

Chapter 3...

Networking and Security

Weightage of Marks = 16, Teaching Hours = 08

Contents

- 3.1 Introduction
- 3.2 Basics of Networking
 - 3.2.1 Socket
 - 3.2.2 IP
 - 3.2.3 TCP
 - 3.2.4 UDP (User Datagram Protocol)
 - 3.2.5 Proxy Server
 - 3.2.6 IP Address
 - 3.2.7 Internet Addressing
- 3.3 Java and the Net
 - 3.3.1 The InetAddress Class
 - 3.3.2 Factory Methods
 - 3.3.3 Instance Methods
- 3.4 TCP/IP Sockets
 - 3.4.1 Socket
 - 3.4.2 TCP/IP Client Sockets
 - 3.4.3 TCP/IP Server Sockets
- 3.5 URL
 - 3.5.1 Format of URL
 - 3.5.2 URL Connection
- 3.6 Security with Java
 - 3.6.1 java.security Package
 - 3.6.2 Permission Classes
 - 3.6.3 Policy Class
- Important Points
- Practice Questions

Objectives

- To Learn the Java's Built in support for Network Programming
- To learn about Sockets, TCP, ISP, URL and the Java Security Package

3.1 INTRODUCTION

- Communication in the present day is vital part of our existence. In today's computing world, there are many expensive resources such as laser printers, Fax machines and so on that needs to be shared.
- To facilitate this robust network have come into existence. Networks allow expensive resources to be shared.
- Java programming environment provides simple yet robust techniques that permit Java applications to communicate across networks and share valuable resources.
- Java communication is built on standard client/server architecture as shown in Fig. 3.1.
- A server must have a port number on which some software is listener/talker. This software is consistently listening for client requests.
- Both the client and server on the networks must be uniquely identified. This is where TCP/IP comes into the picture.

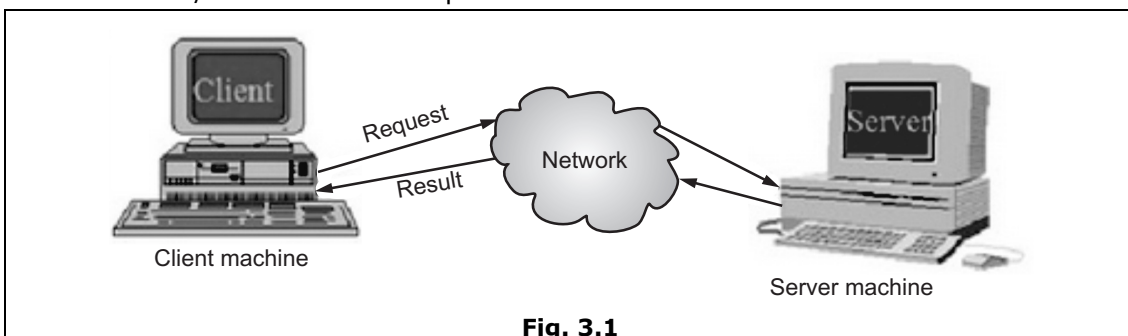


Fig. 3.1

- Using TCP/IP each computer on the network whether a server or a client can be uniquely identified using a number system that has the pattern 172.111.xxx.xxx. This is called the unique IP address of computer on a network.
- HTTP is HyperText Transfer Protocol used for web browsing. Computer networks consist of connection between computers and devices.
- Protocol is set of rules and regulations for computer communication. The computer network is collection of anonymous computers.
- Network feature made the Java programming language, more appropriate for writing networked programs. The Internet is all about connecting machines together.
- One of the most exciting aspects of Java is that it incorporates an easy-to-use, cross-platform model for network communications Berkeley's implementation of TCP/IP remains the primary standard for communications within the Internet.
- The socket paradigm for inter-process and network communication has also been widely adopted outside of Berkeley.

3.2 BASICS OF NETWORKING

3.2.1 Socket

- Java programs communicate through programming abstraction called a socket.
- A socket is simply an endpoint for communications between the machines.
- A socket is a bi-directional communication channel between hosts i.e., a computer on a network often termed as host.
- A network socket is a lot like an electrical socket. Anything that understands the standard protocol can plug into the socket and communicate.
- With electrical sockets, it does not matter if you plug in a lamp or a toaster; as long as they are expecting 50Hz, 115-volt electricity, the devices will work. The same idea applies to network sockets, except we talk about TCP/IP packets and IP addresses rather than electrons and street addresses.
- Internet Protocol (IP) is a low-level routing protocol that breaks data into small packets and sends them to an address across a network, which does not guarantee to deliver said packets to the destination.
- Transmission Control Protocol (TCP) is a higher-level protocol that manages to robustly string together these packets, sorting and re-transmitting them as necessary to reliably transmit our data.
- A third protocol, User Datagram Protocol (UDP), sits next to TCP and can be used directly to support fast, connectionless, unreliable transport of packets.

Networking:

- Network programming revolves around the concept of sockets. A socket is one end-point of a two-way communication link between two programs running on the network.
- In Java sockets are created with the help of Socket classes and these class are found in the java.net package.
- There are separate classes in the java.net package for creating TCP sockets and UDP sockets.
- The client socket is created with the help of the Socket class and the server socket is created using the ServerSocket class.
- Fig. 3.2 shows concept of socket.

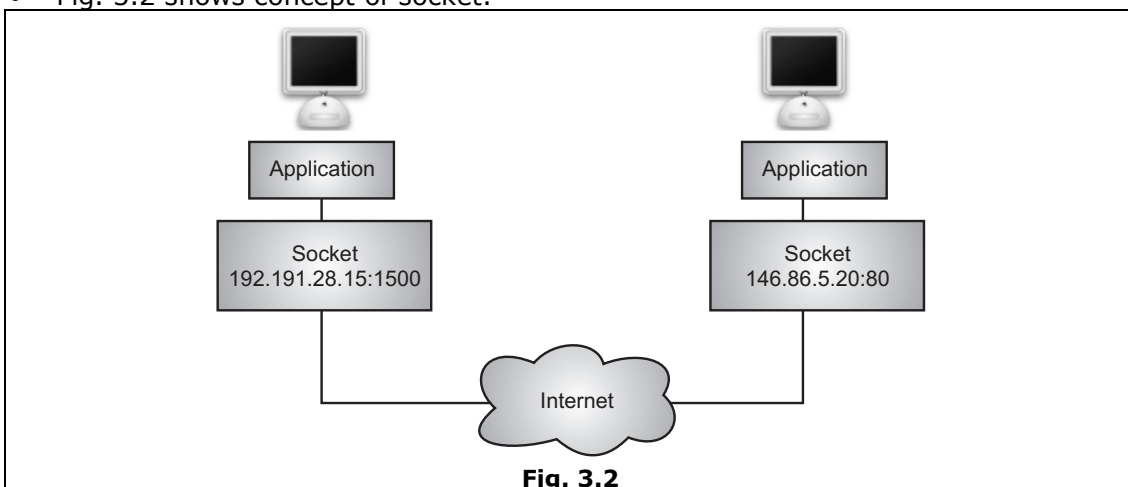


Fig. 3.2

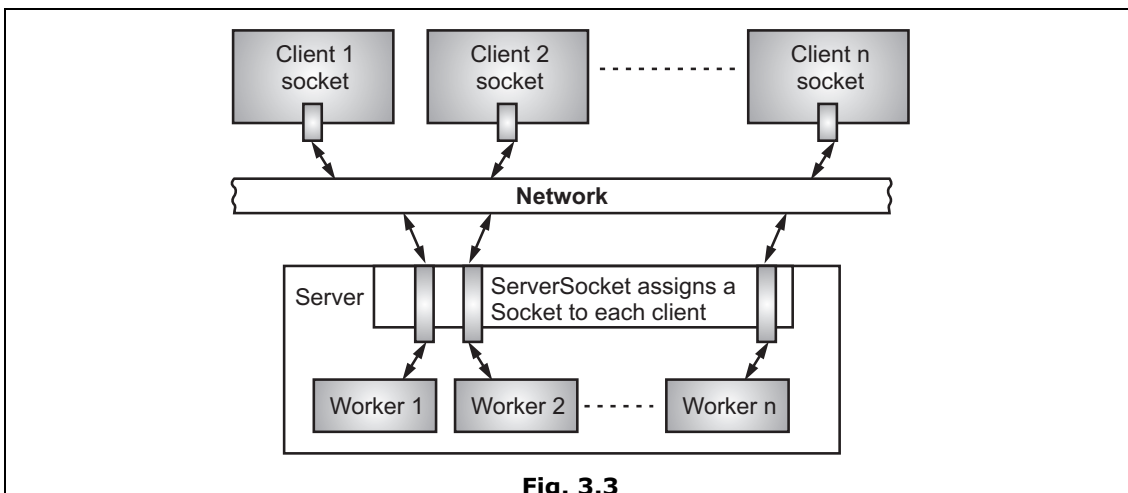
Syntax:

```
// for tcp client side
Socket s= new Socket (InetAddress adr, int port);

//for server side
ServerSocket ss= new ServerSocket (InetAddress adr, int port)
```

1. Client/Server Networking:

- A server is anything that has some resource that can be shared.
- There are computer servers, which provide computing power like print servers, which manage a collection of printers; disk servers, which provide networked disk space; and web servers, which store web pages.
- A client is simply any other entity that wants to gain access to a particular server. The interaction between client and server is just like the interaction between a lamp and an electrical socket.
- The power grid of the house is the server, and the lamp is a power client. The server is a permanently available resource, while the client is free to unplug after it is has been served.

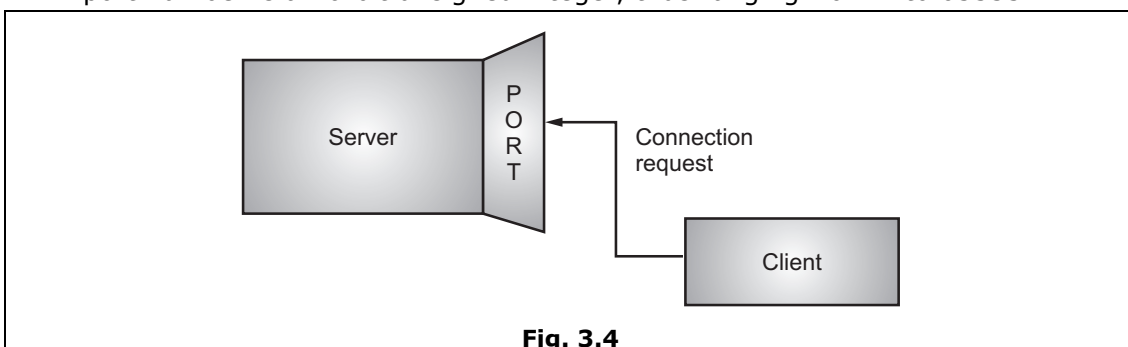
**Fig. 3.3**

- A server is allowed to accept multiple clients connected to the same port number, although each session is unique.
- To manage multiple client connections, a server process must be multithreaded or have some other means of multiplexing the simultaneous I/O.

