Chapter: 2

Internet Addresses

Contents:

- The InetAddress Class: Creating New Inet Address Object, Getter
- Method, Address Types, Testing Reachability and Object Methods
- Inet4Address and Inet6Address
- The Network Interface Class: Factory Method & Getter Method
- Some Useful Programs: SpamCheck, Processing Web Server Logfiles

IP Address:

- An **IP address**, or **Internet Protocol address**, is a **numerical label** assigned to devices that are connected to a computer network.
- The IP address serves as a **unique identifier** that enables devices to communicate with each other over the network.

There are two main types of IP addresses:

- IPv4
- IPv6.

IPv4:

- (Internet Protocol version 4) addresses are **32-bit binary numbers**.
- It consists of 4 numbers separated by the dots.
- Each number can be from 0-255 in decimal numbers. But computers do not understand decimal numbers,

they instead change them to binary numbers which are only 0 and 1. Therefore, in binary, this (0-255) range can be written as (00000000 – 11111111).

- A total of **(2^32) devices approximately** = 4,294,967,296 can be **assigned** with IPv4.
- IP Address range: 0.0.0.0 to 255.255.255.255
- An example of an IPv4 address is 192.0.2.1.

IPv6:

- (Internet Protocol version 6) addresses are **128-bit binary numbers**, which are typically represented in hexadecimal format.
- IPv6 is written as a group of **8 hexadecimal numbers** separated with **colons(:)**
- A total of **(2^128) devices** can be assigned with **unique addresses** which are actually more than enough for upcoming future generations.

- An example of an IPv6 address is 2001:0db8:85a3:0000:0000:8a2e:0370:7334.

InetAddress:

- The **InetAddress** class is a Java class that represents an IP address, which can be either an IPv4 or IPv6 address.
- The **java.net.InetAddress** class provides methods to get the IP of any host name.
- For example www.google.com, www.facebook.com, etc.
- An IP address is represented by 32-bit or 128-bit.
- InetAddress can **handle** both **IPv4 and IPv6 addresses**.

There are two types of addresses:

- ➢ Unicast - An identifier for a **single interface**.
- ➢ Multicast -An identifier for a **set of interfaces**.

Creating new InetAddress Objects:

Steps:

1. Import the **java.net.InetAddress** class:

   ```
   import java.net.InetAddress;
   ```

2. Declare object of InetAddress e.g. address and call the static method getByName(“domain_name”) of InetAddress

   ```
   InetAddress address =
   InetAddress.getByName("www.example.com");
   ```

3. Print the result

   ```
   System.out.println(address);
   ```

Note: that the **getByName()** method can throw an **UnknownHostException** if the host name is not valid. You should handle this exception appropriately in your code.

Example:

```
import java.io.*;
import java.net.*;
public class InetAddressExample{
public static void main(String[] args){
try{
InetAddress ip=InetAddress.getByName("www.google.
com");

System.out.println("Host Name: "+ip.getHostName());
System.out.println("IP Address: "+ip.getHostAddress())
;
}catch(Exception e){System.out.println(e);}
}
}
```

Getter Methods:

1. **getHostName()**: Returns the host name associated with the IP address as a string. If the host name is not available, the method returns the IP address in textual form.
2. **getHostAddress()**: Returns the IP address in textual form as a string.
3. **getAddress()**: Returns the raw IP address as a byte array. For IPv4 addresses, the array will contain four bytes; for IPv6 addresses, it will contain 16 bytes.
4. **getCanonicalHostName()**: Returns the fully qualified domain name (FQDN) of the host associated with the instance of InetAddress class.

