**Java Networking**

Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

Java socket programming provides facility to share data between different computing devices.

Advantage of Java Networking

1. sharing resources
2. centralize software management

**Java Networking Terminology**

The widely used java networking terminologies are given below:

1. IP Address
2. Protocol
3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol
6. Socket

**1) IP Address**

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255.

It is a logical address that can be changed.

**2) Protocol**

A protocol is a set of rules basically that is followed for communication. For example:

* TCP
* FTP
* Telnet
* SMTP
* POP etc.

**3) Port Number**

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

**4) MAC Address**

MAC (Media Access Control) Address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC.

**5) Connection-oriented and connection-less protocol**

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.

But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

**6) Socket**

A socket is an endpoint between two way communications.

* **Networking package is java.net**
  + Stream-based communications
    - Applications view networking as streams of data
    - Connection-based protocol
    - Uses TCP (Transmission Control Protocol
  + Packet-based communications
    - Individual packets transmitted
    - Connectionless service
    - Uses UDP (User Datagram Protocol)
* **Client-server relationship**
  + Client requests some action be performed
  + Server performs the action and responds to client
  + Request-response model
    - Common implementation: Web browsers and Web servers
    - **Java Socket Programming**

Java Socket programming is used for communication between the applications running on different JRE.

Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and Datagram Socket and Datagram Packet classes are used for connection-less socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Important methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public InputStream getInputStream() | returns the InputStream attached with this socket. |
| 2) public OutputStream getOutputStream() | returns the OutputStream attached with this socket. |
| 3) public synchronized void close() | closes this socket |

**ServerSocket class**

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Important methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public Socket accept() | returns the socket and establish a connection between server and client. |
| 2) public synchronized void close() | closes the server socket. |

Example of Java Socket Programming

Let's see a simple of java socket programming in which client sends a text and server receives it.

*File: MyServer.java*

1. **import** java.io.\*;
2. **import** java.net.\*;
3. **public** **class** MyServer {
4. **public** **static** **void** main(String[] args){
5. **try**{
6. ServerSocket ss=**new** ServerSocket(6666);
7. Socket s=ss.accept();//establishes connection
8. DataInputStream dis=**new** DataInputStream(s.getInputStream());
9. String  str=(String)dis.readUTF();
10. System.out.println("message= "+str);
11. ss.close();
12. }**catch**(Exception e){System.out.println(e);}
13. }
14. }

*File: MyClient.java*

1. **import** java.io.\*;
2. **import** java.net.\*;
3. **public** **class** MyClient {
4. **public** **static** **void** main(String[] args) {
5. **try**{
6. Socket s=**new** Socket("localhost",6666);
7. DataOutputStream dout=**new** DataOutputStream(s.getOutputStream());
8. dout.writeUTF("Hello Server");
9. dout.flush();
10. dout.close();
11. s.close();
12. }**catch**(Exception e){System.out.println(e);}
13. }
14. }

To execute this program open two command prompts and execute each program at each command prompt as displayed in the below figure.

After running the client application, a message will be displayed on the server console.

**Example of Java Socket Programming (Read-Write both side)**

In this example, client will write first to the server then server will receive and print the text. Then server will write to the client and client will receive and print the text. The step goes on.

*File: MyServer.java*

1. **import** java.net.\*;
2. **import** java.io.\*;
3. **class** MyServer{
4. **public** **static** **void** main(String args[])**throws** Exception{
5. ServerSocket ss=**new** ServerSocket(3333);
6. Socket s=ss.accept();
7. DataInputStream din=**new** DataInputStream(s.getInputStream());
8. DataOutputStream dout=**new** DataOutputStream(s.getOutputStream());
9. BufferedReader br=**new** BufferedReader(**new** InputStreamReader(System.in));
11. String str="",str2="";
12. **while**(!str.equals("stop")){
13. str=din.readUTF();
14. System.out.println("client says: "+str);
15. str2=br.readLine();
16. dout.writeUTF(str2);
17. dout.flush();
18. }
19. din.close();
20. s.close();
21. ss.close();
22. }}

*File: MyClient.java*

1. **import** java.net.\*;
2. **import** java.io.\*;
3. **class** MyClient{
4. **public** **static** **void** main(String args[])**throws** Exception{
5. Socket s=**new** Socket("localhost",3333);
6. DataInputStream din=**new** DataInputStream(s.getInputStream());
7. DataOutputStream dout=**new** DataOutputStream(s.getOutputStream());
8. BufferedReader br=**new** BufferedReader(**new** InputStreamReader(System.in));
10. String str="",str2="";
11. **while**(!str.equals("stop")){
12. str=br.readLine();
13. dout.writeUTF(str);
14. dout.flush();
15. str2=din.readUTF();
16. System.out.println("Server says: "+str2);
17. }
19. dout.close();
20. s.close();
21. }}

The **Java URL** class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web. For example:

http://www.google.com/

A URL contains many information:

1. **Protocol:** In this case, http or https is the protocol.
2. **Server name or IP Address:** In this case, www.example.com is the server name.
3. **Port Number:** It is an optional attribute. If we write http//www.example.com:80/nitinsir/, 80 is the port number. If port number is not mentioned in the URL, it returns -1.
4. **File Name or directory name:** In this case, index.jsp is the file name.