2. Ports and Reserved Sockets:

- Once connected, a higher-level protocol ensues, which is dependent on which port we are using. A port is an endpoint to a logical connection.

- A port number identifies a specific application running in the machine. A port is a number in the range 1 to 65535. It is a transport layer address used by TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) to handle communication between applications.
- The port numbers are divided into three ranges: the well-known ports, the registered ports, and the dynamic or private ports.
- A port number is a 16-bit unsigned integer, thus ranging from 1 to 65535.

**Fig. 3.4**

- TCP/IP reserves the lower 1,024 ports for specific protocols. Many of these will seem familiar to us if we have spent any time surfing the Internet.
- Port number 21 is for FTP, 23 is for Telnet, 25 is for email, 79 is for finger, 80 is for HTTP, 119 is for net-news and the list goes on. It is up to each protocol to determine how a client should interact with the port.
- For example, HTTP is the protocol that web browsers and servers use to transfer hypertext pages and images. It is quite a simple protocol for a basic page-browsing web server.
- When a client requests a file from an HTTP server, an action known as a hit, it simply prints the name of the file in a special format to a predefined port and reads back the contents of the file.
- The server also responds with a status code number to tell the client whether the request can be fulfilled and why.

3.2.2 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 and has spread prolifically.

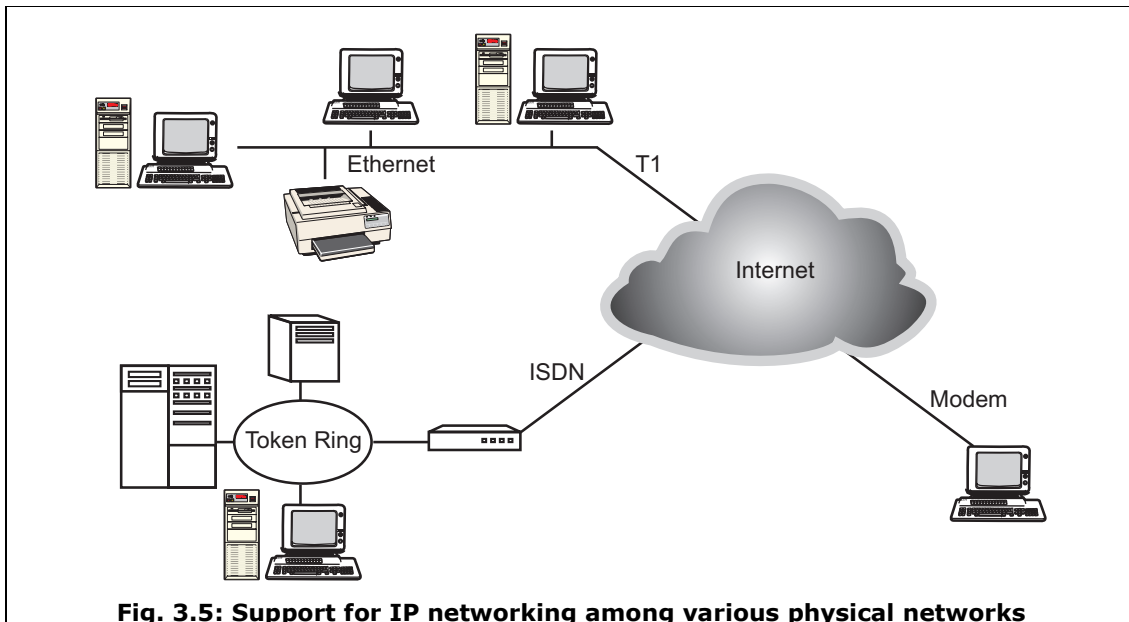


Fig. 3.5: Support for IP networking among various physical networks

- 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 Fig. 3.5).
- Indeed, many organizations use the IP and related protocols within their local area networks, as it can be applied equally well internally and 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.

3.2.3 TCP

- TCP stands for Transmission Control Protocol.
- TCP is a connection-based protocol that provides a reliable flow of data between two computers.
- TCP allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
- As shown in Fig. 3.6, the socket is the door between the application process and TCP.

- The application developer has control of everything on the application-layer side of the socket; however, it has little control of the transport-layer side.

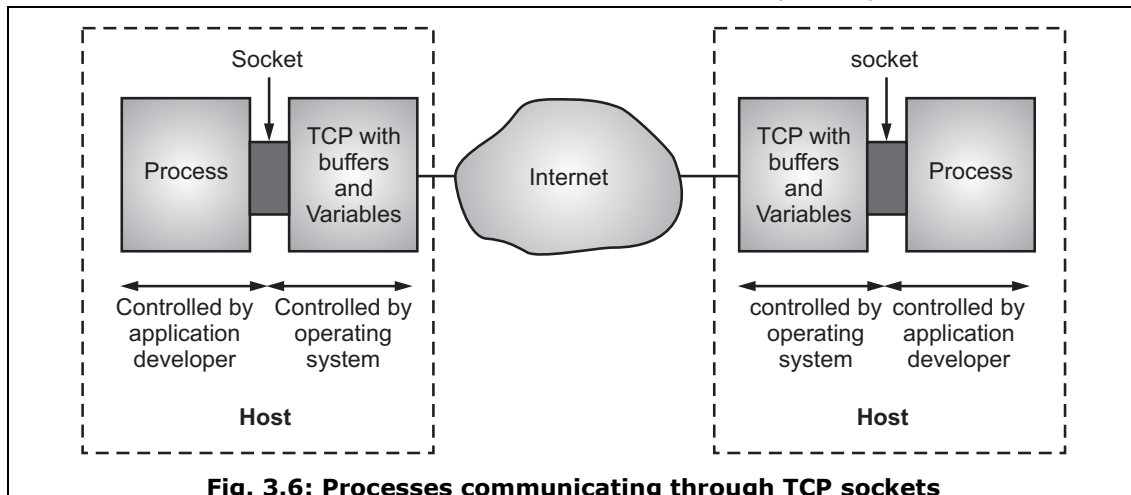


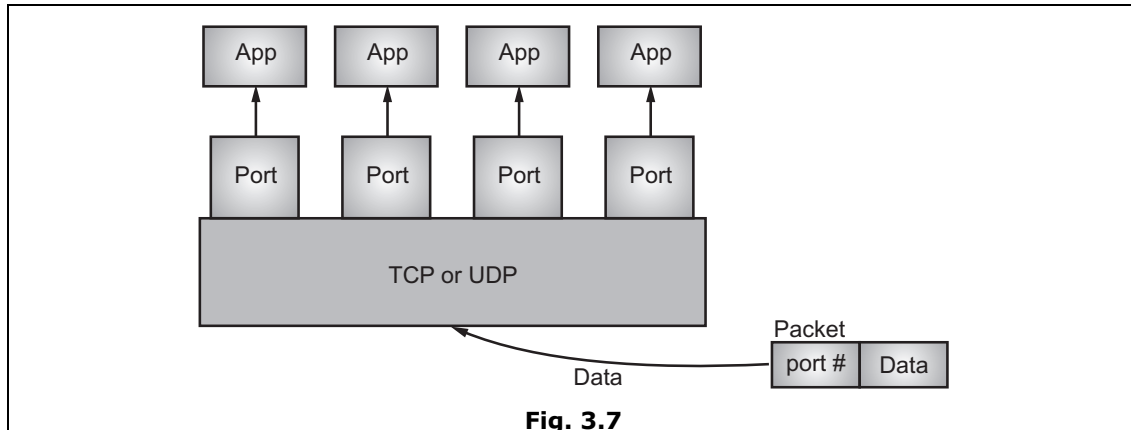
Fig. 3.6: Processes communicating through TCP sockets

- Now let us to a little closer look at the interaction of the client and server programs. The client has the job of initiating contact with the server.
- In order for the server to be able to react to the client's initial contact, the server has to be ready. This implies two things.
 1. The server program can not be dormant; it must be running as a process before the client attempts to initiate contact.
 2. The server program must have some sort of door (i.e., socket) that welcomes some initial contact from a client, (running on an arbitrary machine).

3.2.4 UDP (User Datagram Protocol)

- UDP is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival. UDP is not connection-based like TCP.
- UDP is a simple transport-layer protocol. The application writes a message to a UDP socket, which is then encapsulated in a UDP datagram, which is further encapsulated in an IP datagram, which is sent to the destination. It is described in RFC 768.
- The UDP is an alternative protocol for sending data over IP that is very quick, but not reliable.
- That is, when you send UDP data, you have no way of knowing whether it arrived, much less whether different pieces of data arrived in the order in which you sent them. However, the pieces that do arrive generally arrive quickly.

- In datagram-based communication such as UDP, the datagram packet contains the port number of its destination and UDP routes the packet to the appropriate application, as illustrated in the Fig. 3.7.

**Fig. 3.7**

- Port numbers range from 0 to 65,535 because ports are represented by 16-bit numbers. 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.

Comparison between TCP and UDP:

- The difference between TCP and UDP is often explained by analogy with the phone system and the post office.
- TCP is like the phone system. When we dial a number, the phone is answered and a connection is established between the two parties. As we talk, we know that the other party hears our words in the order in which we say them. If the phone is busy or no one answers, we find out right away.
- UDP, by contrast, is like the postal system. We send packets of mail to an address. Most of the letters arrive, but some may be lost on the way. The letters probably arrive in the order in which we sent them, but that's not guaranteed.

3.2.5 Proxy Servers

- A proxy server is software that runs on server that speaks the language of clients. This software hides the actual server from clients that communicate with it.
- A proxy server speaks the client side of a protocol to another server. This is often required when clients have certain restrictions on which servers they can connect to.

- Thus, a client would connect to a proxy server, which did not have such restrictions, and the proxy server would in turn communicate for the client.
- A proxy server has the additional ability to filter certain requests or cache the results of those requests for future use.
- A caching proxy HTTP server can help reduce the bandwidth demands on a local network's connection to the Internet.
- When a popular web site is being hit by hundreds of users, a proxy server can get the contents of the web server's popular pages once, saving expensive Internet work transfers while providing faster access to those pages to the clients.

