InetAddress dest_addr, int dest_port).
For example:

```
InetAddress addr = InetAddress.getByName("192.168.0.1");
DatagramPacket packet = new DatagramPacket ( new byte[128], 128, addr, 2000);
```

Methods

• **InetAddress getAddress()**— returns the IP address from which a DatagramPacket was sent, or (if the packet is going to be sent to a remote machine), the destination IP address.
• **byte[] getData()**— returns the contents of the DatagramPacket, represented as an array of bytes.
• **int getLength()**— returns the length of the data stored in a DatagramPacket. This can be less than the actual size of the data buffer.
• **int getPort()**— returns the port number from which a DatagramPacket was sent, or (if the packet is going to be sent to a remote machine), the destination port number.
• **void setAddress(InetAddress addr)**— assigns a new destination address to a DatagramPacket.

• **void setData(byte[] buffer)**— assigns a new data buffer to the DatagramPacket. Remember to make the buffer long enough, to prevent data loss.

• **void setLength(int length)**— assigns a new length to the DatagramPacket. Remember that the length must be less than or equal to the maximum size of the data buffer, or an IllegalArgumentException will be thrown. When sending a smaller amount of data, you can adjust the length to fit—you do not need to resize the data buffer.

• **void setPort(int port)**— assigns a new destination port to a DatagramPacket.

Write constructor and methods of DatagramSocket Class

The **DatagramSocket** class provides access to a UDP socket, which allows UDP packets to be sent and received. A **DatagramPacket** is used to represent a UDP packet, and must be created prior to receiving any packets. The same **DatagramSocket** can be used to receive packets as well as to send them. However, read operations are blocking, meaning that the application will continue to wait until a packet arrives. Since UDP packets do not guarantee delivery, this can cause an application to stall if the sender does not resubmit packets.

A DatagramSocket can be used to both send and receive packets. Each DatagramSocket binds to a port on the local machine, which is used for addressing packets. The port number need not match the port number of the remote machine, but if the application is a UDP server, it will usually choose a specific port number. If the DatagramSocket is intended to be a client, and doesn't need to bind to a specific port number, a blank constructor can be specified.

Constructors

To create a client **DatagramSocket**, the following constructor is used:
DatagramSocket() throws java.net.SocketException.
To create a server Datagram Socket, the following constructor is used, which takes as a parameter the port to which the UDP service will be bound:
DatagramSocket(int port) throws java.net.SocketException.

Methods

• **void close()**— closes a socket, and unbinds it from the local port.

• **void connect(InetAddress remote_addr int remote_port)**— restricts access to the specified remote address and port. The designation is a misnomer, as UDP doesn't actually create a "connection" between one machine and another.

• **void disconnect()**— disconnects the DatagramSocket and removes any restrictions imposed on it by an earlier connect operation.

• **InetAddress getInetAddress()**— returns the remote address to which the socket is connected, or null if no such connection exists.

• **int getPort()**— returns the remote port to which the socket is connected, or –1 if no such connection exists.

• **InetAddress getLocalAddress()**— returns the local address to which the socket is bound.

• **int getLocalPort()**— returns the local port to which the socket is bound.

• **int getReceiveBufferSize()** throws java.net.SocketException— returns the maximum buffer size used for incoming UDP packets.

• **int getSendBufferSize()** throws java.net.SocketException— returns the maximum buffer size used for outgoing UDP packets.

• **void receive(DatagramPacket packet)** throws java.io.IOException— reads a UDP packet and stores the contents in the specified packet. The address and port fields of the packet will be overwritten with the sender address and port fields, and the length field of the packet will contain the length of the original packet, which can be less than the size of the packet's byte-array. If a timeout value hasn't been specified by using DatagramSocket.setSoTimeout(int duration), this method will block indefinitely. If a timeout value has been specified, a java.io.InterruptedIOException will be thrown if the time is exceeded.

• **void send(DatagramPacket packet)** throws java.io.IOException— sends a UDP packet, represented by the specified packet parameter.

• **void setReceiveBufferSize(int length)** throws java.net. SocketException— sets the maximum buffer size used for incoming UDP packets. Whether the specified length will be adhered to is dependent on the operating system.