Commonly used methods of Java URL class

The java.net.URL class provides many methods. The important methods of URL class are given below.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public String getProtocol() | it returns the protocol of the URL. |
| public String getHost() | it returns the host name of the URL. |
| public String getPort() | it returns the Port Number of the URL. |
| public String getFile() | it returns the file name of the URL. |
| public URLConnection openConnection() | it returns the instance of URLConnection i.e., associated with this URL. |

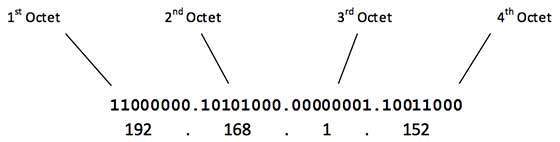
Example of Java URL class

1. //URLDemo.java
2. **import** java.io.\*;
3. **Import** java.net.\*;
4. **public** **class** URLDemo{
5. **public** **static** **void** main(String[] args){
6. **try**{
7. URL url=**new** URL("http://www.google.com/ ");
9. System.out.println("Protocol: "+url.getProtocol());
10. System.out.println("Host Name: "+url.getHost());
11. System.out.println("Port Number: "+url.getPort());
12. System.out.println("File Name: "+url.getFile());
14. }**catch**(Exception e){System.out.println(e);}
15. }
16. }

Internet Protocol hierarchy contains several classes of IP Addresses to be used efficiently in various situations as per the requirement of hosts per network. Broadly, the IPv4 Addressing system is divided into five classes of IP Addresses. All the five classes are identified by the first octet of IP Address.

Internet Corporation for Assigned Names and Numbers is responsible for assigning IP addresses.

The first octet referred here is the left most of all. The octets numbered as follows depicting dotted decimal notation of IP Address:



The number of networks and the number of hosts per class can be derived by this formula:



When calculating hosts' IP addresses, 2 IP addresses are decreased because they cannot be assigned to hosts, i.e. the first IP of a network is network number and the last IP is reserved for Broadcast IP.

Class A Address

The first bit of the first octet is always set to 0 (zero). Thus the first octet ranges from 1 – 127, i.e.

Class A Addresses

Class A addresses only include IP starting from 1.x.x.x to 126.x.x.x only. The IP range 127.x.x.x is reserved for loopback IP addresses.

The default subnet mask for Class A IP address is 255.0.0.0 which implies that Class A addressing can have 126 networks (27-2) and 16777214 hosts (224-2).

Class A IP address format is thus:**0NNNNNNN**.HHHHHHHH.HHHHHHHH.HHHHHHHH

Class B Address

An IP address which belongs to class B has the first two bits in the first octet set to 10, i.e.

Class B Addresses

Class B IP Addresses range from 128.0.x.x to 191.255.x.x. The default subnet mask for Class B is 255.255.x.x.

Class B has 16384 (214) Network addresses and 65534 (216-2) Host addresses.

Class B IP address format is:**10NNNNNN.NNNNNNNN**.HHHHHHHH.HHHHHHHH

Class C Address

The first octet of Class C IP address has its first 3 bits set to 110, that is:

Class C Addresses

Class C IP addresses range from 192.0.0.x to 223.255.255.x. The default subnet mask for Class C is 255.255.255.x.

Class C gives 2097152 (221) Network addresses and 254 (28-2) Host addresses.

Class C IP address format is:**110NNNNN.NNNNNNNN.NNNNNNNN**.HHHHHHHH

Class D Address

Very first four bits of the first octet in Class D IP addresses are set to 1110, giving a range of:

Class D Addresses

Class D has IP address rage from 224.0.0.0 to 239.255.255.255. Class D is reserved for Multicasting. In multicasting data is not destined for a particular host, that is why there is no need to extract host address from the IP address, and Class D does not have any subnet mask.

Class E Address

This IP Class is reserved for experimental purposes only for R&D or Study. IP addresses in this class ranges from 240.0.0.0 to 255.255.255.254. Like Class D, this class too is not equipped with any subnet mask.

Each IP class is equipped with its own default subnet mask which bounds that IP class to have prefixed number of Networks and prefixed number of Hosts per network. Classful IP addressing does not provide any flexibility of having less number of Hosts per Network or more Networks per IP Class.

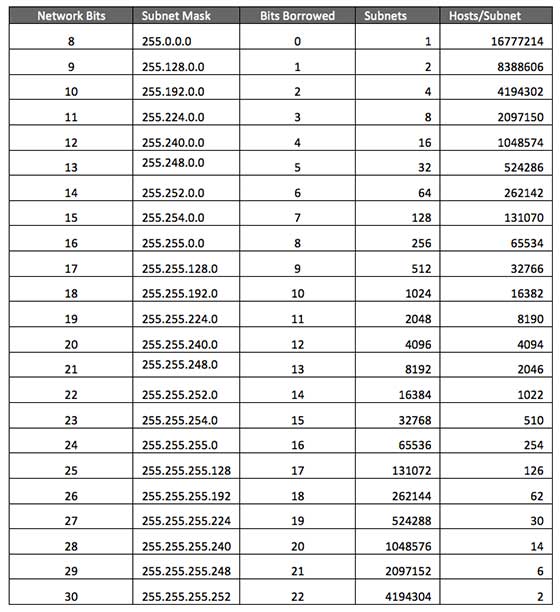
CIDR or **Classless Inter Domain Routing** provides the flexibility of borrowing bits of Host part of the IP address and using them as Network in Network, called Subnet. By using subnetting, one single Class A IP address can be used to have smaller sub-networks which provides better network management capabilities.

