UNIT - 2

Networking

JAVA

2.1 Networking Basics
2.2 Java.net Package
2.3 Connection Oriented Transmission – Stream Socket Class
2.4 Creating a Socket to Remote Host On a Port (Creating TCP Client and Server)
2.5 Simple Socket Program Example

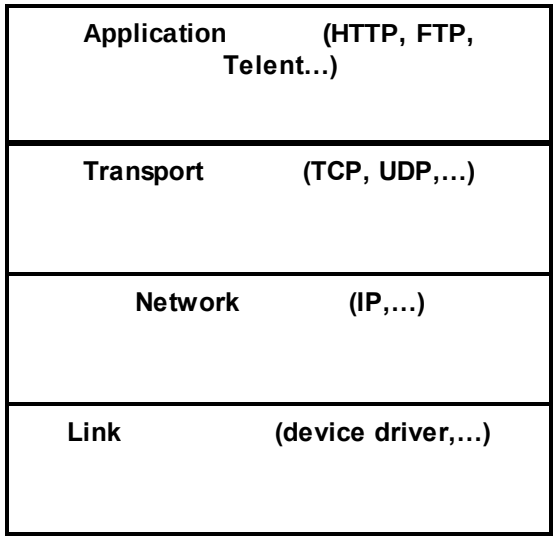
Introduction

- The term network programming refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.
- Java Networking is a concept of connecting two or more computing devices together so that user can share resources.
- Java socket programming provides facility to share data between different computing devices.

2.1 Networking Basics

Java

Computers running on the Internet communicate to each other using either the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP), as this diagram illustrates:



When user writes Java programs that communicate over the network, they are programming at the application layer. Typically, user do not need to concern with the TCP and UDP layers. Instead, they can use the classes in the java.net package. These classes provide system-independent network communication. However, to decide which Java classes user's programs should use, they do need to understand how TCP and UDP differ.

A) TCP (Transmission Control Protocol):

TCP (Transmission Control Protocol) is a connection-based protocol that provides a reliable flow of data between two computers.

When two applications want to communicate to each other reliably, they establish a connection and send data back and forth over that connection. This is analogous to making a telephone call. If user want to speak to XYZ person in London, a connection is established, when user dials phone number and XYZ answers. User sends data back and forth over the connection by speaking to one another over the phone lines. Like the phone company, TCP guarantees that data sent from one end of the connection actually gets to the other end and in the same order, it was sent. Otherwise, an error is reported.

TCP provides a point-to-point channel for applications that require reliable communications. The Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), and Telnet are all examples of applications that require a reliable communication channel. The order in which the data is sent and received over the network is critical to the success of these applications. When HTTP is used to read from a URL, the data must be received in the order in which it was sent. Otherwise, user ends up with a jumbled HTML file, a corrupt zip file, or some other invalid information.

B) UDP (User Datagram Protocol):

UDP (User Datagram Protocol) 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.

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

For many applications, the guarantee of reliability is critical to the success of the transfer of information from one end of the connection to the other. However, other forms of communication do not require such strict standards. In fact, they may be slowed down by the extra overhead or the reliable connection may invalidate the service altogether.

Example:

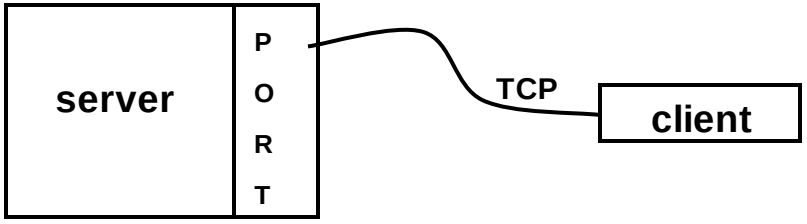
Consider a clock server that sends the current time to its client when requested to do so. If the client misses a packet, it does not really make sense to resend it because the time will be incorrect when the client receives it on the second try. If the client makes two requests and receives packets from the server out of order, it does not really matter because the client can figure out that the packets are out of order and make another request. The reliability of TCP is unnecessary in this instance because it causes performance degradation and may hinder the usefulness of the service.

C) Understanding Ports:

The TCP and UDP protocols use ports to map incoming data to a particular process running on a computer.

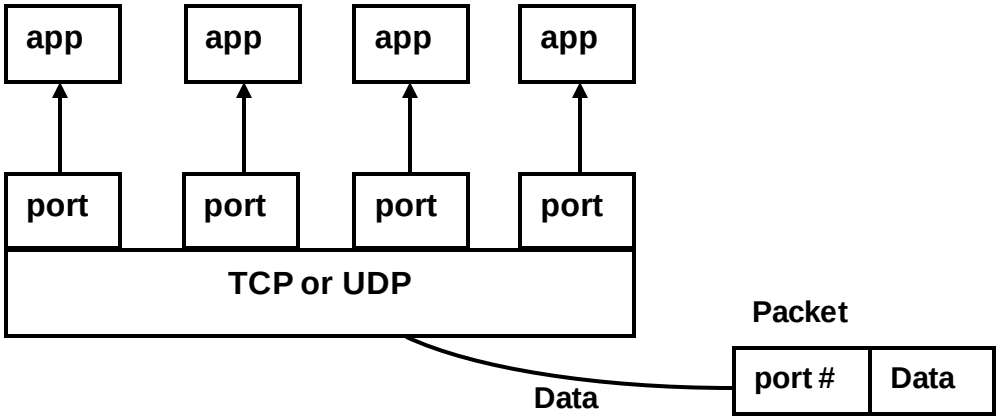
Generally speaking, a computer has a single physical connection to the network. All data destined for a particular computer arrives through that connection. However, the data may be intended for different applications running on the computer. So how does the computer know to which application to forward the data? Through the use of ports.