• **void setSendBufferSize(int length)** throws java.net.SocketException— sets the maximum buffer size used for outgoing UDP packets. Whether the specified length will be adhered to is dependent on the operating system.

Q Write a program to read and send data packets using Datagram protocol

Code for PacketReceiveDemo

```
import java.net.*;
import java.io.*;
public class PacketReceiveDemo
{
public static void main (String args[])

{
try
{
```

```
System.out.println ("Binding to local port 2000");

DatagramSocket socket = new DatagramSocket(2000);
System.out.println ("Bound to local port " + socket.getLocalPort());

DatagramPacket packet = new DatagramPacket( new byte[256], 256 );

socket.receive(packet);
System.out.println ("Packet received!");

InetAddress remote_addr = packet.getAddress();
System.out.println ("Sent by : " + remote_addr.getHostAddress() );
System.out.println ("Sent from: " + packet.getPort());

ByteArrayInputStream bin = new ByteArrayInputStream (packet.getData());
for (int i=0; i < packet.getLength(); i++)
{
int data = bin.read();
  if (data == -1)
       break;
  else
       System.out.print ( (char)data) ;
}
socket.close();
}
catch (IOException t)
{
System.err.println ("Error - " + t);
}
}
}
```

Code for PacketSendDemo

```
import java.net.*;
import java.io.*;
public class PacketSendDemo
{
public static void main (String args[])
{
int argc = args.length;
if (argc != 1)
{
System.out.println ("Syntax :");
System.out.println ("java PacketSendDemo hostname");
return;
}
String hostname = args[0];
try
{
```

```
System.out.println ("Binding to a local port");
DatagramSocket socket = new DatagramSocket();
System.out.println ("Bound to local port " + socket.getLocalPort());
// Create a message to send using a UDP packet
ByteArrayOutputStream bout = new ByteArrayOutputStream();
PrintStream pout = new PrintStream (bout);
pout.print ("Greetings!");
byte[] barray = bout.toByteArray();

DatagramPacket packet = new DatagramPacket( barray, barray.length );
System.out.println ("Looking up hostname " + hostname );
InetAddress remote_addr = InetAddress.getByName(hostname);
System.out.println ("Hostname resolved as " + remote_addr.getHostAddress());
packet.setAddress (remote_addr);
packet.setPort (2000);

socket.send(packet);
System.out.println ("Packet sent!");
}
catch (UnknownHostException uhe)
{
System.err.println ("Can't find host " +
hostname);
}
catch (IOException t)
{
System.err.println ("Error - " + t);
}
}
}
```

Q Write a program to create UDP Client/Server system using Datagram

The following example involves building an echo service, which transmits any packet it receives straight back to the sender.

```
import java.net.*;
import java.io.*;
public class EchoServer
{
     // UDP port to which service is bound
   public static final int SERVICE_PORT = 7;
     // Max size of packet, large enough for almost any client

   public static final int BUFSIZE = 4096;
    // Socket used for reading and writing UDP packets
    private DatagramSocket socket;
public EchoServer()
{
 try
 {
     // Bind to the specified UDP port, to listen  for incoming data packets
    socket = new DatagramSocket( SERVICE_PORT );
    System.out.println ("Server active on port " + socket.getLocalPort() );
 }
 catch (Exception e)
  {
   System.err.println ("Unable to bind port");
   }
}
public void serviceClients()
{
   // Create a buffer large enough for incoming packets
   byte[] buffer = new byte[BUFSIZE];
for (;;)
{
  try
   {
      // Create a DatagramPacket for reading  UDP packets
   DatagramPacket packet = new DatagramPacket( buffer, BUFSIZE );
     // Receive incoming packets
   socket.receive(packet);
   System.out.println ("Packet received from " +   packet.getAddress() + ":" +
                                          packet.getPort() +   " of length " + packet.getLength() );
     // Echo the packet back - address and port  are already set for us !
    socket.send(packet);
  }
  catch (IOException t)
  {
    System.err.println ("Error : " + t);
  }
}
}
public static void main(String args[])
{
   EchoServer server = new EchoServer();
   server.serviceClients();
}
}
```