Class A Subnets

In Class A, only the first octet is used as Network identifier and rest of three octets are used to be assigned to Hosts (i.e. 16777214 Hosts per Network). To make more subnet in Class A, bits from Host part are borrowed and the subnet mask is changed accordingly.

For example, if one MSB (Most Significant Bit) is borrowed from host bits of second octet and added to Network address, it creates two Subnets (21=2) with (223-2) 8388606 Hosts per Subnet.

The Subnet mask is changed accordingly to reflect subnetting. Given below is a list of all possible combination of Class A subnets:



In case of subnetting too, the very first and last IP address of every subnet is used for Subnet Number and Subnet Broadcast IP address respectively. Because these two IP addresses cannot be assigned to hosts, sub-netting cannot be implemented by using more than 30 bits as Network Bits, which provides less than two hosts per subnet.

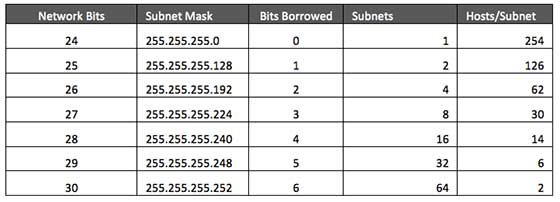
Class B Subnets

By default, using Classful Networking, 14 bits are used as Network bits providing (214) 16384 Networks and (216-2) 65534 Hosts. Class B IP Addresses can be subnetted the same way as Class A addresses, by borrowing bits from Host bits. Below is given all possible combination of Class B subnetting:



Class C Subnets

Class C IP addresses are normally assigned to a very small size network because it can only have 254 hosts in a network. Given below is a list of all possible combination of subnetted Class B IP address:



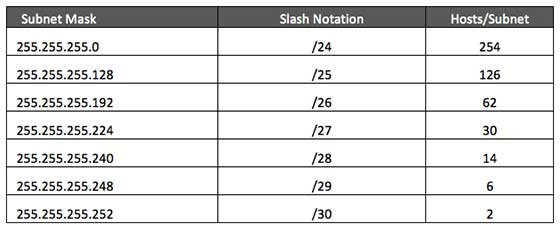
nternet Service Providers may face a situation where they need to allocate IP subnets of different sizes as per the requirement of customer. One customer may ask Class C subnet of 3 IP addresses and another may ask for 10 IPs. For an ISP, it is not feasible to divide the IP addresses into fixed size subnets, rather he may want to subnet the subnets in such a way which results in minimum wastage of IP addresses.

For example, an administrator have 192.168.1.0/24 network. The suffix /24 (pronounced as "slash 24") tells the number of bits used for network address. In this example, the administrator has three different departments with different number of hosts. Sales department has 100 computers, Purchase department has 50 computers, Accounts has 25 computers and Management has 5 computers. In CIDR, the subnets are of fixed size. Using the same methodology the administrator cannot fulfill all the requirements of the network.

The following procedure shows how VLSM can be used in order to allocate department-wise IP addresses as mentioned in the example.

Step - 1

Make a list of Subnets possible.



Step - 2

Sort the requirements of IPs in descending order (Highest to Lowest).

* Sales 100 , Purchase 50, Accounts 25 , Management 5

Step - 3

Allocate the highest range of IPs to the highest requirement, so let's assign 192.168.1.0 /25 (255.255.255.128) to the Sales department. This IP subnet with Network number 192.168.1.0 has 126 valid Host IP addresses which satisfy the requirement of the Sales department. The subnet mask used for this subnet has 10000000 as the last octet.

Step - 4

Allocate the next highest range, so let's assign 192.168.1.128 /26 (255.255.255.192) to the Purchase department. This IP subnet with Network number 192.168.1.128 has 62 valid Host IP Addresses which can be easily assigned to all the PCs of the Purchase department. The subnet mask used has 11000000 in the last octet.

Step - 5

Allocate the next highest range, i.e. Accounts. The requirement of 25 IPs can be fulfilled with 192.168.1.192 /27 (255.255.255.224) IP subnet, which contains 30 valid host IPs. The network number of Accounts department will be 192.168.1.192. The last octet of subnet mask is 11100000.

Step - 6

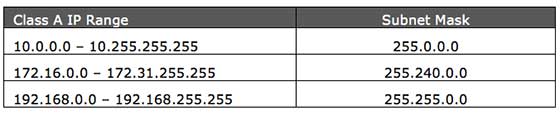
Allocate the next highest range to Management. The Management department contains only 5 computers. The subnet 192.168.1.224 /29 with the Mask 255.255.255.248 has exactly 6 valid host IP addresses. So this can be assigned to Management. The last octet of the subnet mask will contain 11111000.

By using VLSM, the administrator can subnet the IP subnet in such a way that least number of IP addresses are wasted. Even after assigning IPs to every department, the administrator, in this example, is still left with plenty of IP addresses which was not possible if he has used CIDR.

There are a few reserved IPv4 address spaces which cannot be used on the internet. These addresses serve special purpose and cannot be routed outside the Local Area Network.

Private IP Addresses

Every class of IP, (A, B & C) has some addresses reserved as Private IP addresses. These IPs can be used within a network, campus, company and are private to it. These addresses cannot be routed on the Internet, so packets containing these private addresses are dropped by the Routers.