How Caching Works?

- Caching reduces network traffic and offers faster response time for clients that are using the proxy server instead of going directly to remote servers.
- When a client requests a web page or document from the proxy server, the proxy server copies the document from the remote server to its local cache directory structure while sending the document to the client.
- When a client requests a document that was previously requested and copied into the proxy cache, the proxy returns the document from the cache instead of retrieving the document from the remote server again as shown in the Fig. 3.8.
- If the proxy determines that the file is not up to date, the proxy refreshes the document from the remote server and updates its cache before sending the document to the client.

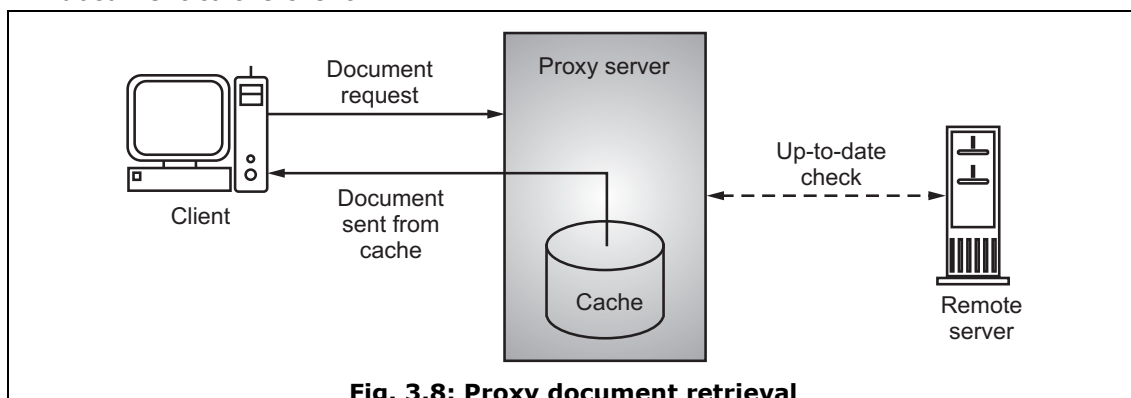


Fig. 3.8: Proxy document retrieval

- Files in the cache are automatically maintained by the Sun Java System Web Proxy Server Garbage Collection utility (CacheGC).
- The CacheGC automatically cleans the cache on a regular basis to ensure that the cache does not get cluttered with out-of-date documents.

3.2.6 IP Address

- Internet is the world's largest computer network, the network of networks, scattered all over the world.
- An IP address is a unique identification number allotted to every computer on a network or Internet.
- IP address contains some bytes which identify the network and the actual computer inside the network.
- Every computer on a network is known by a unique number which is known as IP Address. Generally, we use IPV4 which consists of 4 bytes like 172.17.167.4.

3.2.7 Internet Addressing

- Every computer on the Internet has an address. An Internet address is an umber that uniquely identifies each computer on the Net.
- Originally, all Internet addresses consisted of 32-bit values. This address type was specified by IPv4 (Internet Protocol version 4).
- However, a new addressing scheme, called IPv6 (Internet Protocol, version 6) has come into play.
- IPv6 uses a 128-bit value to represent an address. Although there are several reasons for and advantages to IPv6, the main one is that it supports a much larger address space than does IPv4. Fortunately, IPv6 is downwardly compatible with IPv4.
- There are 32 bits in an IPv4 IP address, and we often refer to them as a sequence of four numbers between 0 and 255 separated by dots (.).
- This makes them easier to remember; because they are not randomly assigned they are hierarchically assigned. The first few bits define which class of network, lettered A, B, C, D, or E, the address represents.
- Most Internet users are on a class C network, since there are over two million networks in class C. The first byte of a class C network is between 192 and 224, with the last byte actually identifying an individual computer among the 256 allowed on a single class C network. This scheme allows for half a billion devices to live on class C networks.

Domain Naming Service (DNS):

- The Internet would not be a very friendly place to navigate if everyone had to refer to their addresses as numbers. For example, it is difficult to imagine seeing <http://192.9.9.1/> at the bottom of an advertisement.
- Thankfully, a clearing house exists for a parallel hierarchy of names to go with all these numbers. It is called the Domain Naming Service (DNS).

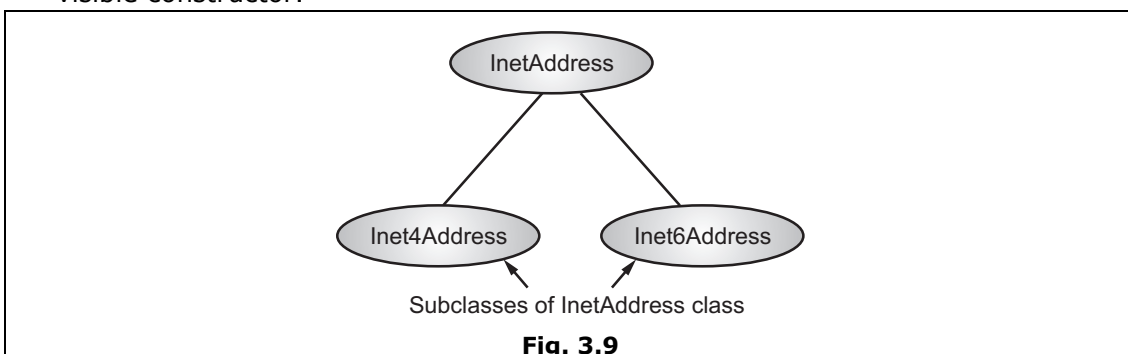
- Just as the four numbers of an IP address describe a network hierarchy from left to right, the name of an Internet address, called its domain name, describes a machine's location in a name space, from right to left.
- For example, `www.google.com` is in the COM domain (reserved for commercial sites), it is called Google (after the company name), and `www` is the name of the specific computer that is Google's web server. `www` corresponds to the rightmost number in the equivalent IP address.

3.3 JAVA AND THE NET

- Now that the stage has been set, let's take a look at how Java relates to all of these network concepts.
- Java supports TCP/IP both by extending the already established stream I/O interface and by adding the features required to build I/O objects across the network.
- Java supports both the TCP and UDP protocol families. TCP is used for reliable stream-based I/O across the network. UDP supports a simpler, hence faster, point-to-point datagram-oriented model.

3.3.1 InetAddress

- Whether we are making a phone call, sending mail, or establishing a connection across the Internet, addresses are fundamental.
- The `InetAddress` class is used to encapsulate both the numerical IP address and the domain name for that address.
- We interact with this class by using the name of an IP host, which is more convenient and understandable than its IP address. The `InetAddress` class hides the number inside.
- The `java.net.InetAddress` class represents an IP address. The `InetAddress` class provides methods to get the IP of any host name.
- `InetAddress` can handle both IPv4 and IPv6 address. `InetAddress` class has no visible constructor.



Example of InetAddress class:

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

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

InetAddress ip=InetAddress.getByName("www.google.com");

System.out.println("Host Name: "+ip.getHostName());
System.out.println("IP Address: "+ip.getHostAddress());

}catch(Exception e){System.out.println(e);}
}
}
```

3.3.2 Factory Methods

- We have already know the InetAddress class has no visible constructors. To create an InetAddress object, we have to use one of the available factory methods.
- Factory methods are merely a convention where by static methods in a class return an instance of that class.
- This is done in lie of overloading a constructor with various parameter lists when having unique method names makes theresults much clearer.
- Three commonly used InetAddress factory methods are shown below:
 1. `static InetAddress getLocalHost() throws UnknownHostException`
 2. `static InetAddress getByName(String hostName) throws
UnknownHostException`
 3. `static InetAddress[] getAllByName(String hostName)
throws UnknownHostException`
- The `getLocalHost()` method simply returns the `InetAddress` object that represents the local host. The `getByName()` method returns an `InetAddress` for a host name passed to it. If these methods are unable to resolve the hostname, they throw an `UnknownHostException`.
- On the Internet, it is common for a single name to be used to represent several machines. In the world of web servers, this is one way to provide some degree of scaling.

- The `getAllByName()` factory method returns an array of `InetAddress`s that represent all of the addresses that a particular name resolves to. It will also throw an `UnknownHostException` if it can't resolve the name to at least one address. Also includes the factory method `getByAddress()`, which takes an IP address and returns an `InetAddress` object.