Data transmitted over the Internet is accompanied by addressing information that identifies the computer and the port for which it is destined. The computer is identified by its 32-bit IP address, which IP uses to deliver data to the right computer on the network. Ports are identified by a 16-bit number, which TCP and UDP use to deliver the data to the right application.



C) Understanding Ports:

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



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 this figure:

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.

D) Networking Classes in the JDK:

Through the classes in java.net, Java programs can use TCP or UDP to communicate over the Internet. The URL, URLConnection, Socket, and ServerSocket classes all use TCP to communicate over the network. The DatagramPacket, DatagramSocket and MulticastSocket classes are for use with UDP.

E) Sockets:

Sockets allow communication between two different processes on the same or different machines. To be more precise, it is a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O actions are done by writing or reading to a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else. To a programmer a socket looks and behaves much like a low-level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes.

a) Use of Socket:

A Unix Socket is used in a client server application framework. A server is a process, which does some function on request from a client. Most of the application level protocols like FTP, SMTP and POP3 make use of Sockets to establish connection between client and server and then for exchanging data.

E) Sockets:

Socket Types:

There are four types of sockets available to the users.

1) **Stream Sockets:**

Delivery in a networked environment is guaranteed. If user send through the stream socket three items "A,B,C", they will arrive in the same order - "A,B,C". These sockets use TCP (Transmission Control Protocol) for data transmission. If delivery is impossible, the sender receives an error indicator. Data records do not have any boundaries.

2) **Datagram Sockets:**

Delivery in a networked environment is not guaranteed. They are connectionless because user does not need to have an open connection as in Stream Sockets - user build a packet with the destination information and send it out. They use UDP (User Datagram Protocol).

3) **Raw Sockets:**

It provides users access to the underlying communication protocols, which support socket abstractions. These sockets are normally datagram oriented, though their exact characteristics are dependent on the interface provided by the protocol.

4) **Sequenced Packet Sockets:**

They are similar to a stream socket, with the exception that record boundaries are preserved. This interface is provided only as part of the Network Systems (NS) socket abstraction, and is very important in most serious NS applications.

F) Port:

When a client process wants to connect a server, the client must have a way of identifying the server that it wants to connect. So if the client knows the 32-bit Internet address of the host on which the server resides it can contact that host. But how does the client identify the particular server process running on that host? To resolve the problem of identifying a particular server process running on a host, both TCP and UDP have defined a group of well-known ports.

a) Ports and Services:

One can find most updated list of internet ports and associated service at IANA - TCP/IP Port Assignments.

b) Port and Service Functions:

Unix provides following functions to fetch service name from the /etc/services file.
struct servent *getservbyname(char *name, char *proto):
This call takes service name and protocol name and returns corresponding port number for that service.

c) How Port Numbers Work:

Port numbers are associated with network addresses. For example, in TCP/IP networking, both Wireless Network, Computer Networks, Network Traffic, Network Management, Network Configuration

F) Port:

d) Internet Addressing:

Users must have noticed that while mentioning any web address, it is mentioned as www and then has some name followed by dot and then dot again and end with either com or any other code. One must have wondered why the addressing is done like that. An Internet address is made up of a code or series or letters, numbers and/or symbols by which the Internet identifies you or a location where information is stored. Through the use of this addresses, people can send you email, look at your Web site and send or receive tiles and documents.

1) Domain Name:

The names of the various computers are designed by Internet Engineering Task Force (IETF). This addressing scheme assigns names and numbers to identify the computers on the Internet. These names are called domain names and numbers are called IP addresses. Every computer system on the Internet has both a domain name and an IP address. This address contains four Domain Names are easier to remember than a combination of 4 level digital numbers of an IP address. Each set of numbers can have the values ranging from 1 to 255. The numbers between the dots is called an octet. An IP address is organized from left to right where the leftmost represents the largest network organization and the rightmost octet describes the actual network connection These combined parts are always unique for the given host.

F) Port:

d) Internet Addressing:

2) Address Types:

Some IP addresses and some patterns of addresses have special meanings. IPv4 addresses in the range 224.0.0.0 to 239.255.255.255 are multicast addresses that send to several subscribed hosts at once. Java includes 10 methods for testing whether an InetAddress object meets any of these criteria:



F) Port:

d) Internet Addressing:

2) Address Types:

a) isAnyLocalAddress():

The isAnyLocalAddress() method returns true if the address is a wildcard address, false otherwise. A wildcard address matches any address of the local system. This is important if the system has multiple network interfaces, as might be the case on a system with multiple Ethernet cards or an Ethernet card and an 802.11 WiFi interface. In IPv4, the wildcard address is 0.0.0.0. In IPv6, this address is 0:0:0:0:0:0:0:0 (a.k.a. ::).

b) isLoopbackAddress():

The isLoopbackAddress() method returns true if the address is the loopback address, false otherwise. The loopback address connects to the same computer directly in the IP layer without using any physical hardware. Thus, connecting to the loopback address enables tests to bypass potentially buggy or nonexistent Ethernet, PPP, and other drivers, helping to isolate problems. Connecting to the loopback address is not the same as connecting to the system's normal IP address from the same system. In IPv4, this address is 127.0.0.1. In IPv6, this address is 0:0:0:0:0:0:0:1 (a.k.a. ::1).