In order to communicate with the outside world, these IP addresses must have to be translated to some public IP addresses using NAT process, or Web Proxy server can be used.

The sole purpose to create a separate range of private addresses is to control assignment of already-limited IPv4 address pool. By using a private address range within LAN, the requirement of IPv4 addresses has globally decreased significantly. It has also helped delaying the IPv4 address exhaustion.

IP class, while using private address range, can be chosen as per the size and requirement of the organization. Larger organizations may choose class A private IP address range where smaller organizations may opt for class C. These IP addresses can be further sub-netted and assigned to departments within an organization.

Loopback IP Addresses

The IP address range 127.0.0.0 – 127.255.255.255 is reserved for loopback, i.e. a Host’s self-address, also known as localhost address. This loopback IP address is managed entirely by and within the operating system. Loopback addresses, enable the Server and Client processes on a single system to communicate with each other. When a process creates a packet with destination address as loopback address, the operating system loops it back to itself without having any interference of NIC.

Data sent on loopback is forwarded by the operating system to a virtual network interface within operating system. This address is mostly used for testing purposes like client-server architecture on a single machine. Other than that, if a host machine can successfully ping 127.0.0.1 or any IP from loopback range, implies that the TCP/IP software stack on the machine is successfully loaded and working.

Link-local Addresses

In case a host is not able to acquire an IP address from the DHCP server and it has not been assigned any IP address manually, the host can assign itself an IP address from a range of reserved Link-local addresses. Link local address ranges from 169.254.0.0 -- 169.254.255.255.

Assume a network segment where all systems are configured to acquire IP addresses from a DHCP server connected to the same network segment. If the DHCP server is not available, no host on the segment will be able to communicate to any other. Windows (98 or later), and Mac OS (8.0 or later) supports this functionality of self-configuration of Link-local IP address. In absence of DHCP server, every host machine randomly chooses an IP address from the above mentioned range and then checks to ascertain by means of ARP, if some other host also has not configured itself with the same IP address. Once all hosts are using link local addresses of same range, they can communicate with each other.

These IP addresses cannot help system to communicate when they do not belong to the same physical or logical segment. These IPs are also not routable.

**Native Class**  
A  Java source file can define a class and its methods. Method functions  can perform computations and other operations based on the primitive  data types, array indexing, and string concatenation. They can also  create objects of other classes and call methods.  
Java  code cannot directly make calls to the local operating system to access  disk files, run programs, use network resources, or query databases.  Instead, such requests must be made indirectly through a **native class**.  
To  build a native class, the programmer starts with Java code that defines  the field variables and the names and arguments of all methods. The methods contain no code. This file is then processed by a utility that  converts it into C. The rest of the programming is done with the native C language and all the subroutines available through it.

**Java DatagramSocket and DatagramPacket**

Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

Java DatagramSocket class

**Java DatagramSocket** class represents a connection-less socket for sending and receiving datagram packets.

A datagram is basically information but there is no guarantee of its content, arrival or arrival time.

Commonly used Constructors of DatagramSocket class

* **DatagramSocket() throws SocketEeption:**it creates a datagram socket and binds it with the available Port Number on the localhost machine.
* **DatagramSocket(int port) throws SocketEeption:**it creates a datagram socket and binds it with the given Port Number.
* **DatagramSocket(int port, InetAddress address) throws SocketEeption:**it creates a datagram socket and binds it with the specified port number and host address.

Java DatagramPacket class

**Java DatagramPacket** is a message that can be sent or received. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Commonly used Constructors of DatagramPacket class

* **DatagramPacket(byte[] barr, int length):**it creates a datagram packet. This constructor is used to receive the packets.
* **DatagramPacket(byte[] barr, int length, InetAddress address, int port):**it creates a datagram packet. This constructor is used to send the packets.

Example of Sending DatagramPacket by DatagramSocket

1. //DSender.java
2. **import** java.net.\*;
3. **public** **class** DSender{
4. **public** **static** **void** main(String[] args) **throws** Exception {
5. DatagramSocket ds = **new** DatagramSocket();
6. String str = "Welcome java";
7. InetAddress ip = InetAddress.getByName("127.0.0.1");
9. DatagramPacket dp = **new** DatagramPacket(str.getBytes(), str.length(), ip, 3000);
10. ds.send(dp);
11. ds.close();
12. }
13. }

Example of Receiving DatagramPacket by DatagramSocket

1. //DReceiver.java
2. **import** java.net.\*;
3. **public** **class** DReceiver{
4. **public** **static** **void** main(String[] args) **throws** Exception {
5. DatagramSocket ds = **new** DatagramSocket(3000);
6. **byte**[ ] buf = **new** **byte**[1024];
7. DatagramPacket dp = **new** DatagramPacket(buf, 1024);
8. ds.receive(dp);
9. String str = **new** String(dp.getData(), 0, dp.getLength());
10. System.out.println(str);
11. ds.close();
12. }
13. }

**Datagrams and Streams**

It's important to understand the difference between datagrams and streams, since IP is a totally datagram-oriented protocol, though most network operations use streams.

In modern data networking, it is important to distinguish between datagrams and streams.