An Echo Client

```
import java.net.*;
import java.io.*;
public class EchoClient
{
     // UDP port to which service is bound
  public static final int SERVICE_PORT = 7;
    // Max size of packet
   public static final int BUFSIZE = 256;
  public static void main(String args[])
 {
    if (args.length != 1)
    {
      System.err.println ("Syntax - java EchoClienthostname");
      return;
     }
   String hostname = args[0];
       // Get an InetAddress for the specified hostname
   InetAddress addr = null;
  try
  {
     // Resolve the hostname to an InetAddr
   addr = InetAddress.getByName(hostname);
  }
catch (UnknownHostException t)
{
   System.err.println ("Unable to resolve host");
   return;
}
try
{
    // Bind to any free port
DatagramSocket socket = new DatagramSocket();
    // Set a timeout value of two seconds
socket.setSoTimeout (2 * 1000);
for (int i = 1 ; i <= 10; i++)
{
     // Copy some data to our packet
  String message = "Packet number " + i ;
    char[] cArray = message.toCharArray();
    byte[] sendbuf = new byte[cArray.length];
for (int offset = 0; offset < cArray.length ; offset++)
{
   sendbuf[offset] = (byte) cArray[offset];
}
     // Create a packet to send to the UDP server
  DatagramPacket sendPacket = new DatagramPacket(sendbuf,
  cArray.length, addr, SERVICE_PORT);
  System.out.println ("Sending packet to " + hostname);
    // Send the packet
  socket.send (sendPacket);
  System.out.print ("Waiting for packet.... ");
    // Create a small packet for receiving UDP packets
  byte[] recbuf = new byte[BUFSIZE];
  DatagramPacket receivePacket = new
  DatagramPacket(recbuf, BUFSIZE);
    // Declare a timeout flag
  boolean timeout = false;
    // Catch any InterruptedIOException that is thrown
    // while waiting to receive a UDP packet
try
{
    socket.receive (receivePacket);
}
catch (InterruptedIOException t)
 {
   timeout = true;
 }
if (!timeout)
 {
  System.out.println ("packet received!");
  System.out.println ("Details : " + receivePacket.getAddress());
   // Obtain a byte input stream to read the
   // UDP packet
ByteArrayInputStream bin = new ByteArrayInputStream (receivePacket.getData(),
                                                      0,receivePacket.getLength() );
   // Connect a reader for easier access
BufferedReader reader = new BufferedReader ( new InputStreamReader ( bin ) );
   // Loop indefinitely
for (;;)
{
    String line = reader.readLine();
       // Check for end of data
     if (line == null)
        break;
     else
       System.out.println (line);
  }
}
else
{
   System.out.println ("packet lost!");
}
// Sleep for a second, to allow user to see packet
try
{
   Thread.sleep(1000);
} catch (InterruptedException t) { }
}
}
catch (IOException t)
{
System.err.println ("Socket error " + t);
}
}
}
```

Q Explain in short Transmission Control Protocol

TCP provides an interface to network communications that is radically different from the User Datagram Protocol (UDP). The properties of TCP make it highly attractive to network programmers, as it simplifies network communication by removing many of the obstacles of UDP, such as ordering of packets and packet loss. While UDP is concerned with the transmission of packets of data, TCP focuses instead on establishing a network connection, through which a stream of bytes may be sent and received. In previous topic we saw that packets may be sent through a network using various paths and may arrive at different times. This benefits performance and robustness, as the loss of a single packet doesn't necessarily disrupt the transmission of other packets. Nonetheless, such a system creates extra work for programmers who need to guarantee delivery of data. TCP eliminates this extra work by guaranteeing delivery and order, providing for a reliable byte communication stream between client and server that supports two-way communication. It establishes a "virtual connection" between two machines, through which streams of data may be sent.

TCP uses a lower-level communications protocol, the Internet Protocol (IP), to establish the connection between machines. This connection provides an interface that allows streams of bytes to be sent and received, and transparently converts the data into IP datagram packets. A common problem with datagrams, is that they do not guarantee that packets arrive at their destination. TCP takes care of this problem. It provides guaranteed delivery of bytes of data. Of course, it's always possible that network errors will prevent delivery, but TCP handles the implementation issues such as resending packets, and alerts the programmer only in serious cases such as if there is no route to a network host or if a connection is lost.