F) Port:

d) Internet Addressing:

2) Address Types:

c) isLinkLocalAddress():

The isLinkLocalAddress() method returns true if the address is an IPv6 link-local address, false otherwise. This is an address used to help IPv6 networks self-configure, much like DHCP on IPv4 networks but without necessarily using a server. Routers do not forward packets addressed to a link-local address beyond the local subnet.

d) isSiteLocalAddress():

The isSiteLocalAddress() method returns true if the address is an IPv6 site-local address, false otherwise. Site-local addresses are similar to link-local addresses except that they may be forwarded by routers within a site or campus but should not be forwarded beyond that site. Site-local addresses begin with the eight bytes FEC0:0000:0000:0000.

e) isMulticastAddress():

The isMulticastAddress() method returns true if the address is a multicast address, false otherwise. Multicasting broadcasts content to all subscribed computers rather than to one particular computer. In IPv4, multicast addresses all fall in the range 224.0.0.0 to 239.255.255.255. In IPv6, they all begin with byte FF. Multicasting will be discussed in Chapter 13.

F) Port:

d) Internet Addressing:

2) Address Types:

f) isMCGlobal():

The isMCGlobal() method returns true if the address is a global multicast address, false otherwise. A global multicast address may have subscribers around the world. All multicast addresses begin with FF.

g) isMCOrgLocal():

The isMCOrgLocal() method returns true if the address is an organization-wide multicast address, false otherwise. An organization-wide multicast address may have subscribers within all the sites of a company or organization, but not outside that organization. Organization multicast addresses begin with FF08 or FF18, depending on whether the multicast address is a well known permanently assigned address or a transient address.

h) isMCSiteLocal():

The isMCSiteLocal() method returns true if the address is a site-wide multicast address, false otherwise. Packets addressed to a site-wide address will only be transmitted within their local site. Site-wide multicast addresses begin with FF05 or FF15, depending on whether the multicast address is a well known permanently assigned address or a transient address.

F) Port:

d) Internet Addressing:

2) Address Types:

i) isMCLinkLocal():

The isMCLinkLocal() method returns true if the address is a subnet-wide multicast address, false otherwise. Packets addressed to a link-local address will only be transmitted within their own subnet. Link-local multicast addresses begin with FF02 or FF12, depending on whether the multicast address is a well known permanently assigned address or a transient address.

j) isMCNodeLocal():

The isMCNodeLocal() method returns true if the address is an interface-local multicast address, false otherwise. Packets addressed to an interface-local address are not sent beyond the network interface from which they originate, not even to a different network interface on the same node. This is primarily useful for network debugging and testing. Interface-local multicast addresses begin with the two bytes FF01 or FF11, depending on whether the multicast address is a well known permanently assigned address or a transient address.

2.2 Java.net Package

Java

The java.net package provides the interfaces and classes for implementing networking applications.

A) Interfaces of java.net Package:

Following table shows the various interfaces of java.net package and their uses:

Interface	Use
ContentHandlerFactory	This interface defines a factory for content handlers.
CookiePolicy	CookiePolicy implementations decide which cookies should be accepted and which should be rejected.
CookieStore	A CookieStore object represents storage for cookie.
DatagramSocketImplFactory	This interface defines a factory for datagram socket implementations.
FileNameMap	A simple interface which provides a mechanism to map between a file name and a MIME type string.
SocketImplFactory	This interface defines a factory for socket implementations.
SocketOptions	Interface of methods to get/set socket options.
URLStreamHandlerFactory	This interface defines a factory for URL stream protocol handlers.

2.2 Java.net Package

Java

B) Classes of java.net package:

The classes in the networking package fall into three general categories:

- Web interface classes
- URL and URLConnection classes
- Raw network interface classes

Following table shows the classes of java.net package and their uses:

Class	Use
ContentHandler	Super class of the classes that turn MIME objects into corresponding Java objects
CookieHandler	A CookieHandler object provides a callback mechanism to hook up a HTTP state management policy implementation into the HTTP protocol handler.
CookieManager	CookieManager provides a concrete implementation of CookieHandler, which separates the storage of cookies from the policy surrounding accepting and rejecting cookies.
DatagramPacket	This class represents a datagram packet.
DatagramSocket	This class represents a socket for sending and receiving datagram packets.
Inet4Address	This class represents an Internet Protocol version 4 (IPv4) address.
Inet6Address	This class represents an Internet Protocol version 6 (IPv6) address.
InetAddress	This class represents an Internet Protocol (IP) address.
NetPermission	This class is for various network permissions.

2.2 Java.net Package

Java

B) Classes of java.net package:

Class	Use
PasswordAuthentication	The class PasswordAuthentication is a data holder that is used by Authenticator.
Proxy	This class represents a proxy setting, typically a type (http, socks) and a socket address.
ResponseCache	Represents implementations of URLConnection caches.
ServerSocket	This class implements server sockets.
Socket	This class implements client sockets (also called just "sockets").
URI	Represents a Uniform Resource Identifier(URI) reference.
URL	Class URL represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web.
URLConnection	The abstract class URLConnection is the superclass of all classes that represent a communications link between the application and a URL.