```
// Demonstrate InetAddress
import java.net.*;
class InetAddressTest
{
public static void main(String args[])
throws UnknownHostException
{
InetAddress Address = InetAddress.getLocalHost();
System.out.println(Address);
Address = InetAddress.getByName("google.com");
System.out.println(Address);
InetAddress SW[] =
InetAddress.getAllByName("www.yahoo.com");
for (int i=0; i<SW.length; i++)
System.out.println(SW[i]);
}
}
```

3.3.3 Instance Methods

- The `InetAddress` class also has several other methods, which can be used on the objects returned by the methods just discussed.
- Here are some of the most commonly used:
 1. `boolean equals(Object other)`: It returns true if this object has the same Internet address as other.
 2. `byte[] getAddress()`: It returns a byte array that represents the object's Internet address in network byte order.
 3. `String getHostAddress()`: It returns a string that represents the host address associated with the `InetAddress` object.
 4. `String getHostName()`: It returns a string that represents the host name associated with the `InetAddress` object.

5. `boolean isMulticastAddress()`: It returns true if this Internet address is a multicast address. Otherwise, it returns false.
6. `String toString()`: It returns a string that lists the host name and the IP address for convenience.

3.4 TCP/IP SOCKETS

- Java offers good support for TCP sockets, in the form of two socket classes, `java.net.Socket` and `java.net.ServerSocket`.

3.4.1 Socket

- A socket is one endpoint of a two-way communication link between two programs running on the computer network.
- A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent.
- In `java.net` package there are two classes for socket purpose.
 1. `Socket`, and
 2. `ServerSocket`.
- The `ServerSocket` class is used to create a server that listens for incoming connections from one or more clients. A Server socket binds itself to a specific port so that clients can connect to it.
- The `Socket` class represents a TCP client socket, which connects to a server socket and infinite protocol exchanges.

3.4.2 TCP/IP Client Sockets

- TCP/IP sockets are used to implement reliable, bi-directional, persistent, point-to-point, and stream-based connections between hosts on the Internet.
- A socket can be used to connect Java's I/O system to other programs that may reside either on the local machine or on any other machine on the Internet.
- Applets may only establish socket connections back to the host from which the applet was downloaded. This restriction exists because it would be dangerous for applets loaded through a firewall to have access to any arbitrary machine.
- There are two kinds of TCP sockets in Java. One is for servers, and the other is for clients.
- The `ServerSocket` class is designed to be a listener, which waits for clients to connect before doing anything. The `Socket` class is designed to connect to server sockets and initiate protocol exchanges.

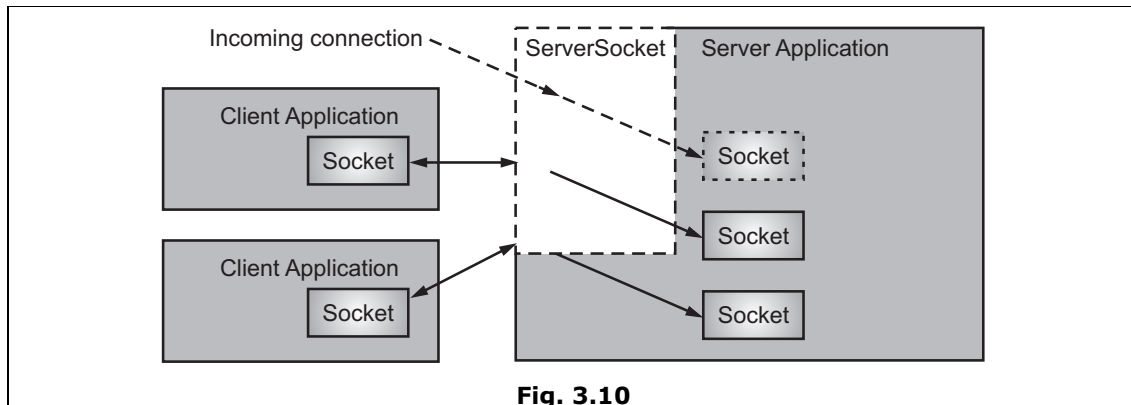


Fig. 3.10

- The creation of a Socket object implicitly establishes a connection between the client and server. There are no methods or constructors that explicitly expose the details of establishing that connection.
- Here, are two constructors used to create client sockets:
 1. `Socket(String hostName, int port)` throws `UnknownHostException`, `IOException`: Creates a socket connecting the local host to the named host and port; can throw an `UnknownHostException` or an `IOException`.
 2. `Socket(InetAddress ipAddress, int port)` throws `UnknownHostException`, `IOException`: Creates a socket using a pre-existing `InetAddress` object and a port; can throw an `IOException`.
- A socket can be examined at any time for the address and port information associated with it, by use of the following methods:
 1. `InetAddress getAddress()`: It returns the `InetAddress` associated with the Socket object.
 2. `int getPort()`: It returns the remote port to which this Socket object is connected.
 3. `int getLocalPort()`: It returns the local port to which this Socket object is connected. Once, the Socket object has been created, it can also be examined to gain access to the input and output streams associated with it. Each of these methods can throw an `IOException` if the sockets have been invalidated by a loss of connection on the Net. These streams are used exactly like the other I/O streams to send and receive data.
 4. `InputStream getInputStream()`: This returns the `InputStream` associated with the invoking socket.
 5. `OutputStream getOutputStream()`: This returns the `OutputStream` associated with the invoking socket.

Program 3.1: Program to demonstrate TCP/IP clients sockets.

```
import java.net.*;
import java.io.*;
public class LowPortScanner
{
public static void main(String[] args)
{
String host = "localhost";
for (int i = 1; i < 1024; i++)
{
try {
Socket s = new Socket(host, i);
System.out.println("There is a server on port " + i + " of " + host);
}
catch (UnknownHostException ex)
{
System.err.println(ex);
break;
}
catch (IOException ex)
{
// must not be a server on this port
}
} // end for
} // end main
} // end PortScanner
```

- Here's the output this program produces on local host.

```
java LowPortScanner
There is a server on port 21 of localhost
There is a server on port 80 of localhost
There is a server on port 110 of localhost
There is a server on port 135 of localhost
There is a server on port 443 of localhost
```

A daytime protocol client:

```
import java.net.*;
import java.io.*;
public class DaytimeClient
{
public static void main(String[] args)
{
String hostname;
try
{
Socket theSocket = new Socket("localhost", 13);
InputStream timeStream = theSocket.getInputStream( );
StringBuffer time = new StringBuffer( );
int c;
while ((c = timeStream.read( )) != -1) time.append((char) c);
String timeString = time.toString( ).trim( );
System.out.println("It is " + timeString + " at " + hostname);
// end try
catch (UnknownHostException ex)
{
System.err.println(ex);
}
catch (IOException ex)
{
System.err.println(ex);
}
} // end main
} // end DaytimeClient
```

- DaytimeClient reads the hostname of a daytime server from the command line and uses it to construct a new Socket that connects to port 13 on the server. Here's what happens:

```
java DaytimeClient
```

```
It is 52956 03-11-13 04:45:28 00 0 0 706.3 UTC(NIST) * at time.nist.gov.
```

3.4.3 TCP/IP Server Sockets

- Java has a different socket class that must be used for creating server applications.
- The `ServerSocket` class is used to create servers that listen for either local or remote client programs to connect to them on published ports.
- `ServerSockets` are quite different from normal `Sockets`. When we create a `ServerSocket`, it will register itself with the system as having an interest in client connections.
- The constructors for `ServerSocket` reflect the port number that we wish to accept connections on and, optionally, how long we want the queue for said port to be.
- The queue length tells the system how many client connections it can leave pending before it should simply refuse connections. The default is 50.
- The `ServerSocket` class contains everything needed to write servers in Java. It has constructors that create new `ServerSocket` objects, methods that listen for connections on a specified port, methods that configure the various server socket options, and the usual miscellaneous methods such as `toString()`.
- In Java, the basic life cycle of a server program is:
 1. A new `ServerSocket` is created on a particular port using a `ServerSocket()` constructor.
 2. The `ServerSocket` listens for incoming connection attempts on that port using its `accept()` method. `accept()` blocks until a client attempts to make a connection, at which point `accept()` returns a `Socket` object connecting the client and the server.
 3. Depending on the type of server, either the `Socket`'s `getInputStream()` method, `getOutputStream()` method, or both are called to get input and output streams that communicate with the client.
 4. The server and the client interact according to an agreed-upon protocol until it is time to close the connection.
 5. The server, the client, or both close the connection.
 6. The server returns to step 2 and waits for the next connection.
- The constructors might throw an `IOException` under adverse conditions. Here, are the constructors:
 1. `ServerSocket(int port)` throws `BindException`, `IOException`: It creates server socket on the specified port with a queue length of 50.
 2. `ServerSocket(int port, int maxQueue)` throws `BindException`, `IOException`: This creates a server socket on the specified port with a maximum queue length of `maxQueue`.

3. `ServerSocket(int port, int maxQueue, InetAddress localAddress)`
throws `IOException`: It creates a server socket on the specified port with a maximum queue length of `maxQueue`. On a multi-homed host, local Address specifies the IP address to which this socket binds.
- `ServerSocket` has a method called `accept()`, which is a blocking call that will wait for a client to initiate communications, and then return with a normal `Socket` that is then used for communication with the client.
-

Program 3.2: Scanner for the server ports:

```
import java.net.*;
import java.io.*;
public class LocalPortScanner
{
public static void main(String[] args)
{
for (int port = 1; port <= 65535; port++)
{
try
{
// the next line will fail and drop into the catch block if
// there is already a server running on the port
ServerSocket server = new ServerSocket(port);
}
catch (IOException ex)
{
System.out.println("There is a server on port " + port+ ".");
} // end catch
} // end for
}
}
```

Accepting and Closing Connections:

- A `ServerSocket` customarily operates in a loop that repeatedly accepts connections. Each pass through the loop invokes the `accept()` method.
- `accept()` method returns a `Socket` object representing the connection between the remote client and the local server.

- Interaction with the client takes place through this Socket object. When the transaction is finished, the server should invoke the Socket object's close() method.
- If the client closes the connection while the server is still operating, the input and/or output streams that connect the server to the client throw an InterruptedException on the next read or write.
- In either case, the server should then get ready to process the next incoming connection. However, when the server needs to shut down and not process any further incoming connections, we should invoke the ServerSocket object's close() method.

```
public Socket accept( ) throws IOException
```

- When server setup is done and we are ready to accept a connection, call the ServerSocket's accept() method. This method "blocks"; that is, it stops the flow of execution and waits until a client connects.
- When a client does connect, the accept() method returns a Socket object. We use the streams returned by this Socket's getInputStream() and getOutputStream() methods to communicate with the client.

For example:

```
ServerSocket server = new ServerSocket(5776);
while (true)
{
    Socket connection = server.accept( );
    OutputStreamWriter out = new OutputStreamWriter
        (connection.getOutputStream( ));
    out.write("You've connected to this server. Bye-bye now.\r\n");
    connection.close( );
}
```

- If we do not want the program to halt while it waits for a connection, put the call to accept() in a separate thread. Finally, most servers will want to make sure that all sockets they accept are closed when they are finished.
- Even if the protocol specifies that clients are responsible for closing connections, clients do not always strictly adhere to the protocol. The call to close() also has to be wrapped in a try block that catches an IOException.
- However, if we do catch an IOException when closing the socket, ignore it. It just means that the client closed the socket before the server could host, allowing another server to bind to the port; it also breaks all currently open sockets that the ServerSocket has accepted.

```
public InetAddress getInetAddress( )
```

- This method returns the address being used by the server (the localhost). If the local host has a single IP address (as most do), this is the address returned by `InetAddress.getLocalHost()`.
- If the local host has more than one IP address, the specific address returned is one of the host's IP addresses. We can't predict which address we will get.

For example:

```
ServerSocket httpd = new ServerSocket(80);
InetAddress ia = httpd.getInetAddress( );
```

- If the `ServerSocket` has not yet bound to a network interface, this method returns null.

```
public int getLocalPort( )
```

- The `ServerSocket` constructors allow us to listen on an unspecified port by passing 0 for the port number. This method lets us find out what port we are listening on.

Program 3.3: A Daytime server for daytime client.