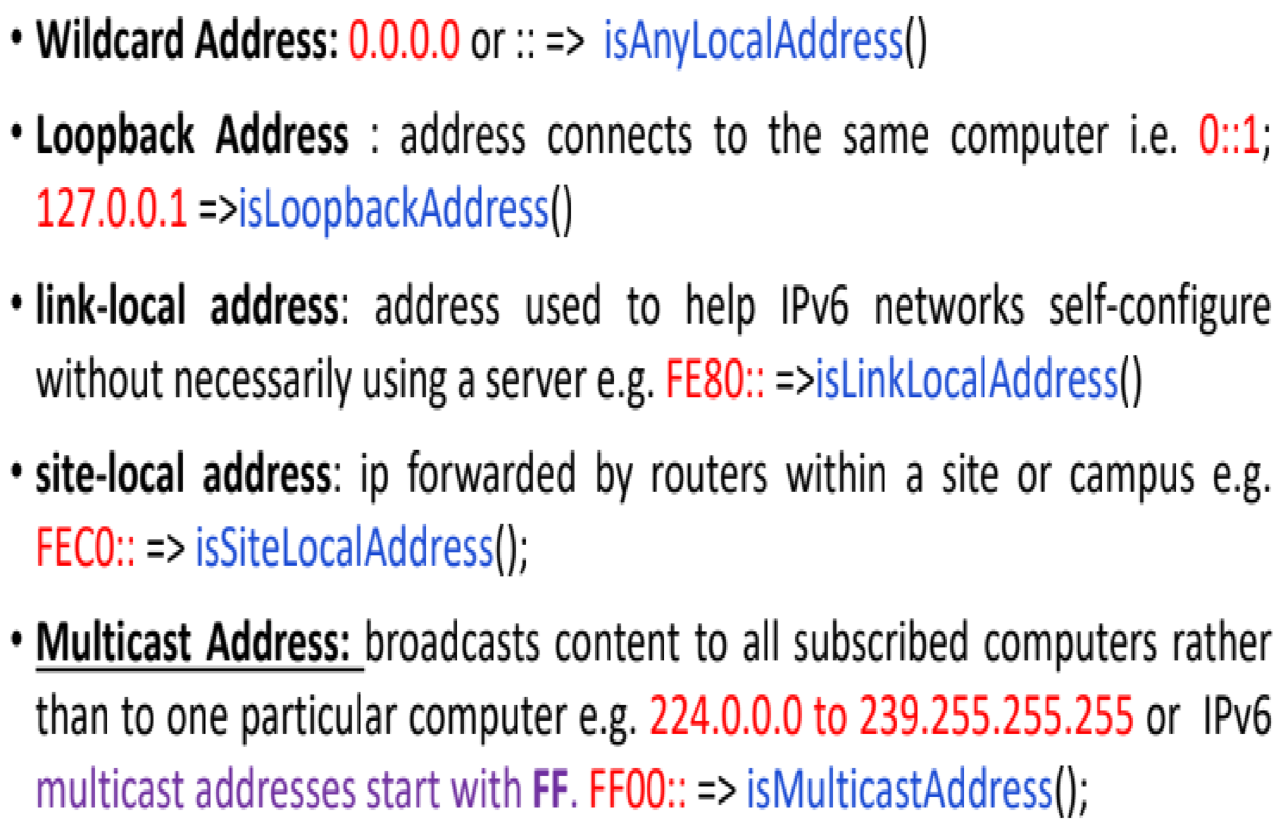
Example:

```
import java.io.*;
import java.net.*;
public class InetAddressExample{
public static void main(String[] args){
try{
InetAddress ip=InetAddress.getByName("www.
google.com");

System.out.println("Host Name: "+ip.getHostNa
me());
System.out.println("IP Address: "+ip.getHostAd
dress());
System.out.println("Canonical Host
Name: "+ip.getCanonicalHostName());
System.out.println("IP Address: "+ip.getHostAd
dress());

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

Address Types:



- **Wildcard Address:** 0.0.0.0 or :: => isAnyLocalAddress()
- **Loopback Address** : address connects to the same computer i.e. 0::1; 127.0.0.1 =>isLoopbackAddress()
- **link-local address**: address used to help IPv6 networks self-configure without necessarily using a server e.g. FE80:: =>isLinkLocalAddress()
- **site-local address**: ip forwarded by routers within a site or campus e.g. FEC0:: => isSiteLocalAddress();
- **Multicast Address:** broadcasts content to all subscribed computers rather than to one particular computer e.g. 224.0.0.0 to 239.255.255.255 or IPv6 multicast addresses start with **FF**. FF00:: => isMulticastAddress();

- **global multicast address:** subscribers around the world e.g. 224.0.2.1; begins FF0E or FF1E => isMCGlobal();
- **organization-wide multicast address**: subscribers within all the sites of a company or organization e.g. begin with FF08 or FF18, => isMCOrgLocal()
- **site-wide multicast address**: Packets addressed to a site-wide address will only be transmitted within their local site e.g. 224.0.2.1; FF05 or FF15 => isMCSiteLocal();
- **subnet-wide multicast address:** Packets addressed to a link-local address will only be transmitted within their own subnet e.g. FF02 or FF12 => isMCLinkLocal();
- **interface-local multicast address:** 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 e.g. FF01 or FF11 => isMCNodeLocal();

Methods:

- isAnyLocalAddress()
- isLoopbackAddress()
- isLinkLocalAddress()
- isMCGlobal()
- isMCLinkLocal()
- isMCNodeLocal()
- isMCOrgLocal()
- isMCSiteLocal()
- isMulticastAddress():

Testing Reachability:

- Connections can be blocked for many reasons, including firewalls, proxy server, misbehaving routers, and broken cables, or simply because the remote host is not turned on
- Testing whether a particular node is reachable from the current host (i.e., whether a network connection can be made)
- If the host responds within timeout milliseconds, the methods return true; otherwise, they return false.
- **Methods:**

  public boolean isReachable(int timeout) throws IOException

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

  ttl(time to live) - the maximum number of network hops the connection will attempt before being discarded.

Inet4Address:

Example:

```
import java.io.*;
import java.net.*;
public class Inet4AddressExample{
public static void main(String[] args){
try{
//Get the Inet4Address object for a given IP
address string
Inet4Address ip=(Inet4Address)Inet4Address.get
ByName("192.168.0.1");

System.out.println("Host Name: "+ip.getHostNa
me());
System.out.println("IP Address: "+ip.getHostAd
dress());

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

Inet6Address:

This class represents IPv6 address and extends the InetAddress class.