C) Network Protocols Supported:

The java.net package provides support for the two common network protocols:



1) TCP:

TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.

2) UDP:

UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

C) Network Protocols Supported:

a) Socket Programming:

This is most widely used concept in Networking and it has been explained in very detail. Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

Steps Occurring when Establishing a TCP Connection:

The following steps occur when establishing a TCP connection between two computers using sockets:

- The server instantiates a ServerSocket object, denoting which port number communication is to occur on.
- The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
- After the server is waiting, a client instantiates a Socket object, specifying the server name and port number to connect to.
- The constructor of the Socket class attempts to connect the client to the specified server and port number. If communication is established, the client now has a Socket object capable of communicating with the server.
- On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

C) Network Protocols Supported:

b) ServerSocket Class Methods:

The java.net.ServerSocket class is used by server applications to obtain a port and listen for client requests The ServerSocket class has four constructors:

SN	Methods with Description
1)	**public ServerSocket(int port) throws IOException** Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.
2)	**public ServerSocket(int port, int backlog) throws IOException** Similar to the previous constructor, the backlog parameter specifies how many incoming clients to store in a wait queue.
3)	**public ServerSocket(int port, int backlog, InetAddress address) throws IOException** Similar to the previous constructor, the InetAddress parameter specifies the local IP address to bind to. The InetAddress is used for servers that may have multiple IP addresses, allowing the server to specify which of its IP addresses to accept client requests on
4)	**public ServerSocket() throws IOException** Creates an unbound server socket. When using this constructor, use the bind() method when you are ready to bind the server socket

C) Network Protocols Supported:

b) ServerSocket Class Methods:

If the ServerSocket constructor does not throw an exception, it means that your application has successfully bound to the specified port and is ready for client requests. Here are some of the common methods of the ServerSocket class:

SN	Methods with Description
1)	**public int getLocalPort()** Returns the port that the server socket is listening on. This method is useful if you passed in 0 as the port number in a constructor and let the server find a port for you.
2)	**public Socket accept() throws IOException** Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely
3)	**public void setSoTimeout(int timeout)** Sets the time-out value for how long the server socket waits for a client during the accept().
4)	**public void bind(SocketAddress host, int backlog)** Binds the socket to the specified server and port in the SocketAddress object. Use this method if you instantiated the ServerSocket using the no-argument constructor.

2.3 Connection Oriented Transmission – Stream Socket Class

Java

Sockets have two major modes of operation: connection-oriented and connectionless. Connection-oriented sockets operate like a telephone; they must establish a connection and a hang up. Everything that flows between these two events arrives in the same order it was sent. Connectionless sockets operate like the mail-delivery is not guaranteed, and multiple pieces of mail may arrive in a different order than they were sent.

A) Transmission Control Protocol:

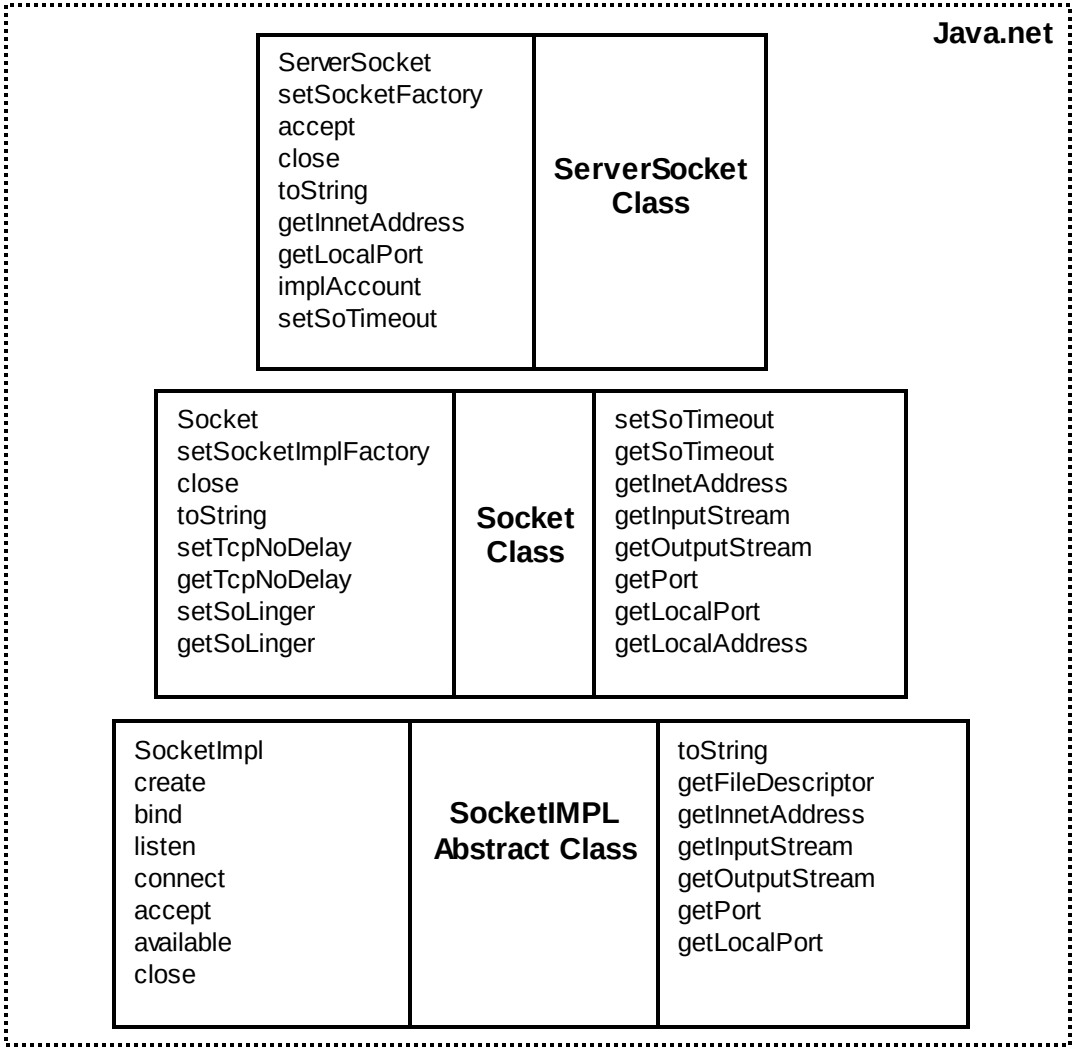
Transmission Control Protocol (TCP), which is used in a transport layer, is one of the most widely used protocols to provide connection-oriented sockets. The application hands over data to a TCP socket and the TCP takes care of streaming the data to the destination host. The TCP takes care of all issues like ordering, fragmentation, assembly, lost data detection, duplicates data transmission, etc., on both sides of the communication, which gives the impression to the applications that data is flowing like a continuous stream of bytes from the source application to the destination application. No physical connection at the hardware level exists between two hosts that use TCP sockets. It is all implemented in software. Sometimes, it is also called a virtual connection. The combination of two sockets uniquely defines a connection.