```
public class DaytimeServer
{
public static void main(String[] args)
{
try
{
ServerSocket server = new ServerSocket(13);
Socket connection = null;
while (true)
{
try
{
connection = server.accept( );
Writer out = new
OutputStreamWriter(connection.getOutputStream( ));
Date now = new Date( );
out.write(now.toString( ) + "\r\n");
out.flush( );
connection.close( );
}
}
```

```
    catch (IOException ex) {}
    finally
    {
    try
    {
    if (connection != null) connection.close( );
    }
    catch (IOException ex) {}
    } // end while
    } // end try
    catch (IOException ex)
    {
    System.err.println(ex);
    } // end catch
    } // end main
}
```

3.5 URL

- The Web is a loose collection of higher-level protocols and file formats, all unified in a web browser. One of the most important aspects of the Web is that Tim Berners-Lee devised a scalable way to locate all of the resources of the Net.
- Once, we can reliably name anything and everything, it becomes a very powerful paradigm. The Uniform Resource Locator (URL) does exactly that.
- The URL provides a reasonably intelligible form to uniquely identify or address information on the Internet. URLs are ubiquitous; every browser uses them to identify information on the Web.
- In fact, the Web is really just that same old Internet with all of its resources addressed as URLs plus HTML. Within Java's network class library, the URL class provides a simple, concise API to access information across the Internet using URLs.
- URL is a reference can address to a resource on the Internet.
- The `java.net.URL` class represents a URL and has complete set of methods to manipulate URL in Java.

- The URL class has several constructors for creating URLs, including the following:
 1. `public URL(String protocol, String host, int port, String file)` throws `MalformedURLException`: Creates a URL by putting together the given parts.
 2. `public URL(String protocol, String host, String file)` throws `MalformedURLException`: Identical to the previous constructor, except that the default port for the given protocol is used.
 3. `public URL(String url)` throws `MalformedURLException`: Creates a URL from the given String.
 4. `public URL(URL context, String url)` throws `MalformedURLException`: Creates a URL by parsing the together the URL and String arguments.

3.5.1 Format of URL

- Two examples of URLs are `http://www.rediff.com/` and `http://www.rediff.com:80/index.htm/`.
- A URL specification is based on four components, (See Fig. 3.11).
 1. The first is the protocol to use, separated from the rest of the locator by a colon (:). Common protocols are `http`, `ftp`, `gopher`, and `file`, although these days almost everything is being done via HTTP (in fact, most browsers will proceed correctly if we leave off the `http://` from our URL specification).
 2. The second component is the host name or IP address of the host to use; this is delimited on the left by double slashes (`//`) and on the right by a slash (`/`) or optionally a colon (:).
 3. The third component, the port number, is an optional parameter, delimited on the left from the host name by a colon (`:`) and on the right by a slash (`/`). (It defaults to port 80, the predefined HTTP port; thus:80 is redundant.)
 4. The fourth part is the actual file path. Most HTTP servers will append a file named `index.html` or `index.htm` to URLs that refer directly to a directory resource. Thus, `http://www.rediff.com/` is the same as `http://www.rediff.com/index.htm`.
- Java URL Class in `java.net` package, deals with URL (Uniform Resource Locator) which uniquely identify or locate resource on internet.

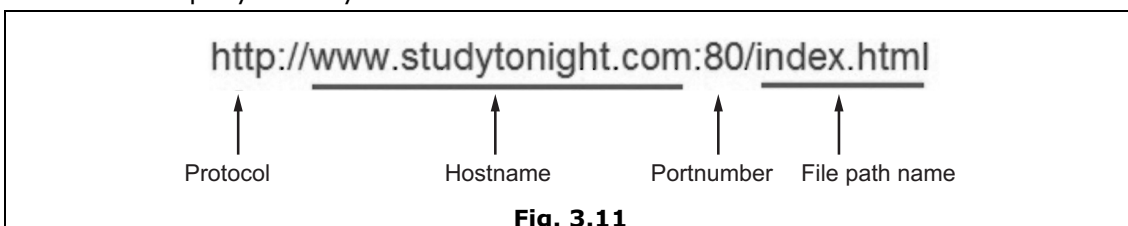


Fig. 3.11

- Java's URL class has several constructors, and each can throw a MalformedURLException. One commonly used form specifies the URL with a string that is identical to what you see displayed in a browser:

```
URL(String urlSpecifier)
```

- The next two forms of the constructor allow you to break up the URL into its component parts:

```
URL(String protocolName, String hostName, int port, String path)
```

```
URL(String protocolName, String hostName, String path)
```

- Another frequently used constructor allows us to use an existing URL as a reference context and then create a new URL from that context. Although this sounds a little contorted, it's really quite easy and useful.

```
URL(URL urlObj, String urlSpecifier)
```

Methods of URL class:

Sr. No.	Methods	Description
1.	<code>public String getPath()</code>	Returns the path of the URL.
2.	<code>public String getQuery()</code>	Returns the query part of the URL.
3.	<code>public String getAuthority()</code>	Returns the authority of the URL
4.	<code>public int getPort()</code>	Returns the port of the URL.
5.	<code>public int getDefaultPort()</code>	Returns the default port for the protocol of the URL.
6.	<code>public String getProtocol()</code>	Returns the protocol of the URL.
7.	<code>public String getHost()</code>	Returns the host of the URL.
8.	<code>public String getFile()</code>	Returns the filename of the URL.

Program 3.4: In the program, we create a URL to cric-info's news page and then examine its properties.

```
import java.net.*;
class URLEDemo
{
public static void main(String args[])
throws MalformedURLException
{
URL hp = new URL("http://contentind.
cricinfo.com/ci/content/current/story/news.html");
System.out.println("Protocol: " + hp.getProtocol());
```

```
System.out.println("Port: " + hp.getPort());
System.out.println("Host: " + hp.getHost());
System.out.println("File: " + hp.getFile());
System.out.println("Ext:" + hp.toExternalForm());
}
}
```

Output:

```
Protocol: http
Port: -1
Host: content-ind.cricinfo.com
File: /ci/content/current/story/news.html
Ext:http://content-
ind.cricinfo.com/ci/content/current/story/news.html
```

- Notice that the port is 1; this means that one was not explicitly set. No what we have created a URL object, we want to retrieve the data associated with it. To access the actual bits or content information of a URL, we create a URLConnection object from it, using its openConnection() method, like this:

```
url.openConnection()
```

openConnection() has the following general form/syntax:

```
URLConnection openConnection( )
```

- It returns a URLConnection object associated with the invoking URLObject. It may throw an IOException.

3.5.2 URLConnection

- URLConnection is an abstract class that represents an active connection o a resource specified by a URL.
- The URLConnection class has two different but related purposes. First, it provides more control over the interaction with a server (especially an HTTP server) than the URL class. With a URLConnection, we can inspect the header sent by the server and respond accordingly.
- We can set the header fields used in the client request. We can use a URLConnection to download binary files. Finally, a URLConnection lets us send data back to a web server with POST or PUT and use other HTTP request methods.

Methods of URLConnection Class:

Sr. No.	Methods	Description
1.	<code>public URLConnection openConnection() throws IOException</code>	Opens a connection to the URL, allowing a client to communicate with the resource.
2.	<code>Object getContent()</code>	Retrieves the contents of the URL connection.
3.	<code>Object getContent(Class[] classes)</code>	Retrieves the contents of the URL connection.
4.	<code>String getContentEncoding()</code>	Returns the value of the content-encoding header field.
5.	<code>int getContentLength()</code>	Returns the value of the content-length header field.
6.	<code>String getContentType()</code>	Returns the value of the content-type header field.
7.	<code>int getLastModified()</code>	Returns the value of the last-modified header field.
8.	<code>long getExpiration()</code>	Returns the value of the expires header field.
9.	<code>long getIfModifiedSince()</code>	Returns the value of this object's IfModifiedSince field.
10.	<code>public InputStream getInputStream() throws IOException</code>	Returns the input stream of the URL connection for reading from the resource.
11.	<code>public OutputStream getOutputStream() throws IOException</code>	Returns the output stream of the URL connection for writing to the resource.
12.	<code>public URL getURL()</code>	Returns the URL that this URLConnection object is connected.

- A program that uses the `URLConnection` class directly follows this basic sequence of steps:
 1. Construct a `URL` object.
 2. Invoke the `URL` object's `openConnection()` method to retrieve a `URLConnection` object for that `URL`.
 3. Configure the `URLConnection`.
 4. Read the header fields.
 5. Get an input stream and read data.
 6. Get an output stream and write data.
 7. Close the connection.
- We do not always perform all these steps. For instance, if the default setup or a particular kind of `URL` is acceptable, then we are likely to skip step 3.
- If we only want the data from the server and do not care about any meta-information, or if the protocol does not provide any meta-information, we will skip step 4.
- If we only want to receive data from the server but not send data to the server, we will skip step 6. Depending on the protocol, steps 5 and 6 may be reversed or interlaced.
- The single constructor for the `URLConnection` class is protected:

```
protected URLConnection(URL url)
```

- Consequently, unless we are sub-classing `URLConnection` to handle a new kind of `URL` (that is, writing a protocol handler), we can only get a reference to one of these objects through the `openConnection()` methods of the `URL` and `URLStreamHandler` classes.
- For example:

```
try {
    URL u = new URL("http://www.greenpeace.org/");
    URLConnection uc = u.openConnection( );
}
catch (MalformedURLException ex) {
    System.err.println(ex);
}
catch (IOException ex) {
    System.err.println(ex);
}
```

Reading Data from a Server:

- Here, is the minimal set of steps needed to retrieve data from a URL using a `URLConnection` object:
 1. Construct a `URL` object.
 2. Invoke the `URL` object's `openConnection()` method to retrieve a `URLConnection` object for that `URL`.
 3. Invoke the `URLConnection`'s `getInputStream()` method.
 4. Read from the input stream using the usual stream API.
 5. The `getInputStream()` method returns a generic `InputStream` that lets you read and parse the data that the server sends.

```
public InputStream getInputStream( )
```

Program 3.5: Download a web page with a `URLConnection`.