The virtual connection between two machines is represented by a socket. Sockets, allow data to be sent and received; there are substantial differences between a UDP socket and a TCP socket, however. First, TCP sockets are connected to a single machine, whereas UDP sockets may transmit or receive data from multiple machines. Second, UDP sockets only send and receive packets of data, whereas TCP allows transmission of data through byte streams (represented as an InputStream and OutputStream). They are converted into datagram packets for transmission over the network, without requiring the programmer to intervene.

Q Differentiate between TCP and UDP

TCP	UDP
No Packet Loss	Packet loss will be there
TCP is concerned instead on establishing a network connection	UDP is concerned with the transmission of packets
Guaranteed transmission	No guarantee of data transmission
No Need of extra work for programmer for delivery of data	Need of extra work for programmer for delivery of data
Two way communication between client and server	No two way communication between client and server
TCP sockets are connected to a single machine	UDP sockets are connected to multiple machines

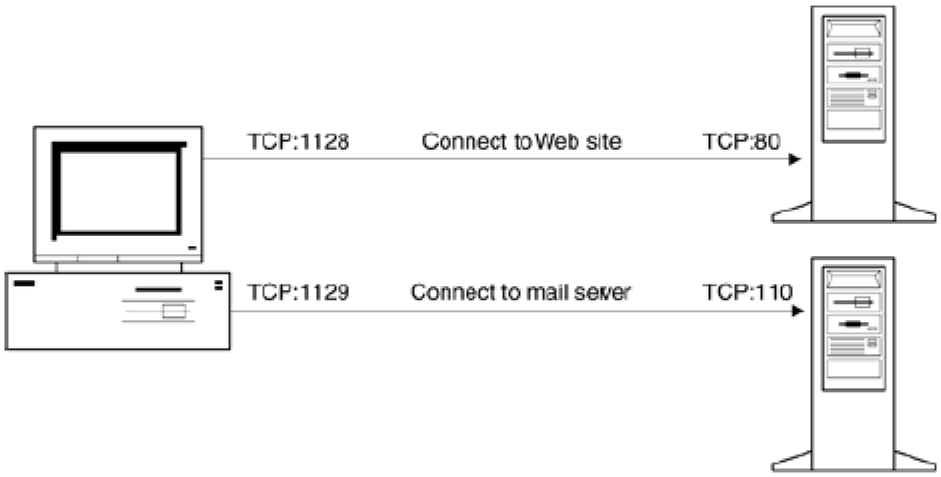
Q List advantages of TCP over UDP

The many advantages to using TCP over UDP are briefly summarized below.

Automatic Error Control
Data transmission over TCP streams is more dependable than transmission of packets of information via UDP. Under TCP, data packets sent through a virtual connection include a checksum to ensure that they have not been corrupted, just like UDP. However, delivery of data is guaranteed by the TCP— data packets lost in transit are retransmitted.
Reliability
Since the data sent between two machines participating in a TCP connection is transmitted by IP datagrams, the datagram packets will frequently arrive out of order. This would throw for a loop any program reading information from a TCP socket, as the order of the byte stream would be disrupted and frequently unreliable. Fortunately, issues such as ordering are handled by TCP— each datagram packet contains a sequence number that is used to order data. Later packets arriving before earlier packets will be held in a queue until an ordered sequence of data is available. The data will then be passed to the application through the interface of the socket.

Ease of Use
While storing information in datagram packets is certainly not beyond the reach of programmers, it doesn't lead to the most efficient way of communication between computers. There's added complexity, and it can be argued that the task of designing and creating software within a deadline provides complexity enough for programmers. Developers typically welcome anything that can reduce the complexity of software development, and the TCP does just this.

Q Explain how Communication take place between Applications Using Ports
Many services and clients run on the same port, and it would be impossible to sort out which one was which without distributing them by port number. When a TCP socket establishes a connection to another machine, it requires two very important pieces of information to connect to the remote end—the IP address of the machine and the port number. In addition, a local IP address and port number will be bound to it, so that the remote machine can identify which application established the connection (as illustrated in Figure). After all, you wouldn't want your e-mail to be accessible by another user running software
on the same system.