2.3 Connection Oriented Transmission – Stream Socket Class

Java

B) Stream Socket Class:

Figure below shows the three classes that implement Java's stream sockets. The Socket class is the workhorse that both clients and servers use to obtain their socket services. In addition, servers use the ServerSocket class to listen for connection requests from clients. Together, these two classes provide abstractions that hide many of the lower-level socket commands from user's applications. If user is interested in the low-level implementation details, check the SocketImpl abstract class. This is the class that Java JDK licensees must implement-and possibly extend for their respective platforms. Both the Socket and ServerSocket classes use these implementations.



2.3 Connection Oriented Transmission – Stream Socket Class

Java

B) Stream Socket Class:

Following is a short description of what these classes do:

1) **Socket:**

Socket implements a streamed socket. Programmer use one of eight Socket constructors to create a socket and passes it the destination port and address information. Once you create a socket, you invoke getInputStream and getOutputStream to return input and output stream objects that you use to read and write data.

2) **ServerSocket:**

ServerSocket implements a streamed server-side socket that listens for incoming connections from clients. You invoke the ServerSocket constructor to create a server-side socket that listens for a client connection on a local port you specify; you can also specify an optional amount of time to wait for a connection. The call to ServerSockel returns when a connection arrives. You can then invoke accept to accept the connection; it returns a Socket object that a server thread can use for future communications with this client.

3) **SocketImpl:**

SocketImpl is an abstract class that defines the socket methods. JDK licensees must implement for their platforms. The licensees may even provide specialized extensions that are suitable for different environments. For example, they may provide AppleTalk or IPX/SPX versions of sockets.

2.4 Creating a Socket to Remote Host On a Port (Creating TCP Client and Server)

Java

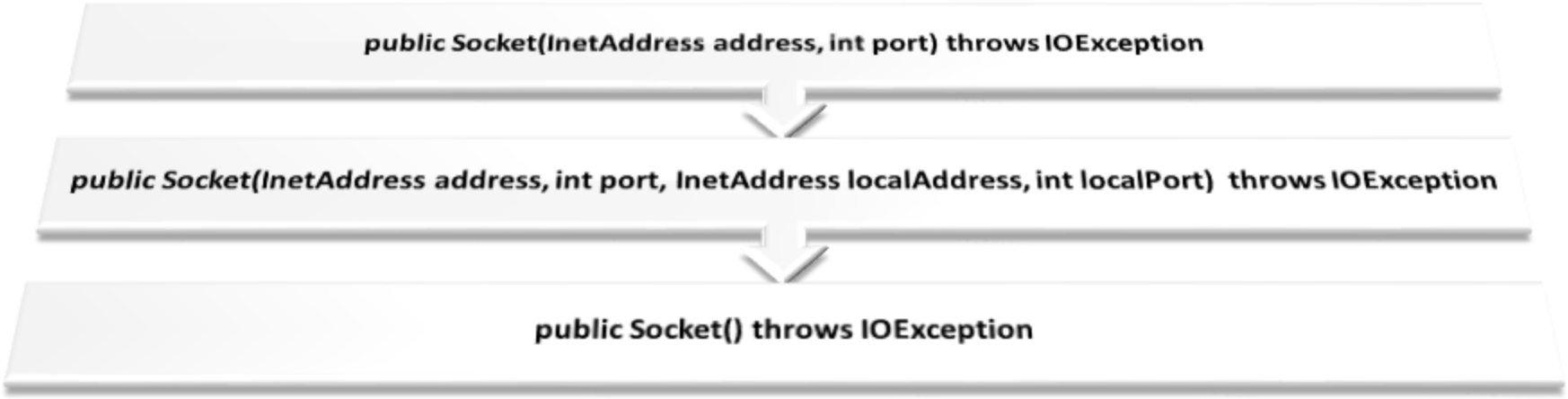
In order to communicate with a remote host the Java client must first create a Socket, which will establish the TCP connection. While establishing the connection a host name and port number must be specified. There must be a server actively listening on the specified port or the connection will fail with IOException.