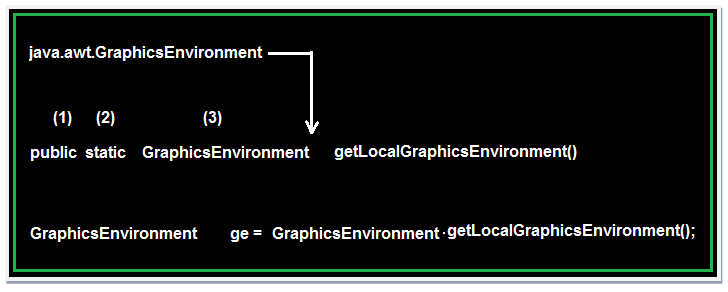
A stream is what we'd typically think of as a communication channel. Remote logins, file transfers, mail delivery - all use streams. A stream appears to be like a pipeline. It has two endpoints. Data is put in one end and comes out the other. None of the data is duplicated, or discarded, or reorganized in any way. Two streams can be paired together to form a full duplex connection.

A datagram, often called a packet, is much more atomic in nature. It is a small piece of data, often required to be less than a maximum length (typically in the 256 to 2000 byte range). Datagrams are completely self contained. They have a source and a destination, but nothing that could be called a connection. Datagrams have no relationship to any others that came before or after them.

Although most networking communication uses streams, all Internet transfers are in the form of datagrams. Internet streams are actually emulated using datagrams by the [TCP Protocol](http://www.freesoft.org/CIE/Topics/83.htm). To diagnose Internet operation, a packet decoder such as [TCPdump](http://www.freesoft.org/CIE/Topics/55.htm) is used to view individual packets. This, along with a knowledge of TCP operation, enables the Internet engineer to assemble a mental picture of network operation.

## Factory Methods

A factory method is one whose return type is similar to the class name in which class is present. The purpose of factory method is to create an object without using new operator.



## Rules for writing factory method

1. The return type of the factory method must be similar to class name in which class it presents.
2. Every factory method in java is static.
3. The access specifier of the factory method must be public.

Factory methods are static methods that return an instance of the native class. Since the InetAddress class no visible constructors we use the factory methods to create the InetAddress objects. The commonly used InetAddress class factory methods are

**1.    static InetAddress [ ] getAllByName (String hostname)**

Returns an array of InetAddress instances representing the hostname.

**2.    static InetAddress getByName (String hostname)**

Returns an InetAddress instance representing the hostname.

**3.    static InetAddress getLocalHost ( )**

Returns the IP address of the localhost machine.

**InetAddress**

This class encapsulates the numerical IP address and the domain name for the address. Factory methods of a class allow you to call the method without referencing the object. The factory methods of this class are:

* getLocalHost() method: Returns the name of the local computer
* getByName() method: Returns the address by the Domain name
* getAllByName() method: Returns all the addresses by their domain name  
  The instance methods of a class are methods that can be called from an object only. The instance methods for the class are:
* getAddress() method: Returns a four-element byte array that represents the object’s IP address in network byte order
* getHostAddress() method: Returns the host address
* getHostName() method: Returns the hostname that is associated with the host address.

## Socket Programming in Java

**Networking** is a concept of connecting two or more computing devices together so that we can share resources like printer, scanner, memory.

In Networking application mainly two programs are running one is **Client program** and another is **Server program**. In Core java **Client program** can be design using **Socket** class and **Server program** can be design using **ServerSocket** class.

Both Socket and ServerSocket classes are predefined in **java.net** package

### Advantage of Network Programming

The main advantage of network Programming is sharing of data and resources, some more advantages are;

* Sharing resources like printer, Scanner.
* Centralize software management, Software install on only one system and used in multiple system.
* Sharing of data due to this reduce redundancy of application.
* Burden on the developer can be reduced.
* Wastage of memory can be reduced because no need to install same application on every system.
* Time consuming process to develop application is reduced.

### Terms used in Socket Programming

**Port number:**It is unique identification value represents residing position of a server in the computer. It is four digit +ve number.

**Port Name:**It is a valid user defined name to know about client system, the default port name for any local computer is **localhost.**. Port name should be the some value which is given at Server programming.

## Socket class

Socket class are used for design a client program, it have some constructor and methods which are used in designing client program.

**Constructor:**Socket class is having a constructor through this Client program can request to server to get connection.

## Syntax to call Socket() Constructor

Socket s=**new** Socket("localhost", 8080);

// localhost -- port name and 8080 -- port number

**Note:**If given port name is invalid then UnknownHostException will be raised.

**Method of Socket class**

* public InputStream getInputStream()
* public OutputStream getOutputStream()
* public synchronized void close()

### getInputStream()

This method take the permission to write the data from client program to server program and server program to client program which returns OutputStream class object.

## Syntax

Socket s=**new** Socket("localhost", 8080);

OutputStream os=**new** s.getOutputStream();

DataOutputStream dos=**new** DataOutputStream(os);

### getOutputStream()

This method is used to take the permission to read data from client system by the server or from the server system by the client which returns InputStream class object.

## Syntax

Socket s=**new** Socket("localhost", 8080);

InputStream **is**=**new** s.getInputStream();

DataInputStream dis=**new** DataInputStream(**is**);

### close()

This method is used to request for closing or terminating an object of socket class or it is used to close client request.

## Syntax

Socket s=**new** Socket("localhost", 8080);

s.close();

## ServerSocket class

The ServerSocket class can be used to create a server socket. ServerSocket object is used to establish the communication with clients.