```
import java.net.*;
import java.io.*;
public class SourceViewer2 {
public static void main (String[] args) {
if (args.length > 0) {
try {
//Open the URLConnection for reading
URL u = new URL(args[0]);
URLConnection uc = u.openConnection( );
InputStream raw = uc.getInputStream( );
InputStream buffer = new BufferedInputStream(raw);
// chain the InputStream to a Reader
Reader r = new InputStreamReader(buffer);
int c;
while ((c = r.read( )) != -1) {
System.out.print((char) c);
}
}
catch (MalformedURLException ex) {
System.err.println(args[0] + " is not a parseable URL");
}
catch (IOException ex) {
System.err.println(ex);
}
} // end if
} // end main
} // end SourceViewer2
```

- The differences between URL and URLConnection are not apparent with just a simple input stream as in this example. The biggest differences between the two classes are:
 1. URLConnection provides access to the HTTP header.
 2. URLConnection can configure the request parameters sent to the server.
 3. URLConnection can write data to the server as well as read data from the server.

3.6 SECURITY WITH JAVA

- Security is a multifaceted feature of the Java platform. There are a number of facilities within Java that allow you to write a Java application that implements a particular security policy.
- The Java platform provides a number of features designed to improve the security of Java applications.

3.6.1 java.security Package

- The java.security package contains the classes and interfaces that implement the Java security architecture.
- These classes can be divided into two broad categories:
 - There are classes that implement access control and prevent untrusted code from performing sensitive operations.
 - There are authentication classes that implement message digests and digital signatures and can authenticate Java classes and other objects.
- The central access control class is AccessController; it uses the currently installed Policy object to decide whether a given class has Permission to access a given system resource.
- The Permissions and ProtectionDomain classes are also important pieces of the Java access control architecture.
- Fig. 3.12 shows access control classes of the java.security package.
- The key classes for authentication are MessageDigest and Signature; they compute and verify cryptographic message digests and digital signatures. These classes use public-key cryptography techniques and rely on the PublicKey and PrivateKey classes.

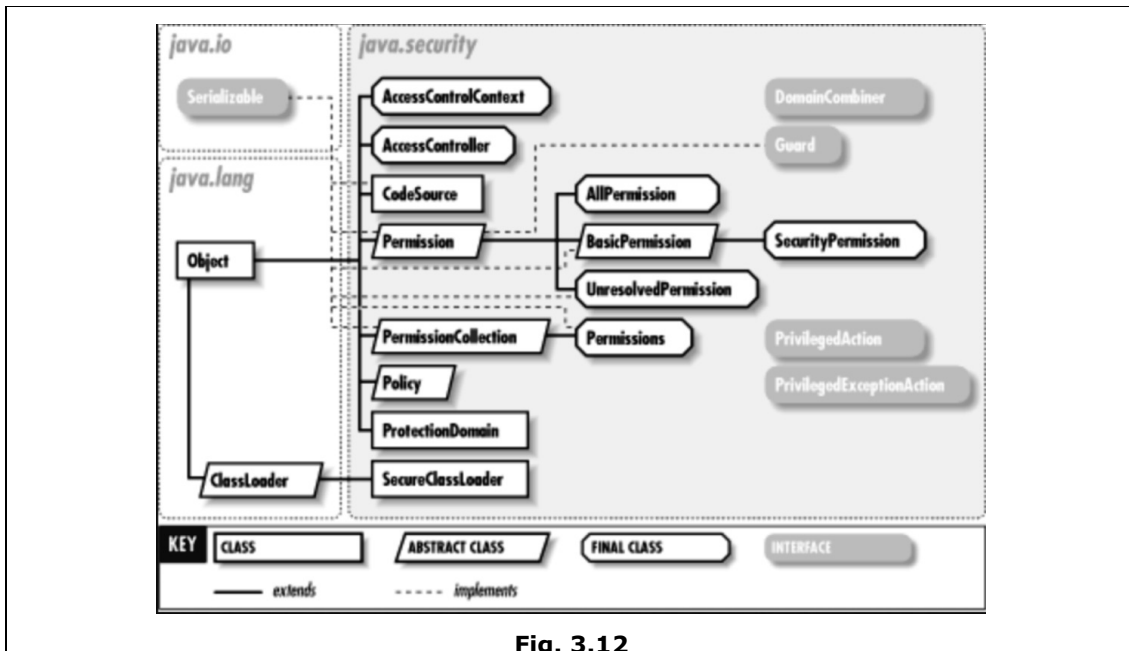


Fig. 3.12

- They also rely on an infrastructure of related classes, such as `SecureRandom` for producing cryptographic-strength pseudo-random numbers, `KeyPairGenerator` for generating pairs of public and private keys, and `KeyStore` for managing a collection of keys and certificates. (This package defines a `Certificate` interface, but it is deprecated; see the `java.security.cert` package for the preferred `Certificate` class.)
- Fig. 3.13 shows the authentication classes of `java.security`.
- The `CodeSource` class unites the authentication classes with the access control classes. It represents the source of a Java class as a URL and a set of `java.security.cert.Certificate` objects that contain the digital signatures of the code. The `AccessController` and `Policy` classes look at the `CodeSource` of a class when making access control decisions.
- All the cryptographic-authentication features of this package are provider-based, which means they are implemented by security provider modules that can be plugged easily into any Java 1.2 (or later) installation. Thus, in addition to defining a security API, this package also defines a Service Provider Interface (SPI).
- Various classes with names that end in "Spi" are part of this SPI. Security provider implementations must subclass these Spi classes, but applications never need to use them. Each security provider is represented by a `Provider` class, and the `Security` class allows new providers to be dynamically installed.

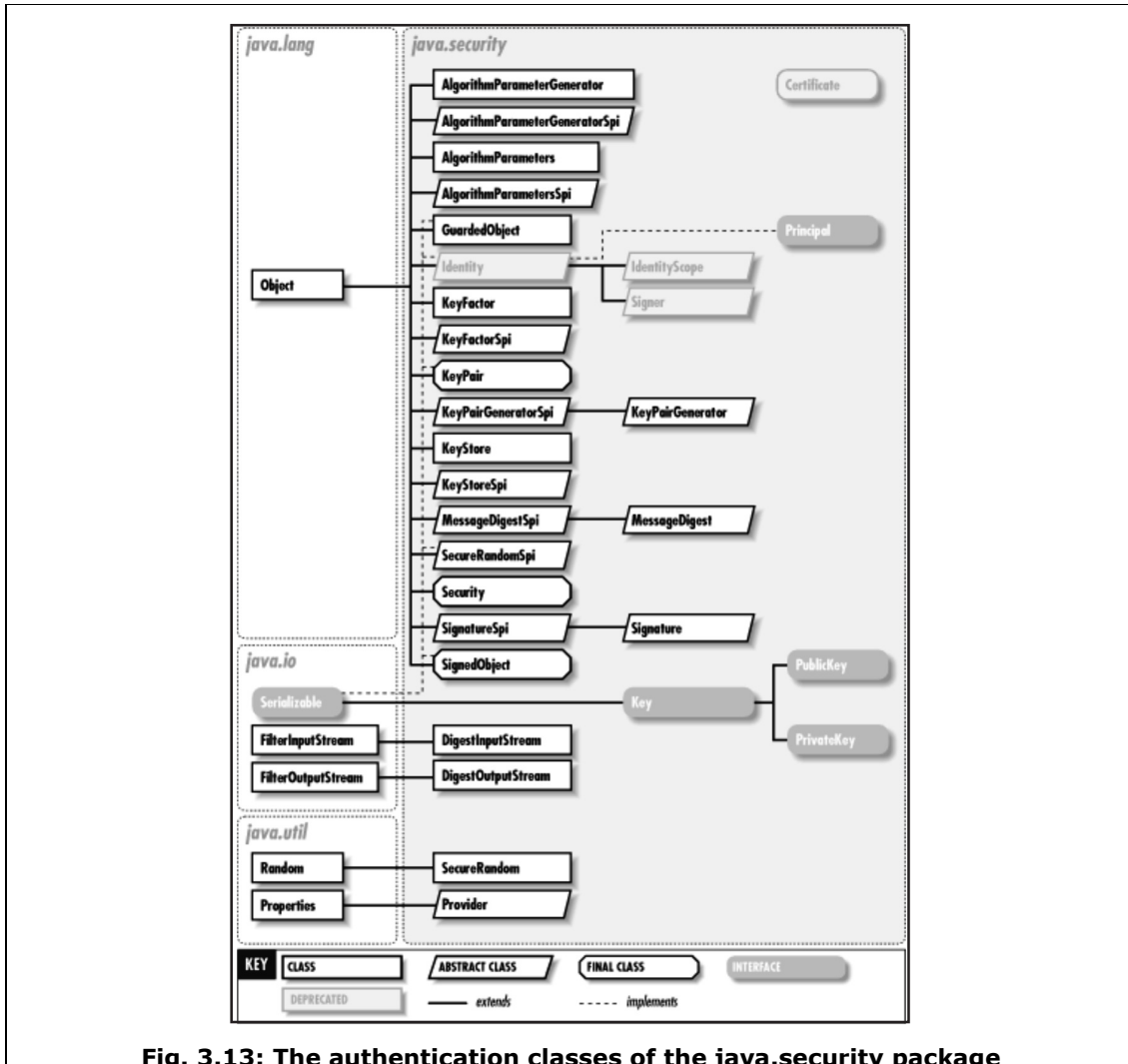


Fig. 3.13: The authentication classes of the java.security package

- The `java.security` package contains several useful utility classes. For example, `DigestInputStream` and `DigestOutputStream` make it easy to compute message digests. `GuardedObject` provides customizable access control for an individual object.
- `SignedObject` protects the integrity of an arbitrary Java object by attaching a digital signature, making it easy to detect any tampering with the object. Although the `java.security` package contains cryptographic classes for authentication, it does not contain classes for encryption or decryption.
- U.S. export control laws prevent Sun from including encryption and decryption functionality in the core Java platform. Instead, this functionality is part of the Java Cryptography Extension, or JCE. The JCE builds upon the cryptographic infrastructure of `java.security`.

3.6.2 Permission Classes

- The permission classes represent access to system resources. The `java.security.Permission` class is an abstract class and is subclassed, as appropriate, to represent specific accesses.

- As an example of a permission, the following code can be used to produce a permission to read the file named "abc" in the /tmp directory:

```
perm = new java.io.FilePermission("/tmp/abc", "read");
```

- New permissions are subclassed either from the `Permission` class or one of its subclasses, such as `java.security.BasicPermission`.
- Subclassed permissions (other than `BasicPermission`) generally belong to their own packages. Thus, `FilePermission` is found in the `java.io` package.
- A crucial abstract method that needs to be implemented for each new class of permission is the `implies` method. Basically, "a implies b" means that if one is granted permission "a", one is naturally granted permission "b". This is important when making access control decisions.
- Associated with the abstract class `java.security.Permission` are the abstract class named `java.security.PermissionCollection` and the final class `java.security.Permissions`.
- Class `java.security.PermissionCollection` represents a collection (i.e., a set that allows duplicates) of `Permission` objects for a single category (such as file permissions), for ease of grouping.
- In cases where permissions can be added to the `PermissionCollection` object in any order, such as for file permissions, it is crucial that the `PermissionCollection` object ensure that the correct semantics are followed when the `implies` function is called.
- Class `java.security.Permissions` represents a collection of collections of `Permission` objects, or in other words, a super collection of heterogeneous permissions.
- Applications are free to add new categories of permissions that the system supports. How to add such application-specific permissions is discussed later in this section.
- Now we describe the syntax and semantics of all built-in permissions.
 1. **java.security.Permission:** This abstract class is the ancestor of all permissions. It defines the essential functionalities required for all permissions.

Each permission instance is typically generated by passing one or more string parameters to the constructor. In a common case with two parameters, the first parameter is usually "the name of the target" (such as the name of a file for which the permission is aimed), and the second parameter is the action (such as "read" action on a file). Generally, a set of actions can be specified together as a comma-separated composite string.

2. **java.security.PermissionCollection:** This class holds a homogeneous collection of permissions. In other words, each instance of the class holds only permissions of the same type.
3. **java.security.Permissions:** This class is designed to hold a heterogeneous collection of permissions. Basically, it is a collection of java.security.PermissionCollection objects.
4. **java.security.UnresolvedPermission:** The internal state of a security policy is normally expressed by the permission objects that are associated with each code source. Given the dynamic nature of Java technology, however, it is possible that when the policy is initialized the actual code that implements a particular permission class has not yet been loaded and defined in the Java application environment.

The UnresolvedPermission class is used to hold such "unresolved" permissions. Similarly, the class java.security.UnresolvedPermissionCollection stores a collection of UnresolvedPermission permissions.

5. **java.io.FilePermission:** The targets for this class can be specified in the following ways, where directory and file names are strings that cannot contain white spaces.