A) Creating and Using Sockets:

The java.net.Socket class allows us to perform all four fundamental socket operations connecting to remote machines, sending data, receiving data and closing the connection.

a) Constructors of java.net.Socket Class:

Connection is accomplished through the constructors. Each Socket object is associated with exactly one remote host. To connect to a different host, you must create a new Socket object.



2.4 Creating a Socket to Remote Host On a Port (Creating TCP Client and Server)

Java

A) Creating and Using Sockets:

a) Constructors of java.net.Socket Class:

1) public Socket(InetAddress address, int port) throws IOException:

These constructors create a Socket and connect to the specified host and port. Host can be a host name or IP address and port must be in a range of 1-65535.

2) public Socket(InetAddress address, int port, InetAddress localAddress, int localPort) throws IOException:

These constructors create a socket and connect it to the specified remote host on the specified remote port.

3) public Socket() throws IOException:

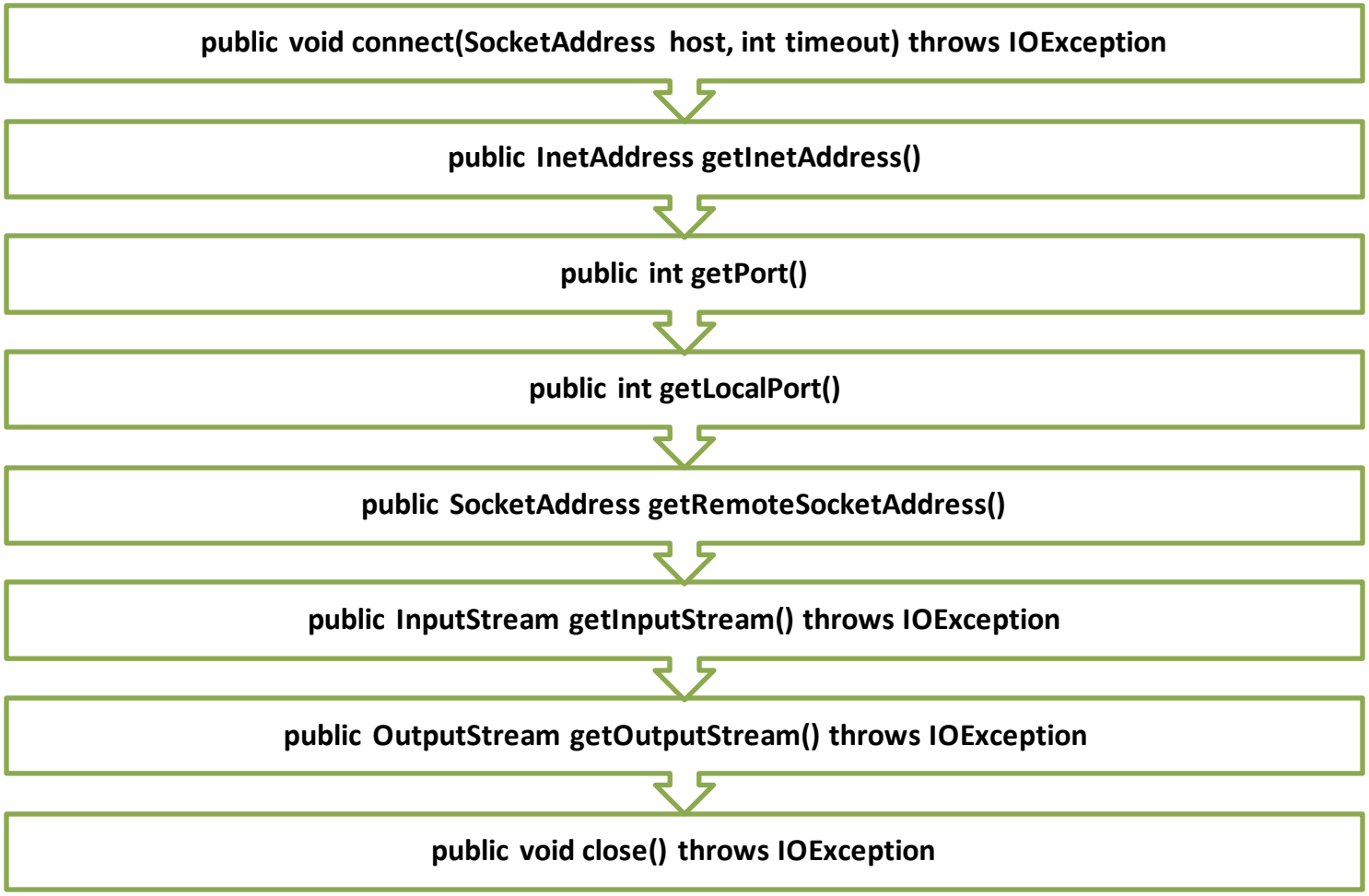
Creates an unbound socket. When using this constructor, use the connect() method.

