UNIT-2 (Internet Address)

InetAddress Class: (Imp)

- It is part of **java.net** package used representation IP address, both IPv4 and IPv6.
- It can resolve hostname to IP address and vice versa.

TCP (Transmission Control Protocol)

- (TCP) is a communications standard that enables application programs and computing devices to exchange messages over a network.
- It is designed to send packets across the internet and ensure the successful delivery of data and messages over networks.
- Ensuring it gets delivered to the destination application or device that IP has defined.

IP (Internet Protocol) defines how to **address** and **route** each packet to make sure it reaches the right destination. Each gateway computer on the network checks this IP address to determine where to forward the message.

Creating InetAddress Object

There are no public constructors in the **InetAddress** class. Instead, **InetAddress** has **Static Factory Methods** that connect to a **DNS** (Domain Name Server) server to resolve a **hostname** (IP address to Name). The most common is **InetAddress.getByName()**. For example, this is how you look up www.google.com:

```
Import java.net.*;

InetAddress obj=InetAddress.getByName("www.samriddhi.com.np");
      System.out.println(obj);//www.samriddhi.com.np/181.214.31.79
```

Output:

```
www.samriddhi.com.np/181.214.31.79
```

Steps to Create new InetAddress Object

1. Create a java file named: **InetAddressExample.java**
2. Add header **import java.net.***;
3. Declare object of **InetAddress** with calling **getByName("www.samriddhi.com.np")** static method.
   InetAddress address = InetAddress.getByName("**www.samriddhi.com.np**");
4. Print the result using
   System.out.println(address);

You can also do a **reverse** lookup by IP address. For example, if you want the hostname for the address `181.214.31.79`, pass the **dotted quad** address to InetAddress.getByName()

```
import java.net.*;
public class App {
    public static void main(String[] args) throws UnknownHostException {
      InetAddress obj=InetAddress.getByName("www.samriddhi.com.np");
      System.out.println(obj);//www.samriddhi.com.np/181.214.31.79

    }
```

Output:

www.samriddhi.com.np/181.214.31.79

Commonly used methods of InetAddress class:(important)

Factory Method:

- **getByName(String host):** creates an InetAddress object based on the provided hostname.
- **getByAddress(byte[] addr):** returns an InetAddress object from a byte array of the raw IP address.
- **getAllByName(String host):** returns an array of InetAddress objects from the specified hostname, as a hostname can be associated with several IP addresses.
- **getLocalHost():** returns the address of the localhost.

Getter Method:

- **getHostAddress():** it returns the IP address in string format.
- **getHostname():** it returns the host name of the IP address.
- **getCanonicalHostName():**return fully qualified domain name for this IP.
- **getAddress()** :returns a multiple the dotted quad format of the IP address.

Example 1. Find the address and hostname of the local machine.

```
import java.net.*;
public class App {
    public static void main(String[] args) throws UnknownHostException {
      InetAddress address = InetAddress.getLocalHost();
      System.out.println(address.getHostAddress());
    }
}
```

Output:

192.168.1.68

If, for some reason, you need all the addresses of a host, call **getAllByName()** instead, which returns an array:

```
InetAddress[] addresses = InetAddress.getAllByName("www.google.com");
for (InetAddress address : addresses)
{
System.out.println(address);
}
```

Output:

www.google.com/142.250.192.164

www.google.com/2404:6800:4002:816:0:0:0:2004

Getter Method:

The InetAddress class contains four getter methods that return the hostname as a string and the IP address as both a string and a byte array:

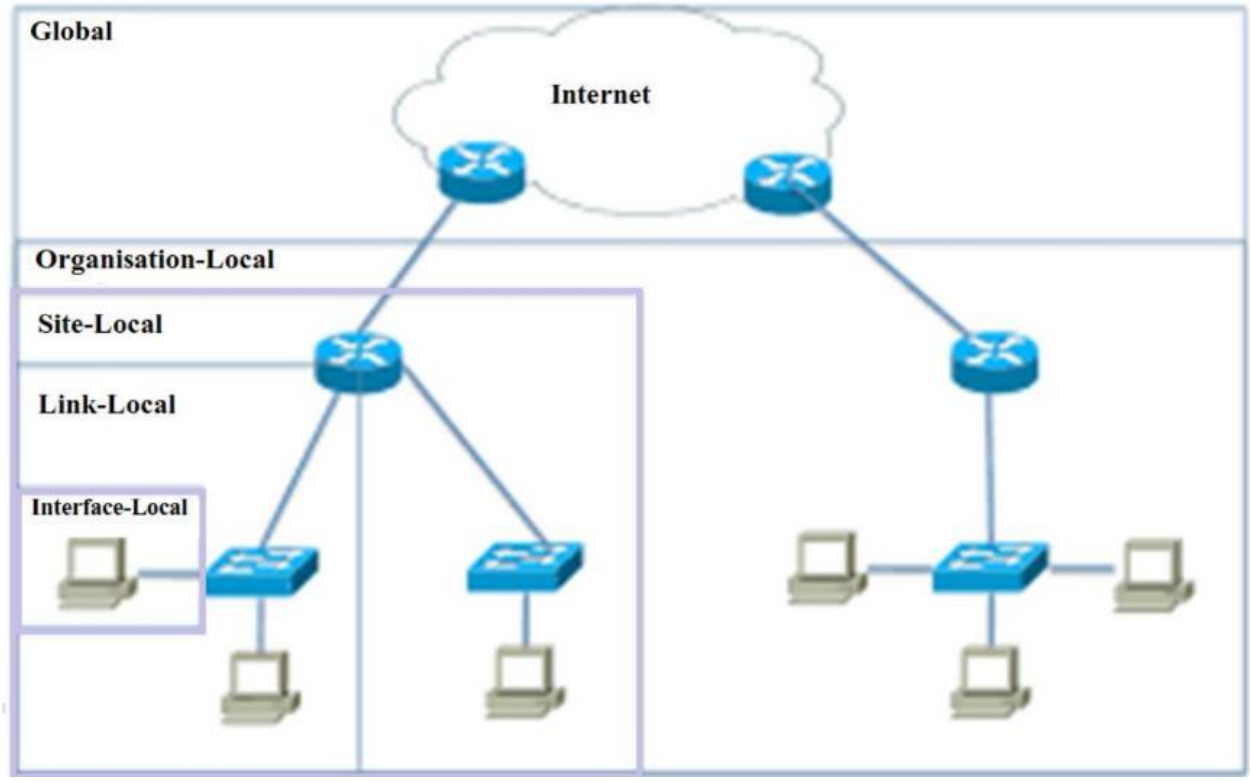
public String getHostName();//returns a String the name of the host with the IP address.
public String getCanonicalHostName()//return fully qualified domain name for this IP.
public byte[] getAddress() // returns a multiple the dotted quad format of the IP address.
public String getHostAddress()// return plain format ipaddress.

Write a java program to find canonicalHostName, HostName, IpAddress of www.iost.com.np

```
import java.net.*;
public class App {
    public static void main(String[] args) throws UnknownHostException {
      InetAddress address = InetAddress.getByName("www.google.com.np");
      System.out.println(address.getCanonicalHostName());
      System.out.println(address.getHostAddress());
      System.out.println(address.getHostName());
      byte[] arr =address.getAddress();
      for(byte b : arr)
      {
      System.out.println(b);
      }
    }
}
```

Q. **Write a program to retrieve Hostname, IP of Local Machine.**

ADDRESS TYPES



1. public boolean isAnyLocalAddress( )
2. public boolean isLoopbackAddress( )
3. public boolean isLinkLocalAddress( )
4. public boolean isSiteLocalAddress( )
5. public boolean isMulticastAddress( )
6. public boolean isMCGlobal( )
7. public boolean isMCNodeLocal( )
8. public boolean isMCLinkLocal( )
9. public boolean isMCSiteLocal( )
10. public boolean isMCOrgLocal( )

The **isAnyLocalAddress()** method returns true if the address is a wildcard address, false otherwise.

- This is important if the system has **multiple network interfaces**, as might be the case on a system with multiple Ethernet cards.

- **In IPv4, the wildcard address is 0.0.0.0. In IPv6, this address is 0:0:0:0:0:0:0:0**

The **isLoopbackAddress()** method returns true if the address is the loopback address, false otherwise.

- Used for testing and inter-process communication on the local machine. The address 127.0.0.1 is commonly known as "localhost."
- **In IPv4, this address is 127.0.0.1. In IPv6, this address is 0:0:0:0:0:0:0:1**

The **isLinkLocalAddress()** method returns true if the address is an IPv6 link-local address, false otherwise.