Q Explain Operations performed by Socket
TCP sockets can perform a variety of operations. They can:

• Establish a connection to a remote host
• Send data to a remote host
• Receive data from a remote host
• Close a connection
In addition, there is a special type of socket that provides a service that will bind to a specific port number. This type of socket is normally used only in servers, and can perform the following operations:

Q Explain The Client/Server Paradigm in short

The client/server paradigm divides software into two categories, clients and servers. A client is software that initiates a connection and sends requests, whereas a server is software that listens for connections and processes requests.

Network Clients

Network clients initiate connections and usually take charge of network transactions. The server is there to fulfill the requests of the client—a client does not fulfill the requests of a server. The network client speaks to the server using an agreed-upon standard for communication, the network protocol. For example, an HTTP client uses a set of commands different from a mail client, and has a completely different purpose. Connecting an HTTP client to a mail server, or a mail client to an HTTP server, will result not only in an error message but in an error message that the client will not understand. For this reason, as part of the protocol specification, a port number is used so that the client can locate the server. A Web server typically runs on port 80.

Network Servers

The role of the network server is to bind to a specific port (which is used by the client to locate the server), and to listen for new connections. While the client is temporary, and runs only when the user chooses, the server must run continually. It runs indefinitely, and is normally automatically started when the host computer of the server is started. So the server waits, and waits, and waits, until a client establishes a connection to the server port. Some servers can handle only a single connection at a time, while others can handle many connections concurrently, through the use of threads. When a connection is being processed, the server is submissive. It waits for the client to send requests, and dutifully processes them.

Q Explain TCP Socket Class

The Socket class represents client sockets, and is a communication channel between two TCP communications ports belonging to one or two machines. A socket may connect to a port on the local system, avoiding the need for a second machine, but most network software will usually involve two machines. TCP sockets can't communicate with more than two machines.

Constructors

The constructor for the **java.net.Socket** class is.

Socket (String ipaddress,int portno)

The easiest way to create a socket is to specify the hostname of the machine and the port of the service. For example, to connect to a Web server on port 80, the following code might be used:

```
try
{
Socket mySocket = new Socket ( "www.awl.com", 80);
// ......
}
catch (Exception e)
{  System.err.println ("Err – " + e); }
```

• **protected Socket ()**— creates an unconnected socket using the default implementation provided by the current socket factory.

• **Socket (InetAddress address int port) throws java.io.IOException**, **java.lang.SecurityException**— creates a socket connected to the specified IP address and port. If a connection cannot be established, or if connecting to that host violates a security restriction (such as when an applet tries to connect to a machine other than the machine from which it was loaded), an exception is thrown.

• **Socket (String host, int port) throws java.net.UnknownHostException, java.io.IOException, java.lang.SecurityException**— creates a socket connected to the specified host and port. This method allows a string to be specified, rather than an InetAddress. If the hostname could not be resolved, a connection could not be established, or a security restriction is violated, an exception is thrown.

Methods

• **void close()** throws **java.io.IOException**— closes the socket connection.

• **InetAddress getInetAddress()**— returns the address of the remote machine that is connected to the socket.

• **InputStream getInputStream() throws java.io.IOException**— returns an input stream, which reads from the application this socket is connected to.

• **OutputStream getOutputStream() throws java.io.IOException**— returns an output stream, which writes to the application that this socket is connected to.

• **InetAddress getLocalAddress()**— returns the local address associated with the socket (useful in the case of multihomed machines).

• **int getLocalPort()**— returns the port number that the socket is bound to on the local machine.

• **int getPort()**— returns the port number of the remote service to which the socket is connected.

2.14 Creating a TCP Client

Having discussed the functionality of the Socket class, we will now examine a complete TCP client. The client we'll look at here is a daytime client, which, as its name suggests, connects to a daytime server to read the current day and time. Establishing a socket connection and reading from it is a fairly simple process, requiring very little code. By default, the daytime service runs on port 13. Not every machine has a daytime server running, but a Unix server would be a good system to run the client against.