2.4 Creating a Socket to Remote Host On a Port (Creating TCP Client and Server)

Java

A) Creating and Using Sockets:

b) Methods of Socket Class:



2.4 Creating a Socket to Remote Host On a Port (Creating TCP Client and Server)

Java

A) Creating and Using Sockets:

b) Methods of Socket Class:

1) **public void connect(SocketAddress host, int timeout) throws IOException:**
   This method connects the socket to the specified host. This method is needed only when user instantiated the Socket using the no-argument constructor.
2) **public InetAddress getInetAddress():**
   This method returns the address of the other computer that this socket is connected to.
3) **public int getPort():**
   Returns the port the socket is bound to on the remote machine.
4) **public int getLocalPort():**
   Returns the port the socket is bound to on the local machine.
5) **public SocketAddress getRemoteSocketAddress():**
   Returns the address of the remote socket.
6) **public InputStream getInputStream() throws IOException:**
   Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.
7) **public OutputStream getOutputStream() throws IOException:**
   Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket.
8) **public void close() throws IOException:**
   Closes the socket, which makes this Socket object no longer capable of connecting again to any server.

2.4 Creating a Socket to Remote Host On a Port (Creating TCP Client and Server)

Java

B) Creating TCP Servers & Clients:

TCP /IP ServerSockets:

The java.net.ServerSocket class is used by server applications to obtain a port and listen for client requests.

a) Constructors of ServerSocket:

- **public ServerSocket(int port) throws IOException**
- **public ServerSocket(int port, int backlog) throws IOException**
- **public ServerSocket(int port, int backlog, InetAddress address) throws IOException**
- **public ServerSocket() throws IOException**

2.4 Creating a Socket to Remote Host On a Port (Creating TCP Client and Server)

Java

B) Creating TCP Servers & Clients:

TCP /IP ServerSockets:

a) Constructors of ServerSocket:

1) public ServerSocket(int port) throws IOException:
Creates a server socket bound to the specified port. An exception occurs if the port is already bound by another application.

2) public ServerSocket(int port, int backlog) throws IOException:
Similar to the previous constructor, the backlog parameter specifies how many incoming clients to store in a wait queue.

3) public ServerSocket(int port, int backlog, InetAddress address) throws IOException:
Similar to the previous constructor, the InetAddress parameter specifies the local IP address to bind to.

4) public ServerSocket() throws IOException:
Creates an unbound server socket. When using this constructor, use the bind() method when you are ready to bind the server socket.

If the ServerSocket constructor does not throw an exception, it means that your application has successfully bound to the specified port and is ready for client requests.

2.4 Creating a Socket to Remote Host On a Port (Creating TCP Client and Server)

Java

B) Creating TCP Servers & Clients:

TCP /IP ServerSockets:

b) Methods of ServerSocket:

- public int getLocalPort()
- public Socket accept() throws IOException
- public void setSoTimeout(int timeout)
- public void bind(SocketAddress host, int backlog)
- public InetAddress getInetAddress()
- public void close()throws IOException

2.4 Creating a Socket to Remote Host On a Port (Creating TCP Client and Server)

Java

B) Creating TCP Servers & Clients:

TCP /IP ServerSockets:

b) Methods of ServerSocket:

1) public int getLocalPort():
Returns the port that the server socket is listening on.

2) public Socket accept() throws IOException:
Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely

3) public void setSoTimeout(int timeout):
Sets the time-out value for how long the server socket waits for a client during the accept().

4) public void bind(SocketAddress host, int backlog):
Binds the socket to the specified server and port in the SocketAddress object. Use this method if you instantiated the ServerSocket using the no-argument constructor.

5) public InetAddress getInetAddress():
Returns the local address of this server socket.

6) public void close()throws IOException:
Closes the socket. Any thread currently blocked in accept() will throw a SocketException.

2.4 Creating a Socket to Remote Host On a Port (Creating TCP Client and Server)

Java

B) Creating TCP Servers & Clients:

TCP /IP ServerSockets:

c) Client /Server Communication using TCP /IP:

A TCP stream socket server application listens on the network for incoming client request packets. A TCP stream socket client application initiates communication with the server by sending a request packet. When the server receives the request, it processes it and responds. After this initial sequenced message exchange, client and server can exchange data. The following steps occur when establishing a TCP connection between two computers using sockets:

- The server instantiates a ServerSocket object, indicating which port number communication is to occur on.
- The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
- After the server is waiting, a client instantiates a Socket object, specifying the server name and port number to connect to.
- The constructor of the Socket class attempts to connect the client to the specified server and port number. If communication is established, the client now has a Socket object capable of communicating with the server.
- On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