ServerSocket class are used for design a server program, it have some constructor and methods which are used in designing server program.

**Constructor:**ServerSocket class contain a constructor used to create a separate port number to run the server program.

## Syntax to call ServerSocket() Constructor

ServerSocket ss=**new** ServerSocket(8080);

// 8080 -- port number

**Method of ServerSocket class**

* public Socket accept()
* public InputStream getInputStream()
* public OutputStream getOutputStream()
* public synchronized void close()

**accept():**Used to accept the client request it returns class reference.

## Syntax

Socket s=**new** Socket("localhost", 8080);

ServerSocket ss=**new** ServerSocket(8080);

Socket s=ss.accept();

### Rules to design server program

* Every server program should run accepted port number (any 4 digit +ve numeric value) It can set by relating an object for server socket class (In any used defined java program).

## Syntax

ServerSocket ss=**new** ServerSocket(8080);

* Accept client request.
* Read input data from client using InputStream class.
* Perform valid business logic operation
* Send response (writing output data) back to client using OutputStream class.
* close or terminate client request.

### Rules to design client program

* Obtain connection to the server from the client program (any user defined class) by passing port number and port name in the socket class.

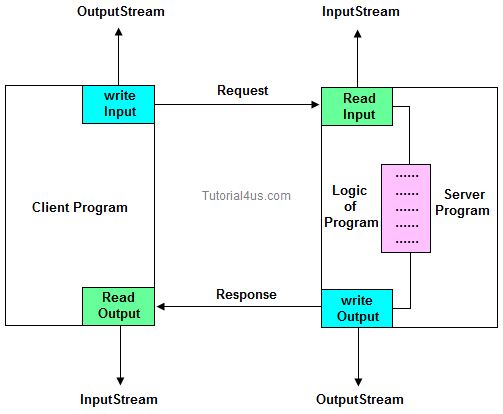
## Syntax

Socket s=**new** Socket(8080, "localhost");

// 8080 is port number and localhost is port name

* Send request (writing input data) to the server using OutputStream class.
* Read output data from the server using InputStream class.
* Display output data

**Note:**close the connection is optional.



## Server Code

// Saved by Server.java

**import** java.net.\*;

**import** java.io.\*;

**class** Server

{

**public** **static** **void** main(String[] args)

{

**try**

{

**int** pno=Integer.parseInt(args[0]);

ServerSocket ss=**new** ServerSocket(pno);

System.**out**.println("server is ready to accept clint request");

Socket s1=ss.accept();

InputStream **is**=s1.getInputStream();

DataInputStream dis=**new** DataInputStream(**is**);

**int** n=dis.readInt();

System.**out**.println("Value from client : "+n);

**int** res=n\*n;

OutputStream os=s1.getOutputStream();

DataOutputStream dos=**new** DataOutputStream(os);

dos.writeInt(res);

s1.close();

}

**catch** (Exception e)

{

System.**out**.println(e);

}

}

}

## Client Code

// Saved by Client.java

**import** java.net.\*;

**import** java.io.\*;

**import** java.util.\*;

**class** Client

{

**public** **static** **void** main(String[] args)

{

**try**

{

String pname=args[0];

**int** pno=Integer.parseInt(args[1]);

Socket s=**new** Socket(pname,pno);

System.**out**.println("clint obtailed connection from server");

System.**out**.println("Enter a number ");

Scanner sn=**new** Scanner(System.**in**);

**int** data=sn.nextInt();

OutputStream os=s.getOutputStream();

DataOutputStream dos=**new** DataOutputStream(os);

dos.writeInt(data);

InputStream **is**=s.getInputStream();

DataInputStream dis=**new** DataInputStream(**is**);

**int** res=dis.readInt();

System.**out**.println("Result from server : "+res);

}

**catch** (Exception e)

{

System.**out**.println(e);

}

}

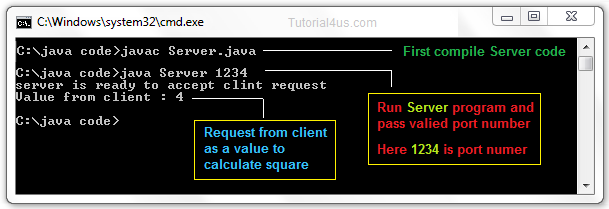
}

Download Code [Client Server code](http://www.sitesbay.com/java/code/client-server.rar)

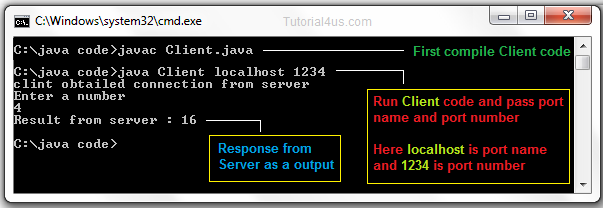
### Steps to run above program

* First open two command prompt window separately.
* Compile Client program on one command prompt and compile server program on other command prompt.
* First run Server program and pass valid port number.
* Second run client program and pass valid port name and post number (which is already passed at server).
* Finally give the request to server from client (from client command prompt).
* Now you can get response from server.

### Compile and run server program



### Compile and run client program



### Limitation of J2SE network programming

* Using this concept you can share resource locally but not globally.
* Using this concept you cannot develop internet based application.
* Using this concept you can develop only half-duplex application.
* Using this concept you cannot get service from universal protocols like http, ftp etc...
* Using this concept you cannot get services from universal server software like tomcat, weblogic etc..