Q Write a program which performs client server communication

Code for Client

```
import java.net.*
import java.io.*;
public class Client
{
public static final int SERVICE_PORT = 13;
BufferedReader br,in;
PrintStream pout;

public static void main(String args[])
{
try
{
            Socket daytime = new Socket (localhost, SERVICE_PORT);
            System.out.println ("Connection established");
            in = new BufferedReader ( new InputStreamReader(daytime.getInputStream()));
        OutputStream out = nextClient.getOutputStream();
        PrintStream pout = new PrintStream (out);
        br= new BufferedReader(new InputStreamReader(System.in)));

While(true)
{
        receive();
        send();
 }
}
catch (IOException ioe)
{
System.err.println ("Error " + ioe);
}
}
void send()
{
        System.out.println(“Write message to send to client”);
        try
         {
          String str=br.readLine()
          Pout.println(str);
         }
         catch (Exception t)
        { }
}

void receive()()
{
        try
         {
          String str=in.readLine()
         System.out.println(“Client”+str);
```

```
            }
           catch (Exception t)
          { }
}
}
```

Q Explain ServerSocket Class

A special type of socket, the server socket, is used to provide TCP services. Client sockets bind to any free port on the local machine, and connect to a specific server port and host. The difference with server sockets is that they bind to a specific port on the local machine, so that remote clients may locate a service. Client socket connections will connect to only one machine, whereas server sockets are capable of fulfilling the requests of multiple clients.

The way it works is simple—clients are aware of a service running on a particular port . They establish a connection, and within the server, the connection is accepted. Multiple connections can be accepted at the same time, or a server maychoose to accept only one connection at any given moment. Once accepted, the connection is represented as a normal socket, in the form of a Socket object—once you have mastered the Socket class, it becomes almost as simple to write servers as it does clients. The only difference between a server and a client is that the server binds to a specific port, using a ServerSocket object. This ServerSocket object acts as a factory for client connections—you don't need to create instances of the Socket class yourself. These connections are modeled as a normal socket, so you can connect input and output filter streams (or even a reader and writer) to the connection.

Q Explain ServerSocket class

A server socket is used to create socket on the server side . it will be bound to a local port and ready to accept incomingconnections. When clients attempt to connect, they are placed into a queue. Once all free space inthe queue is exhausted, further clients will be refused.

• **ServerSocket(int port) throws java.io.IOException, java.lang.SecurityException**— binds the server socket to the specified port number, so that remote clients may locate the TCP service. By default, the queue size is set to 50, but an alternate constructor is provided that allows modification of this setting.

• **ServerSocket(int port, int numberOfClients) throws java.io. IOException, java.lang.SecurityException—** binds the server socket to the specified port number and allocates sufficient space to the queue to support the specified number of client sockets.

Methods
All methods are public unless otherwise noted.
• **Socket accept() throws java.io.IOException, java.lang.Security Exception**— waits for a client to request a connection to the server socket, and accepts it. This is a blocking I/O operation, and will not return until a connection is made (unless the timeout socket option is set). When a connection is established, it will be returned as a Socket object.

• **void close() throws java.io.IOException**— closes the server socket, which unbinds the TCP port and allows other services to use it.
• **InetAddress getInetAddress()**— returns the address of the server socket, which may be different from the local address in the case of a multihomed machine (i.e., a machine whose localhost is known by two or more IP addresses).

• **int getLocalPort()**— returns the port number to which the server socket is bound.

Code for Server

```
import java.net.*;
import java.io.*;
public class Server
{
 public static final int SERVICE_PORT = 13;
BufferedReader br,in;
PrintStream pout;
public static void main(String args[])
{
   ServerSocket server = new ServerSocket (SERVICE_PORT);
  while(true)
 {
  try
   {
        Socket nextClient = server.accept();
        OutputStream out = nextClient.getOutputStream();
        PrintStream pout = new PrintStream (out);
         in= new BufferedReader (new
                                InputStreamReader(Client.getInputStream()));
         br= new BufferedReader(new InputStreamReader(System.in)));
         send();
         receive();
  }catch(exception e){}
}

void send()
{
        System.out.println("Write message to send to client");
        try
         {
          String str=br.readLine()
          Pout.println(str);
         }
         catch (Exception t)
        { }
}

void receive()
{
        try
         {
          String str=in.readLine()
         System.out.println("Client"+str);
         }
         catch (Exception t){ }
}
}
```