2.4 Creating a Socket to Remote Host On a Port (Creating TCP Client and Server)

Java

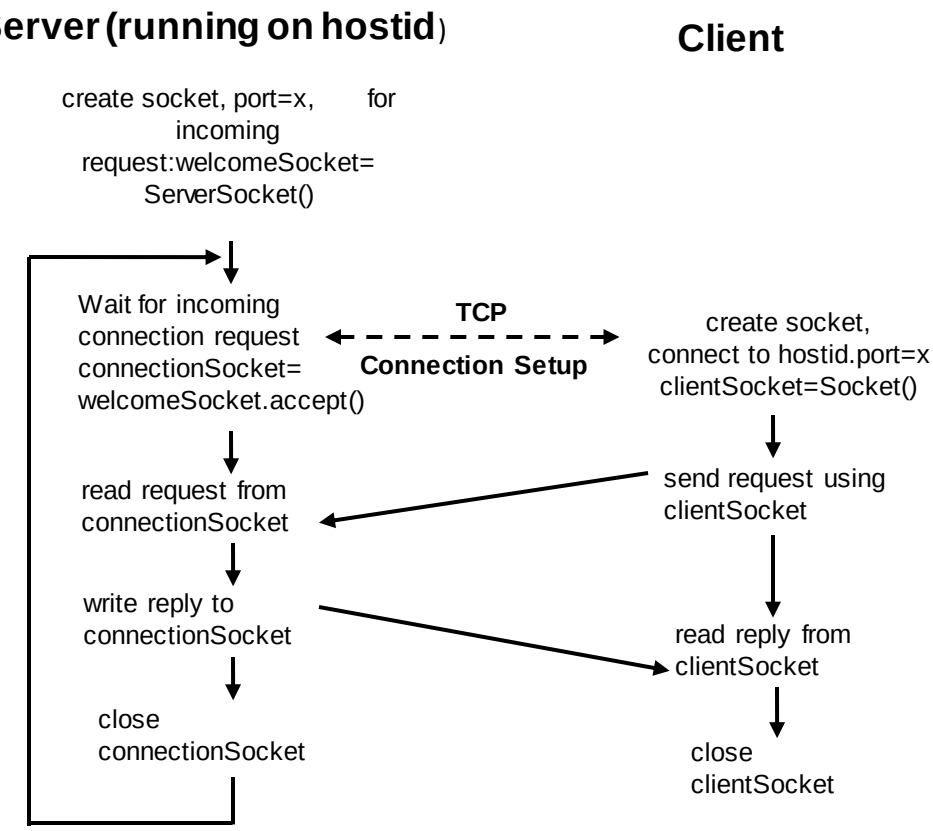
B) Creating TCP Servers & Clients:

TCP /IP ServerSockets:

c) Client /Server Communication using TCP /IP:

- After the connections are established, communication can occur using I/O streams. Each socket has both an OutputStream and an InputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream. TCP is a two-way communication protocol, so data can be sent across both streams at the same time.

The following figure illustrates the steps needed to implement client and server using StreamSocket



2.4 Creating a Socket to Remote Host On a Port (Creating TCP Client and Server)

Java

B) Creating TCP Servers & Clients:

TCP /IP ServerSockets:

d) Steps to Implement a Server:

1) Step 1:

Create a server socket object. The objects is used to connect to the application using port.

ServerSocket s=new ServerSocket(int port, int queue length);

Port -port number

Queue length=Maximum number of clients that can request the server

2) Step 2:

Wait for a request from the client. This is implemented using Socket object.

Socket ss=s.accept();

3) Step 3:

Create input and output streams for receiving and sending communication to the client respectively.

BufferedReader in = new BufferedReader(new InputStream (ss.getInputStream()));

BufferedReader – For receiving communication from the client.

PrintWriter out = new PrintWriter(ss.getOutStream(),true);

PrintWriter = For sending communication to the client

2.4 Creating a Socket to Remote Host On a Port (Creating TCP Client and Server)

Java

B) Creating TCP Servers & Clients:

TCP /IP ServerSockets:

d) Steps to Implement a Server:

4) Step 4:

Server and client communicate using the objects in and out.

```
out.println(message);
Message = (String)in.readLine();
```

5) Step 5:

When communication is over close all the created input, output and socket objects.

```
in.close();
out.close();
ss.close();
```

e) Steps to Implement a Client using StreamSocket:

1) Step 1:

Create a client socket object to connect the client with the server. The object con is used to connect to the server application using port.

```
Socket cs=new Socket(InetAddress, int port);
```

Port -port number

Address –address of the server.

2.4 Creating a Socket to Remote Host On a Port (Creating TCP Client and Server)

Java

B) Creating TCP Servers & Clients:

TCP /IP ServerSockets:

e) Steps to Implement a Client using StreamSocket:

2) Step 2:

Create input and output streams for receiving and sending communication to the client respectively.

BufferedReader in = new BufferedReader(new InputStream cs.getInputStream()));

BufferedReader – For receiving communication from the client.

PrintWriter out = new PrintWriter(cs.getOutStream(),true);

PrintWriter = For sending communication to the client

3) Step 3:

Server and client communicate using the objects in and out.

out.println(message);

Message = (String)in.readLine();

4) Step 4:

When communication is over close all the created input, output and socket objects.

in.close();

out.close();

cs.close();

2.4 Creating a Socket to Remote Host On a Port (Creating TCP Client and Server)

Java

B) Creating TCP Servers & Clients:

TCP /IP ServerSockets:

e) Steps to Implement a Client using StreamSocket:

2) Step 2:

Create input and output streams for receiving and sending communication to the client respectively.

BufferedReader in = new BufferedReader(new InputStream cs.getInputStream()));

BufferedReader – For receiving communication from the client.

PrintWriter out = new PrintWriter(cs.getOutStream(),true);

PrintWriter = For sending communication to the client

3) Step 3:

Server and client communicate using the objects in and out.

out.println(message);

Message = (String)in.readLine();

4) Step 4:

When communication is over close all the created input, output and socket objects.

in.close();

out.close();

cs.close();