To overcome all these limitation use Servlet and JSP technology.

**InetAddress**

This class encapsulates the numerical IP address and the domain name for the address. Factory methods of a class allow you to call the method without referencing the object. The factory methods of this class are:

* getLocalHost() method: Returns the name of the local computer
* getByName() method: Returns the address by the Domain name
* getAllByName() method: Returns all the addresses by their domain name  
  The instance methods of a class are methods that can be called from an object only. The instance methods for the class are:
* getAddress() method: Returns a four-element byte array that represents the object’s IP address in network byte order
* getHostAddress() method: Returns the host address
* getHostName() method: Returns the hostname that is associated with the host address.

## Difference Between

### Difference between non-static and static variable ?

|  |  |  |
| --- | --- | --- |
|  | **Non-Static method** | **Static method** |
| 1 | These method never be preceded by static keyword Example:  **void** fun1()  {  ......  ......  } | These method always preceded by static keyword Example:  **static** **void** fun2()  {  ......  ......  } |
| 2 | Memory is allocated multiple time whenever method is calling. | Memory is allocated only once at the time of loading. |
| 3 | It is specific to an object so that these are also known as instance method. | These are common to every object so that it is also known as member method or class method. |
| 4 | These methods always access with object reference Syntax:  **Objref.methodname();** | These property always access with class reference Syntax:  **className.methodname();** |
| 5 | If any method wants to be execute multiple time that can be declare as non static. | If any method wants to be execute only once in the program that can be declare as static . |

### Difference between Method and Constructor

|  |  |  |
| --- | --- | --- |
|  | **Method** | **Constructor** |
| 1 | Method can be any user defined name | Constructor must be class name |
| 2 | Method should have return type | It should not have any return type (even void) |
| 3 | Method should be called explicitly either with object reference or class reference | It will be called automatically whenever object is created |
| 1 | Method is not provided by compiler in any case. | The java compiler provides a default constructor if we do not have any constructor. |

### Difference between package keyword and import keyword

* Package keyword is always used for creating the undefined package and placing common classes and interfaces.
* import is a keyword which is used for referring or using the classes and interfaces of a specific package.

### Difference between Class and Object

|  |  |  |
| --- | --- | --- |
|  | **Class** | **Object** |
| 1 | Class is a container which collection of variables and methods. | object is a instance of class |
| 2 | No memory is allocated at the time of declaration | Sufficient memory space will be allocated for all the variables of class at the time of declaration. |
| 3 | One class definition should exist only once in the program. | For one class multiple objects can be created. |

### Difference between Statement and PreparedStatement ?

|  |  |  |
| --- | --- | --- |
|  | **Statement** | **PreparedStatement** |
| 1 | Statement interface is slow because it compile the program for each execution | PreparedStatement interface is faster, because its compile the command for once. |
| 2 | We can not use ? symbol in sql command so setting dynamic value into the command is complex | We can use ? symbol in sql command, so setting dynamic value is simple. |
| 3 | We can not use statement for writing or reading binary data (picture) | We can use PreparedStatement for reading or writing binary data. |

### Difference between throw and throws ?

|  |  |  |
| --- | --- | --- |
|  | **throw** | **throws** |
| 1 | throw is a keyword used for hitting and generating the exception which are occurring as a part of method body | throws is a keyword which gives an indication to the specific method to place the common exception methods as a part of try and catch block for generating user friendly error messages |
| 2 | The place of using throw keyword is always as a part of method body. | The place of using throws is a keyword is always as a part of method heading |
| 3 | When we use throw keyword as a part of method body, it is mandatory to the java programmer to write throws keyword as a part of method heading | When we write throws keyword as a part of method heading, it is optional to the java programmer to write throw keyword as a part of method body. |

### Difference between checked Exception and un-checked Exception ?

|  |  |  |
| --- | --- | --- |
|  | **Checked Exception** | **Un-Checked Exception** |
| 1 | checked Exception are checked at compile time | un-checked Exception are checked at run time |
| 2 | e.g.  IOException, SQLException, FileNotFoundException etc. | e.g.  ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, NumberNotFoundException etc. |

### Difference between Abstraction and Encapsulation ?

Encapsulation is not provides fully security because we can access private member of the class using reflation API, but in case of Abstraction we can't access static, abstract data member of class.

### Difference between Abstract and Interface ?

|  |  |  |
| --- | --- | --- |
|  | **Abstract** | **Interface** |
| 1 | It is collection of abstract method and concrete methods. | It is collection of abstract method. |
| 2 | There properties can be reused commonly in a specific application. | There properties commonly usable in any application of java environment. |
| 3 | It does not support multiple inheritance. | It support multiple inheritance. |
| 4 | Abstract class is preceded by abstract keyword. | It is preceded by Interface keyword. |
| 5 | Which may contain either variable or constants. | Which should contains only constants. |
| 6 | The default access specifier of abstract class methods are default. | There default access specifier of interface method are public. |
| 7 | These class properties can be reused in other class using extend keyword. | These properties can be reused in any other class using implements keyword. |
| 8 | Inside abstract class we can take constructor. | Inside interface we can not take any constructor. |
| 9 | For the abstract class there is no restriction like initialization of variable at the time of variable declaration. | For the interface it should be compulsory to initialization of variable at the time of variable declaration. |
| 10 | There are no any restriction for abstract class variable. | For the interface variable can not declare variable as private, protected, transient, volatile. |
| 11 | There are no any restriction for abstract class method modifier that means we can use any modifiers. | For the interface method can not declare method as strictfp, protected, static, native, private, final, synchronized. |