```
file
directory (same as directory/)
directory/file
directory/* (all files in this directory)
* (all files in the current directory)
directory/- (all files in the file system under this directory)
```

The actions are: **read**, **write**, **delete**, and **execute**. Therefore, the following are valid code samples for creating file permissions:

```
import java.io.FilePermission;
FilePermission p = new FilePermission("myfile", "read,write");
FilePermission p = new FilePermission("/home/gong/", "read");
FilePermission p = new FilePermission("/tmp/mytmp", "read,delete");
FilePermission p = new FilePermission("/bin/*", "execute");
FilePermission p = new FilePermission("*", "read");
FilePermission p = new FilePermission("/-", "read,execute");
FilePermission p = new FilePermission("-", "read,execute");
FilePermission p = new FilePermission("<<ALL FILES>>", "read");
```

- 6. java.net.SocketPermission:** This class represents access to a network via sockets. The target for this class can be given as "hostname:port_range", where hostname can be given in the following ways:

```
hostname (a single host)
IP address (a single host)
localhost (the local machine)
"" (equivalent to "localhost")
hostname.domain (a single host within the domain)
hostname.subdomain.domain
*.domain (all hosts in the domain)
*.subdomain.domain
* (all hosts)
```

That is, the host is expressed as a DNS name, as a numerical IP address, as "localhost" (for the local machine) or as "" (which is equivalent to specifying "localhost").

The port_range can be given as follows:

```
N (a single port)
N- (all ports numbered N and above)
-N (all ports numbered N and below)
N1-N2 (all ports between N1 and N2, inclusive)
```

Here, N, N1, and N2 are non-negative integers ranging from 0 to 65535 ($2^{16}-1$).

Below are some examples of socket permissions.

```
import java.net.SocketPermission;
SocketPermission p = new SocketPermission
    ("java.example.com", "accept");
p = new SocketPermission("192.0.2.99", "accept");
p = new SocketPermission("*.com", "connect");
p = new SocketPermission("*.example.com:80", "accept");
p = new SocketPermission("*.example.com:-1023", "accept");
p = new SocketPermission("*.example.com:1024-", "connect");
p = new SocketPermission("java.example.com:8000-9000",
    "connect, accept");
p = new SocketPermission("localhost:1024-",
    "accept, connect, listen");
```

- 7. java.security.BasicPermission:** The BasicPermission class extends the Permission class. It can be used as the base class for permissions that want to follow the same naming convention as BasicPermission.

The name for a BasicPermission is the name of the given permission (for example, "exitVM", "setFactory", "queuePrintJob", etc).

Some of the BasicPermission subclasses are java.lang.RuntimePermission, java.security.SecurityPermission, java.util.PropertyPermission, and java.net.NetPermission.

- 8. java.util.PropertyPermission:** The targets for this class are basically the names of Java properties as set in various property files.

This is one of the BasicPermission subclasses that implements actions on top of BasicPermission. The actions are read and write. Their meaning is defined as: "read" permission allows the getProperty method in java.lang.System to be called to get the property value, and "write" permission allows the setProperty method to be called to set the property value.

- 9. java.lang.RuntimePermission:** The target for a RuntimePermission can be represented by any string, and there is no action associated with the targets. For example, RuntimePermission("exitVM") denotes the permission to exit the Java Virtual Machine.

The target names are:

```
createClassLoader
getClassLoader
setContextClassLoader
setSecurityManager
createSecurityManager
exitVM
setFactory
setIO
modifyThread
stopThread
modifyThreadGroup
getProtectionDomain
readFileDescriptor
writeFileDescriptor
loadLibrary.{library name}
accessClassInPackage.{package name}
defineClassInPackage.{package name}
accessDeclaredMembers.{class name}
queuePrintJob
```

- 10. java.awt.AWTPermission:** This is in the same spirit as the RuntimePermission; it's a permission without actions. The targets for this class are:

```
accessClipboard
accessEventQueue
listenToAllAWTEvents
showWindowWithoutWarningBanner
```

- 11. java.net.NetPermission:** This class contains the following targets and no actions:

```
requestPasswordAuthentication
setDefaultAuthenticator
specifyStreamHandler
```

- 12. java.lang.reflect.ReflectPermission:** This is the Permission class for reflective operations. A ReflectPermission is a named permission (like RuntimePermission) and has no actions. The only name currently defined is

```
suppressAccessChecks
```

which allows suppressing the standard Java programming language access checks -- for public, default (package) access, protected, and private members -- performed by reflected objects at their point of use.

- 13. java.io.SerializablePermission:** This class contains the following targets and no actions:

```
enableSubclassImplementation
enableSubstitution
```

- 14. java.security.SecurityPermission:** SecurityPermissions control access to security-related objects, such as Security, Policy, Provider, Signer, and Identity objects. This class contains the following targets and no actions:

```
getPolicy
setPolicy
getProperty.{key}
setProperty.{key}
insertProvider.{provider name}
removeProvider.{provider name}
setSystemScope
setIdentityPublicKey
setIdentityInfo
printIdentity
addIdentityCertificate
removeIdentityCertificate
clearProviderProperties.{provider name}
putProviderProperty.{provider name}
removeProviderProperty.{provider name}
getSignerPrivateKey
setSignerKeyPair
```

15. java.security.AllPermission: This permission implies all permissions. It is introduced to simplify the work of system administrators who might need to perform multiple tasks that require all (or numerous) permissions. It would be inconvenient to require the security policy to iterate through all permissions. Note that AllPermission also implies new permissions that are defined in the future.

16. javax.security.auth.AuthPermission: AuthPermission handles authentication permissions and authentication-related object such as Subject, SubjectDomainCombiner, LoginContext, and Configuration. This class contains the following targets and no actions:

```
doAs
doAsPrivileged
getSubject
getSubjectFromDomainCombiner
setReadOnly
modifyPrincipals
modifyPublicCredentials
modifyPrivateCredentials
refreshCredential
destroyCredential
createLoginContext.{name}
getLoginConfiguration
setLoginConfiguration
refreshLoginConfiguration
```

3.6.3 Policy Class

- Java security policy provides java.security.policy class.
- The system security policy for a Java application environment, specifying which permissions are available for code from various sources, is represented by a Policy object. More specifically, it is represented by a Policy subclass providing an implementation of the abstract methods in the Policy class.
- In order for an applet (or an application running under a SecurityManager) to be allowed to perform secured actions, such as reading or writing a file, the applet (or application) must be granted permission for that particular action.
- There could be multiple instances of the Policy object, although only one is "in effect" at any time. The currently-installed Policy object can be obtained by calling the getPolicy method, and it can be changed by a call to the setPolicy method (by code with permission to reset the Policy).

Policy File Format:

- In the Policy reference implementation, the policy can be specified within one or more policy configuration files. The configuration files indicate what permissions are allowed for code from specified code sources.
- A policy configuration file essentially contains a list of entries. It may contain a "keystore" entry, and contains zero or more "grant" entries.
- A keystore is a database of private keys and their associated digital certificates such as X.509 certificate chains authenticating the corresponding public keys. The keytool utility is used to create and administer keystores.
- The keystore specified in a policy configuration file is used to look up the public keys of the signers specified in the grant entries of the file.
- At this time, there can be only one keystore entry in the policy file (others after the first one are ignored), and it can appear anywhere outside the file's grant entries.
- It has the following syntax:

```
keystore "some_keystore_url", "keystore_type";
```

Here, "some_keystore_url" specifies the URL location of the keystore, and "keystore_type" specifies the keystore type.

- The URL is relative to the policy file location. Thus, if the policy file is specified in the security properties file as:

```
policy.url.1=http://foo.bar.example.com/blah/some.policy
```

and that policy file has an entry:

```
keystore ".keystore";
```

then the keystore will be loaded from:

```
http://foo.bar.example.com/blah/.keystore
```

The URL can also be absolute.

- A keystore type defines the storage and data format of the keystore information, and the algorithms used to protect private keys in the keystore and the integrity of the keystore itself.
- Each grant entry in a policy file essentially consists of a CodeSource and its permissions.
- Each grant entry in the policy file is of the following format, where the leading "grant" is a reserved word that signifies the beginning of a new entry and optional items appear in brackets. Within each entry, a leading "permission" is another reserved word that marks the beginning of a new permission in the

entry. Each grant entry grants a set of permissions to a specified code source and principals.

```
grant [SignedBy "signer_names"] [, CodeBase "URL"]
    [, Principal [principal_class_name] "principal_name"]
    [, Principal [principal_class_name] "principal_name"] ... {
    permission permission_class_name [ "target_name" ]
        [, "action"] [, SignedBy "signer_names"];
    permission ...
};
```

Programs:**1. Write a program to create a server for sending some strings to the client.**

```
//A server that sends message to client
import java.net.*;
import java.io.*;
class Server1
{ public static void main(String args[]) throws IOException
{ //Create Server side socket
ServerSocket ss = new ServerSocket (777);
//make this socket accept client connection
Socket s = ss.accept ();
System.out.println ("A connection established...");
//attach OutputStream to socket
OutputStream obj = s.getOutputStream ();
//to send data to Socket
PrintStream ps = new PrintStream (obj);
//now send the data
String str = "Hello Client";
ps.println (str);
ps.println ("Bye");
//close connection
s.close ();
ss.close ();
ps.close ();
}
}
```

Note: Do not run this program till client is also created.

2. Write a program to create a client which accepts all the strings sent by the server.

