# **2.1 Socket Addresses**

Addresses can be specified by a string that contains either a numeric address(Ipv4, Ipv6) or a name.

The InetAddress abstraction represents a network destination, encapsulating both names and numerical address information.

The class has two subclasses, Inet4Address and Inet6Address, representing the two versions in use.

Instances of InetAddress are immutable: once created, each one always refers to the same address

# **2.2 TCP Sockets**

Two classes for TCP: Socket and ServerSocket.

- Socket represents one end of a TCP connection.

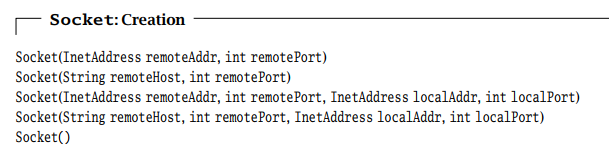
- ServerSocket listens for TCP connectionrequests and creates a new Socket instance to handle each incoming connection

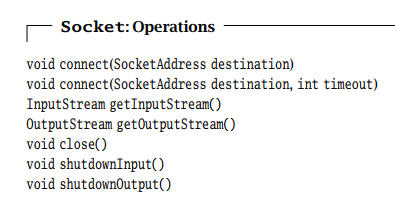
=> *servers handle both ServerSocket and Socket instances, while clients use only Socket.*

## 2.2.1 TCP Client

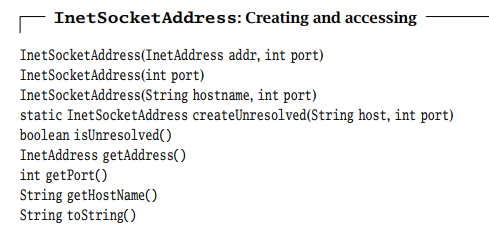
Thetypical TCP client has three steps:1. Construct an instance of **Socket** - establishes a TCP connection

2. Communicate using the socket’s I/O streams.3. **close()**

****







## 2.2.2 TCP Server

The typical TCP server has two steps:

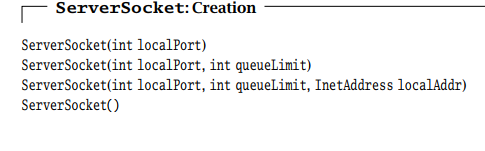
1. Construct a ServerSocket instance, specifying the local port.

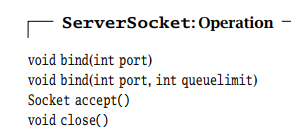
2. Repeatedly:

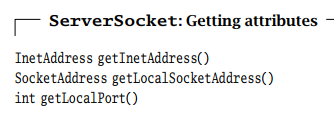
a. Call the accept() method to get the next incoming client connection.Upon establishment of a new client connection, an instance of Socket for the new connection is created and returned by accept().

b. Communicate with the client using IOStream

c. When finished, close the new client socket connection.

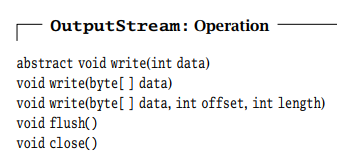


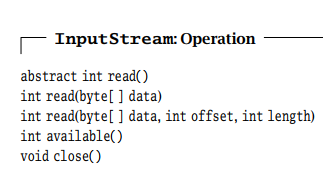




## 2.2.3 Input and Output Streams

Java inputstreams support reading bytes, and output streams support writing bytes

OutputStream: write bytes => flush => close

InputStream: read bytes => close

# **2.3 UDP Sockets**

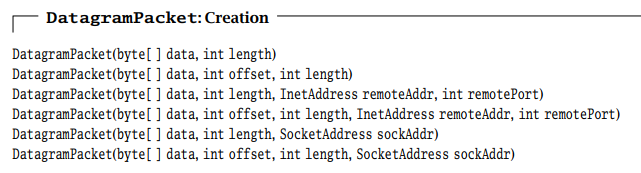
Different form TCP. UDP has two functions:

1) it adds another layer of addressing (ports) to that of IP.

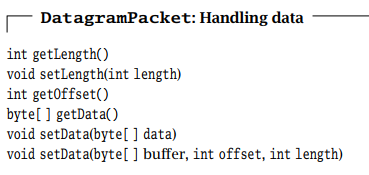
2) it detects some forms of data corruption that may occur in transit and discards any corrupted messages.

## 2.3.1 DatagramPacket

Instead of sending and receiving streams of bytes, UDP endpoints exchange self-contained messages, called datagrams.

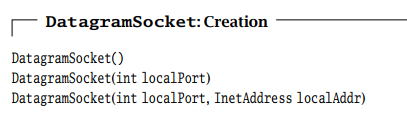


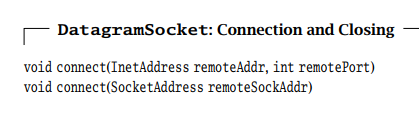




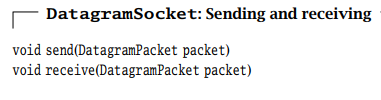
## 2.3.2 UDP Client

The typical UDP client has three steps:1. Construct a **DatagramSocket** (optionally specifying the local address and port)2. Communicate by sending and receiving instances of **DatagramPacket** using the **send()** and **receive()** methods of **DatagramSocket**.3. When finished, **close()** method of **DatagramSocket**.









## 2.3.3 UDP Server

The typical UDP server has three steps:1. Construct a **DatagramSocket**, specifying the local port and, optionally, thelocal address. 2. Receive an instance of **DatagramPacket** using the **receive()** method of **DatagramSocket**. When **receive()** returns, the datagram contains the client’s address so weknow where to send the reply.3. Communicate by sending and receiving **DatagramPackets** using the **send()** and **receive()**methods of **DatagramSocket**.

## 2.3.4 Sending and Receiving with UDP Sockets

- receive() on UDP does not provide recovery from network errors and does not Buffer data for possible retransmission.

- The received data may have come from different senders

- receive() will never return more than one message; if receive() with DatagramPacket containing n size buffer, and the first message size > n => n bytes are returned. The remaining are discarded => lost.

=> supply a DatagramPacket that has enough space