- Automatically assigned addresses used for communication within a single network segment when no DHCP (Dynamic Host Configuration Protocol) server is available.
- All link local addresses begin with the eight bytes **FE80:0000:0000:0000.** The next eight bytes are filled with a local address, often copied from the Ethernet MAC address

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**. The next eight bytes are filled with a local address, often copied from the Ethernet MAC address assigned by the Ethernet card manufacturer.

The i**sMulticastAddress()** method returns true if the address is a multicast address, false otherwise.

- Used to send data to multiple destinations simultaneously.
- 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 FF00.**

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. In IPv6, global multicast addresses begin with **FF0E or FF1E**

The **isMCOrgLocal()** method returns true if the address is an organization-wide mul- ticast 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**,

The **isMCSiteLocal()** method returns true if the address is a site-wide multicast ad- dress, 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**

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 trans- mitted within their own subnet. Link-local multicast addresses begin with **FF02 or FF12**,

Example:

InetAddress address = InetAddress.getByName("127.0.0.1");

if (address.isLoopbackAddress()) {

System.out.println(address + " is a loopback address.");

Output:

/127.0.0.1 is a loopback address.

IPV4 Address

```
public static void main(String[] args) throws UnknownHostException
    {

      Inet4Address ip1=(Inet4Address) InetAddress.getByName("www.google.com");
      Inet4Address ip2=(Inet4Address) InetAddress.getByName("www.yahoo.com");
      System.out.println("Host Address:"+ip1.getHostAddress());
      System.out.println("IsLoopbackAddress:"+ip1.isLoopbackAddress());
      System.out.println("IsMulticastAddress:"+ip1.isMulticastAddress());
      System.out.println("Ip1==ip2:"+ip1.equals(ip2));
```

```
}
```

Output:

Host Address:142.250.77.228

IsLoopbackAddress:false

IsMulticastAddress:false

Ip1==ip2:false

IPV6 Address

```
public static void main(String[] args) throws UnknownHostException
 {

  String host="localhost";
  byte add[]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
  Inet6Address ip1=Inet6Address.getByAddress(host, add, 5);
  Inet6Address ip2=Inet6Address.getByAddress(null, add, 6);

  System.out.println("Host Address:"+ip1.getHostAddress());
  System.out.println("isLoopbackAddress:"+ip1.isLoopbackAddress());
  System.out.println("ip1==ip2:"+ip1.equals(ip2));


}
```

Output:

Host Address:0:0:0:0:0:0:0:1%5

isLoopbackAddress:true

ip1==ip2:true

Write a java program to check Loopbackaddress,Private address,multicast address, anylocaladdress.

```
import java.net.*;
public class App {
    public static void main(String[] args) throws UnknownHostException {
        InetAddress address = InetAddress.getByName("127.0.0.1");
            if (address.isLoopbackAddress()) {
                System.out.println(address + " is a loopback address.");
            }
            address = InetAddress.getByName("192.168.1.1");
            if (address.isSiteLocalAddress()) {
                System.out.println(address + " is a private address.");
            }
            address = InetAddress.getByName("224.0.0.1");
            if (address.isMulticastAddress()) {
                System.out.println(address + " is a multicast address.");
            }
            address = InetAddress.getByName("0.0.0.0");
            if (address.isAnyLocalAddress()) {
                System.out.println(address + " is the unspecified address.");
            }
    }
}
```

/127.0.0.1 is a loopback address.

/192.168.1.1 is a private address.

/224.0.0.1 is a multicast address.

/0.0.0.0 is the unspecified address.

Testing Reachability:

The InetAddress class has two **isReachable**() methods that test whether a particular node is reachable from the current host

Connections can be blocked for many reasons, including **firewalls**, **proxy servers**, **misbehaving routers**, and **broken cables**, or simply because the remote host is **not turned** on when you try to connect.

public boolean isReachable(int timeout) throws IOException

public boolean isReachable(NetworkInterface interface, int ttl, int timeout) throws IOException

Ping is a network tool to test whether a particular host is reachable across an IP network. This network can be IPv4 or IPv6. The standard ping works by sending

ICMP (**Internet Control Message Protocol)** packets to the host and listening for ICMP replies.

An IOException will be thrown if there's a network error.

- Discover if a host is up and running.
- Check to see if a webserver is up and running (TCP ping)
- Network performance analysis.