```
// a client that accepts data from server
import java.util.*;
import java.io.*;
import java.net.*;
class Client1
{ public static void main(String args[]) throws IOException
{ //Create client socket
Socket s = new Socket ("localhost", 777);
//attach input stream to Socket
InputStream obj = s.getInputStream ();
//to receive data from socket
BufferedReader br = new BufferedReader (new InputStreamReader
(obj));
//read data coming from server
String str;
while ((str = br.readLine() ) != null )
System.out.println (str);
//close connection
s.close ();
br.close ();
}
}
```

Note: After compiling Server1.java and Client1.java, run these programs in two separate dos windows:

3. Write a program to send number from client and server will reply number as even or odd.

```
For client
import java.net.*;
import java.io.*;
public class EvenOddClient
{
    public static void main(String[] args) throws
    SecurityException,IOException
    {
        Socket s=new Socket("localhost",1000);
        BufferedReader br=new BufferedReader(new
        InputStreamReader(System.in));
```

```
System.out.println("Enter no:");
int n=Integer.parseInt(br.readLine());
OutputStream o=s.getOutputStream();
PrintStream ps=new PrintStream(o);
ps.println(n);
BufferedReader br1 =new BufferedReader(new
InputStreamReader(s.getInputStream()));
int c=br1.read();
while(c!=-1)
    {
        System.out.print((char)c);
        c=br1.read();
    }
}
```

For server:

```
import java.net.*;
import java.io.*;
import java.util.*;
public class EvenOddServer
{
    public static void main(String[] args) throws
        SecurityException,IOException
    {
        ServerSocket ss=new ServerSocket(1000);
        System.out.println("Searching..");
        Socket s=ss.accept();
        BufferedReader br =new BufferedReader(new
            InputStreamReader(s.getInputStream()));
        int n=Integer.parseInt(br.readLine());
        OutputStream o=s.getOutputStream();
        PrintStream ps=new PrintStream(o);
        System.out.println("Checking");
        if(n%2==0)
            ps.println("No is Even");
        else
            ps.println("No is Odd");
    }
}
```

4. Write a program for user name.

Program For Server:

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

public class UsernameServer
{
public static void main(String args[])throws Exception
{
ServerSocket ss = new ServerSocket(1000);
System.out.println("Searching...");
System.out.println();

Socket s= ss.accept( );
System.out.println("Connected to
                    Host:"+s.getInetAddress().getHostName());
System.out.println("Connected to InetAdress:"+s.getInetAddress());
System.out.println("Connected to Port:"+s.getPort());
System.out.println();

BufferedReader br = new BufferedReader(new
                    InputStreamReader(s.getInputStream( )));
String username=br.readLine( );
System.out.println("Communicating with "+username+"...");

OutputStream o=s.getOutputStream( );
PrintStream ps= new PrintStream(o);
ps.println("Hello "+username+"!!");
}
}
```

Program for Client:

```
import java.net.*;
import java.io.*;
public class UsernameClient
{
public static void main(String args[]) throws Exception
{
Socket s = new Socket("suchitra",1000);
System.out.println("Connected to
                    Host:"+s.getInetAddress().getHostName());
```

```
System.out.println("Connected to InetAddress:"+s.getInetAddress());
System.out.println("Connected to Port:"+s.getPort());
System.out.println();
BufferedReader br1=new BufferedReader(new
                                   InputStreamReader(System.in));

System.out.print("Enter Username:");
String myname=br1.readLine();
System.out.println();
OutputStream o=s.getOutputStream( );
PrintStream ps = new PrintStream(o);
ps.println(myname);
BufferedReader br2 = new BufferedReader(new
InputStreamReader(s.getInputStream( )));
System.out.println(br2.readLine( ));
}
}
```

Program 5:

```
import java.net.*;
import java.io.*;
public class GreetingClient
{
public static void main(String [] args)
{
String serverName = "localhost";
int port = 999;
try
{
System.out.println("Connecting to" + serverName + "on port" + port);
Socket client = new Socket(serverName, port);
System.out.println("Just connected to"+
                   client.getRemoteSocketAddress());

OutputStream outToServer = client.getOutputStream();
DataOutputStream out = new DataOutputStream(outToServer);
out.writeUTF("Hello from " + client.getLocalSocketAddress());
```

```
InputStream inFromServer = client.getInputStream();
DataInputStream in = new DataInputStream(inFromServer);
System.out.println("Server says " + in.readUTF());
client.close();
} catch (IOException e)
{
    e.printStackTrace();
}
}
}
import java.net.*;
import java.io.*;
public class GreetingServer extends Thread
{
    private ServerSocket serverSocket;
    public GreetingServer(int port) throws IOException
    {
        serverSocket = new ServerSocket(port);
    }
    public void run()
    {
        while(true)
        {
            try
            {
                System.out.println("Waiting for client on port " +
                    serverSocket.getLocalPort() + "...");
                Socket server = serverSocket.accept();
                System.out.println("Just connected to " +
                    server.getRemoteSocketAddress());

                DataInputStream in = new
                DataInputStream(server.getInputStream());
                System.out.println(in.readUTF());
```

```
        DataOutputStream out = new
        DataOutputStream(server.getOutputStream());
        out.writeUTF("Thank you for connecting to " +
                    server.getLocalSocketAddress() + "\nGoodbye!");
        server.close();
    }
    catch(SocketTimeoutException s)
    {
        System.out.println("Socket timed out!");
        break;
    }
    catch(IOException e)
    {
        e.printStackTrace();
        break;
    }
}
}
}

public static void main(String [] args)
{
    try
    {
        Thread t = new GreetingServer(999);
        t.start();
    }
    catch(IOException e)
    {
        e.printStackTrace();
    }
}
}
```

Important Points

- Java programming environment provides simple yet robust techniques that permit Java applications to communicate across networks and share valuable resources.
- A network socket is a lot like electric socket, various plugs around the network have a standard way of delivering their pay load.
- The Internet Protocol (IP) is a Layer 3 protocol, (network layer) that is used to transmit data packets over the Internet.
- The IP address is a four-byte (32-bit) address, which is usually expressed in dotted decimal format.
- The Transmission Control Protocol (TCP) is a Layer 4 protocol, (transport layer) that provides guaranteed delivery and ordering of bytes.
- 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).
- A server is an entity that has some resources that can be shared.
- A client is simply another entity that wants to gain access to those resources.
- Proxy server is like a machine that acts as a proxy for application protocol.
- A proxy server has the additional ability to filter clients request or cache the results of those requests for future use.
- All Internet address are consist of 32 bit values and they are often referred to as a sequence of four numbers between 0 and 25.5 separated by periods [.,].
- The package `java.net` provides different classes for implementing networking applications.
- `URL` and `URLConnection` classes provides a quick and easy way to access content using uniform resource locators.
- The `ContentHandler` and `URLStreamHandler` are the abstract classes used to extend the capabilities of `URL` class.
- `SocketImplFactory` interface defines a method that returns a `SocketImpl` instance appropriate to the underlying operating system.
- The `URLStreamHandler` class uses this interface to obtain `ContentHandler` objects for different content types.
- 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`.
- Uniform Resource Locator (URL) is the global address of documents and other resources on the World Wide Web.

- The abstract class `URLConnection` is the superclass of all classes that represent a communications link between the application and a URL.
- The Transmission Control Protocol (TCP) is a stream-based method of network communication that is far different from any discussed previously.
- TCP uses a lower-level communications protocol, the Internet Protocol (IP), to establish the connection between machines.
- The client/server paradigm divides software into two categories, clients and servers.
- A client is a software that initiates a connection and sends requests, whereas a server is a software that listens for connections and processes requests.
- Network clients initiate connections and usually take charge of network transactions.
- 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.
- Java offers good support for TCP sockets, in the form of two socket classes, `java.net.Socket` and `java.net.ServerSocket`.
- The `Socket` class represents client sockets and is a communication channel between two TCP communications ports belonging to one or two machines.
- Sockets can perform a variety of tasks, such as reading information, sending data, closing a connection and setting socket options.
- The Java networking API provides a consistent, platform neutral interface that allows client applications to connect to remote services.
- A socket is created, an input stream is obtained and timeouts are enabled in the rare event that a server as simple as daytime fails during a connection.
- The `ServerSocket` class does not really do that much, other than accept connections and act as a factory for `Socket` objects that model the connection between client and server.
- The most important function of a server socket is to accept client sockets.
- UDP is a connectionless transport protocol, meaning that it does not guarantee either packet delivery or that packets arrive in sequential order.
- UDP communication can be more efficient than guaranteed-delivery data streams.
- The `java.security` package contains the classes and interfaces that implement the Java security architecture.
- The permission classes represent access to system resources. The `java.security.Permission` class is an abstract class and is subclassed, as appropriate, to represent specific accesses.
- Class `java.security.Permissions` represents a collection of collections of `Permission` objects, or in other words, a super collection of heterogeneous permissions.
- Java security policy provides `java.security.policy` class.

Practice Questions

1. What is networking?
2. What are the advantages of networking?
3. What is Protocol? List out at least two protocols.
4. Explain with suitable diagram the HTTP communication between Web server and web client.
5. What is Client Server Model?
6. What is Internet?
7. Define the following terms:
 - (a) Web Browser
 - (b) Web server
 - (c) Protocol
 - (d) HTTP
 - (e) Home Page
 - (f) Firewall
 - (g) Proxy server.
8. Write short note on URL Programming.
9. Explain use of Socket class.
10. Explain TCP/IP client sockets with suitable example.
11. Explain TCP/IP server sockets with suitable example.
12. What is the difference between TCP/IP and UDP?
13. Explain policy class in detail.
14. Enlist various permission classes.
15. Explain the use of ServerSocket class.
16. Explain url rewriting in detail.
17. Explain the use of InetAddress class.
18. Write a program to retrieve hostname using methods in InetAddress class.
19. Write a program that demonstrates TCP/IP based communication between client and server.
20. Write a program that demonstrates UDP based communication between client and server.
21. Write a program to demonstrate use of URL and URL Connection class for communication.
22. Explain java security in detail.