### Difference between Overloading and Overriding ?

|  |  |  |
| --- | --- | --- |
|  | **Overloading** | **Overriding** |
| 1 | Whenever same method or Constructor is existing multiple times within a class either with different number of parameter or with different type of parameter or with different order of parameter is known as Overloading. | Whenever same method name is existing multiple time in both base and derived class with same number of parameter or same type of parameter or same order of parameters is known as Overriding. |
| 2 | Arguments of method must be different at least arguments. | Argument of method must be same including order. |
| 3 | Method signature must be different. | Method signature must be same. |
| 4 | Private, static and final methods can be overloaded. | Private, static and final methods can not be overrided. |
| 5 | Access modifiers point of view no restriction. | Access modifiers point of view not reduced scope of Access modifiers but increased. |
| 6 | Also known as compile time polymorphism or static polymorphism or early binding. | Also known as run time polymorphism or dynamic polymorphism or late binding. |
| 7 | Overloading can be exhibited both are method and constructor level. | Overriding can be exhibited only at method leve. |
| 8 | The scope of overloading is within the class. | The scope of Overriding is base class and derived class. |
| 9 | Overloading can be done at both static and non-static methods. | Overriding can be done only at non-static method. |
| 10 | For overloading methods return type may or may not be same. | For overriding method return type should be same. |

### Difference between sleep() and suspend() ?

Sleep() can be used to convert running state to waiting state and automatically thread convert from waiting state to running state once the given time period is completed. Where as suspend() can be used to convert running state thread to waiting state but it will never return back to running state automatically.

### Difference between String and StringBuffer ?

|  |  |  |
| --- | --- | --- |
|  | **String** | **StringBuffer** |
| 1 | The data which enclosed within double quote (" ") is by default treated as String class. | The data which enclosed within double quote (" ") is not by default treated as StringBuffer class |
| 2 | String class object is immutable | StringBuffer class object is mutable |
| 3 | When we create an object of String class by default no additional character memory space is created. | When we create an object of StringBuffer class by default we get 16 additional character memory space. |

### Difference between StringBuffer and StringBuilder ?

All the things between StringBuffer and StringBuilder are same only difference is StringBuffer is synchronized and StringBuilder is not synchronized. synchronized means one thread is allow at a time so it thread safe. Not synchronized means multiple threads are allow at a time so it not thread safe.

|  |  |  |
| --- | --- | --- |
|  | **StringBuffer** | **StringBuilder** |
| 1 | It is thread safe. | It is not thread safe. |
| 2 | Its methods are synchronized and provide thread safety. | Its methods are not synchronized and unable to provide thread safety. |
| 3 | Relatively performance is low because thread need to wait until previous process is complete. | Relatively performance is high because no need to wait any thread it allows multiple thread at a time. |
| 1 | Introduced in 1.0 version. | Introduced in 1.5 version. |

### Difference between Method and Constructor ?

|  |  |  |
| --- | --- | --- |
|  | **Method** | **Constructor** |
| 1 | Method can be any user defined name | Constructor must be class name |
| 2 | Method should have return type | It should not have any return type (even void) |
| 3 | Method should be called explicitly either with object reference or class reference | It will be called automatically whenever object is created |
| 1 | Method is not provided by compiler in any case. | The java compiler provides a default constructor if we do not have any constructor. |

### Difference between non-static and static Method

|  |  |  |
| --- | --- | --- |
|  | **Non-Static method** | **Static method** |
| 1 | These method never be preceded by static keyword Example:  **void** fun1()  {  ......  ......  } | These method always preceded by static keyword Example:  **static** **void** fun2()  {  ......  ......  } |
| 2 | Memory is allocated multiple time whenever method is calling. | Memory is allocated only once at the time of class loading. |
| 3 | It is specific to an object so that these are also known as instance method. | These are common to every object so that it is also known as member method or class method. |
| 4 | These methods always access with object reference Syntax:  **Objref.methodname();** | These property always access with class reference Syntax:  **className.methodname();** |
| 5 | If any method wants to be execute multiple time that can be declare as non static. | If any method wants to be execute only once in the program that can be declare as static . |

### Difference between non-static and static variable

|  |  |  |
| --- | --- | --- |
|  | **Non-static variable** | **Static variable** |
| 1 | These variable should not be preceded by any static keyword Example:  **class** A  {  **int** a;  } | These variables are preceded by static keyword. Example **class** A  {  **static** **int** b;  } |
| 2 | Memory is allocated for these variable whenever an object is created | Memory is allocated for these variable at the time of loading of the class. |
| 3 | Memory is allocated multiple time whenever a new object is created. | Memory is allocated for these variable only once in the program. |
| 4 | Non-static variable also known as instance variable while because memory is allocated whenever instance is created. | Memory is allocated at the time of loading of class so that these are also known as class variable. |
| 5 | Non-static variable are specific to an object | Static variable are common for every object that means there memory location can be sharable by every object reference or same class. |
| 6 | Non-static variable can access with object reference. Syntax obj\_ref.variable\_name | Static variable can access with class reference. Syntax class\_name.variable\_name |