Example:

```
  public static void main(String[] args) throws IOException
    {
      System.out.print(InetAddress.getByName("192.168.0.1").isReachable(30));
    }
Output: true
```

Example:

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

                NetworkInterface networkInterface =
NetworkInterface.getByName("127.0.0.1");
                int ttl = 30; // Set TTL
                int timeout = 5000; // Set timeout in milliseconds
                boolean reachable =
InetAddress.getByName("192.168.56.1").isReachable(networkInterface,ttl,timeout);
                System.out.println("Is reachable: " + reachable);

    }
}
```

Object Methods:

java.net.InetAddress inherits from **java.lang.Object**

It overrides three methods to provide more specialized behavior:

```
 public boolean equals(Object o)//check two inetaddress is
same
 public int hashCode()//return int calculated soley from ip
address
 public String toString()  //short text representation of
object
```

Example:

```
import java.net.*;
public class App {
    public static void main(String[] args) throws Exception {
        try {
            InetAddress g = InetAddress.getByName("www.google.com");
            InetAddress g1 = InetAddress.getByName("www.google.com.np");
            System.out.println(g.toString());
            System.out.println(g.hashCode());
            System.out.println(g1.hashCode());
            if(g.equals(g1)) {
            System.out.println("www.google.com same www.google.com.np");
            }
            else {
            System.out.println("www.google.com  not same www.google.com.np");
            }
            } catch (UnknownHostException ex) {
                System.out.println("Host lookup failed.");
            }
    }
}
```

Output:

```
www.google.com/142.250.194.164
-1896168796
-1896169213
www.google.com  not same www.google.com.np
```

Inet4Address and Inet6Address:

Java uses two classes, Inet4Address and Inet6Address, in order to distinguish IPv4 addresses from IPv6 addresses

```
public final class Inet4Address extends InetAddress
public final class Inet6Address extends InetAddress
```

Most of the time, you really shouldn't be concerned with whether an address is an IPv4 or IPv6 address.

Inet4Address overrides several of the methods in **InetAddress** but doesn't change their behavior in any public way. **Inet6Address** is similar, but it does add one new method not present in the superclass, **isIPv4CompatibleAddress():**

```
public boolean isIPv4CompatibleAddress()
```

Network Interface Class:

- **NetworkInterface** class to get information about the network interfaces installed on your machine
- In today's computer multiple network card installed on a single machine.
- Sometimes we might need get information like how many network cards are installed and details of each network card .
- For example: You might want to find out an if an interface is down before starting a server application.
- So Network interface is the point of interconnection between a computer on a private or public network.
- A network interface is generally a **network interface card** a network interface card is a circuit board or card that is installed in a computer so that it can be connected to network.
- The network interface class method are **static** and therefore you cannot create a new of this class.
- Network interface class provides methods to enumerate all the local of the interface and to create **inetaddress** object from them this inetaddress objects can than be used to create **sockets, server sockets** and so forth.
- You can ask for a network interface by ip address, byname or by enumeration.

Factory Method

- public static NetworkInterface getByName(String name) throws SocketException
- public static NetworkInterface getByInetAddress(InetAddress address) throws SocketException
- public static Enumeration getNetworkInterfaces() throws SocketException

1. public static NetworkInterface getByName(String name) throws SocketException

The getByName() method returns a NetworkInterface object representing the **network interface** with the **particular name**. If there's no interface with that name, it returns null.

Easier to use in linux system than windows you can easily find the interface name in ip the ipconfig command output.

```
NetworkInterface networkInterface=NetworkInterface.getByName("eth3");
        if(networkInterface!=null)
        {
            System.out.println("NIC Name:"+networkInterface.getName());
            System.out.println("NIC Display
Name:"+networkInterface.getDisplayName());
        }
        else
        {
            System.out.println("Network Interface is Not Found");
        }
```

Output:

NIC Name:eth3

NIC Display Name:WAN Miniport (Network Monitor)

2. public static NetworkInterface getByInetAddress(InetAddress address) throws SocketException

The **getByInetAddress()** method returns a **NetworkInterface** object representing the network interface bound to the specified IP address. If no network interface is bound to that IP address on the local host, it returns null. If anything goes wrong, it throws a **SocketException**. For example, this code fragment finds the network interface for the local loopback address:

```
public static void main(String[] args) throws UnknownHostException,
SocketException {
        InetAddress address=InetAddress.getByName("192.168.1.68");
        NetworkInterface
networkInterface=NetworkInterface.getByInetAddress(address);
        if(networkInterface!=null)
        {
            System.out.println("NIC Name:"+networkInterface.getName());
            System.out.println("NIC Display
Name:"+networkInterface.getDisplayName());
        }
        else
        {
            System.out.println("Network Interface is Not Found");
        }
    }
```

Output:

NIC Name:wlan1

NIC Display Name:Qualcomm QCA9377 802.11ac Wireless Adapter

public static Enumeration getNetworkInterfaces() throws SocketException

The getNetworkInterfaces() method returns a java.util.Enumeration **listing all the network interfaces on the local host.**

Lab: Write a java program to list all the network interfaces on the localhost.

Example:

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

public class App {
    public static void main(String[] args) throws IOException {
           Enumeration<NetworkInterface> interfaces =
NetworkInterface.getNetworkInterfaces( );
                while (interfaces.hasMoreElements( ))
                {
                    NetworkInterface ni = (NetworkInterface)
interfaces.nextElement( );
                    System.out.println(ni);
                }

    }
}
```

Getter Method

- public Enumeration getInetAddresses()
- public String getName()
- public String getDisplayName()

Once you have a **NetworkInterface** object, you can inquire about its IP address and name. This is pretty much the only thing you can do with these objects.

1. public Enumeration getInetAddresses()

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

public class App {
    public static void main(String[] args) throws IOException {
        NetworkInterface eth0 = NetworkInterface.getByName("eth1");
        Enumeration<InetAddress> addresses = eth0.getInetAddresses();
        while (addresses.hasMoreElements()) {
         System.out.println(addresses.nextElement());
    }
```

```
}
}
```

2. public String getName()

The getName() method returns the name of a particular NetworkInterface object, such as eth0 or eth1.

3. public String getDisplayName()

The getDisplayName() method allegedly returns a more human-friendly name for the particular NetworkInterface—something like "Ethernet Card 0." On Windows, you may see slightly friendlier names such as "Local Area Connection" or "Local Area Connection 2."

SpamCheck:

A Java program that checks if the hosts connecting to your network are known spammers or not.

The way RBLs (Real-time Blackhole List) work is by keeping a list of IP addresses that are known to be spammy. Other mail servers around the world can look up these addresses by using simple Domain Name Service (DNS) queries. **domain name system**, is the phonebook of the Internet, connecting web browsers with websites.

Example:

```
import java.net.*;

public class App {
    public static final String BHSERVICE = "sbl.spamhaus.org";
    public static void main(String[] args) throws UnknownHostException {
        for (String arg: args) {
            if (isKnownSpammer(arg)) {
                System.out.println(arg + " is a known spammer.");
            }
            else {
                System.out.println(arg + " doesn't appear to be a spammer.");
            }

        }

    }

    private static boolean isKnownSpammer(String arg) {
```

```
        try {
            InetAddress address = InetAddress.getByName(arg);
            byte[] quad = address.getAddress();
            String query = BHSERVICE;
            for (byte octet : quad) {
                int unsignedByte = octet < 0 ? octet + 256 : octet;
                query = unsignedByte + "." + query;
```

// `octet < 0 ? octet + 256 : octet;`: This is a ternary conditional operation that checks if the value of `octet` is less than 0. If true, it adds 256 to `octet`; otherwise, it leaves `octet` unchanged.

```
            }
            InetAddress.getByName(query);
            return true;
        } catch (UnknownHostException e) {
            return false;
        }
    }

}
```

```
Output:
PS E:\BCAAdvanceJava\Tutorial\SwingFormExample\src> java App  207.34.56.23
125.12.32.4 94.181.33.149
207.34.56.23 doesn't appear to be a spammer.
125.12.32.4 doesn't appear to be a spammer.
94.181.33.149 is a known spammer.
```

Processing web server Logfiles

- Web server logs track the hosts that access a website.
- By default, the log reports the IP addresses of the sites that connect to the server.
- Most web servers have an option to store hostnames instead of IP addresses, but this can hurt performance

Most web servers have standardized on the common logfile format. A typical line in the common logfile format looks like this:

205.160.186.76 unknown - [17/Jun/2013:22:53:58 -0500] "GET /bgs/greenbg.gif HTTP 1.0" 200 50

This line indicates that a web browser at IP address 205.160.186.76 requested the file /bgs/greenbg.gif from this web server at 11:53 P.M (and 58 seconds) on June

17, 2013. The file was found (response code 200) and 50 bytes of data were successfully transferred to the browser