Methods of this class provide facility to represent and interpret IPv6 addresses. **Methods of this class takes input in the following formats:**

1. **x:x:x:x:x:x:x:x** –This is the general form of IPv6 address where each x can be replaced with a 16 bit hexadecimal value of the address.

   For example:

   4B0C:0:0:0:880C:99A8:4B0:4411

2. When the address contains multiple set of 8 bits as '0'. In such cases '::' is replaced in place of 0's to make the address shorter.

   For example:

4B0C::880C:99A8:4B0:4411

3. **x:x:x:x:x:x:d.d.d.d** –A third format is used when hybrid addressing(IPv6 + IPv4) has to be taken care of. In such cases the first 12 bytes are used for IPv6 addressing and remaining 4 bytes are used for IPv4 addressing.

   For example:

   2001:0db8:85a3::8a2e:0370:192.168.0.1

InetAddress – Factory Methods:

- The **InetAddress** class is used to encapsulate both, the numerical IP address and the domain name for that address.
- The InetAddress class has no visible constructors.
- The InetAddress class has the inability to create objects directly, hence **factory methods** are used for the purpose.

- Factory Methods are **static methods** in a class that return an object of that class.

Method	**Description**
public static InetAddress getLocalHost() *throws UnknownHostException*	This method returns the instance of InetAddress containing the local hostname and address.
public static InetAddress getByName (String host) *throws UnknownHostException*	This method returns the instance of InetAddress containing IP and Host name of host represented by **host** argument.

Method	**Description**
public static InetAddress[] getAllByName(String hostName) *throws UnknownHostException*	This method returns the array of the instance of InetAddress class which contains IP addresses.
public static InetAddress getByAddress(byte IPAddress[]) *throws UnknownHostException*	This method returns an InetAddress object created from the raw IP address.
public static InetAddress getByAddress(String hostName, byte IPAddress[]) *throws UnknownHostException*	This method creates and returns an InetAddress based on the provided hostname and IP address.

Example:

```
        // To get and print InetAddress of Local Host
        InetAddress address1 = InetAddress.getLocalHost();
        System.out.println("InetAddress of Local Host : "
                           + address1);
        // To get and print InetAddress of Named Host
        InetAddress address2
            = InetAddress.getByName("45.22.30.39");
        System.out.println("InetAddress of Named Host : "
                           + address2);
        // To get and print ALL InetAddresses of Named
Host
        InetAddress address3[]
            = InetAddress.getAllByName("172.19.25.29");
        for (int i = 0; i < address3.length; i++) {
            System.out.println(
                "ALL InetAddresses of Named Host : "
                + address3[i]);
        }

        // To get and print InetAddresses of Host with
specified IP Address
        byte IPAddress[] = { 125, 0, 0, 1 };
        InetAddress address4
            = InetAddress.getByAddress(IPAddress);
        System.out.println(
            "InetAddresses of Host with specified IP Address
: "
            + address4);

        // To get and print InetAddresses of Host with
specified IP Address and                hostname
        byte[] IPAddress2
            = { 105, 22, (byte)223, (byte)186 };
        InetAddress address5 = InetAddress.getByAddress(
            "gfg.com", IPAddress2);
        System.out.println(
            "InetAddresses of Host with specified IP Address
and hostname : "
            + address5);
```

Output:

```
InetAddress of Local Host : localhost/127.0.0.1

InetAddress of Named Host : /45.22.30.39

ALL InetAddresses of Named Host : /172.19.25.29

InetAddresses of Host with specified IP Address :
/125.0.0.1

InetAddresses of Host with specified IP Address and
hostname : gfg.com/105.22.223.186
```

SpamCheck:

- A number of services monitor spammers, and inform clients whether a host attempting to connect to them is a known spammer or not.
- These real-time blackhole lists need to respond to queries extremely quickly, and process a very high load.
- Thousands, maybe millions, of hosts query them repeatedly to find out whether an IP address attempting a connection is or is not a known spammer.
- To find out if a certain IP address is known spammer, reverse the byte of the address, add the domain of the blackhole service, and look it up. If the address is found, it’s a spammer. If it isn’t, it’s not.
- For instance, if you want to ask sbl.spamhaus.org if 192.168.0.1 is a spammer, you would look up the hostname 1.0.168.192.sbl.spamhaus.org.

- If the DNS query succeeds, if it returns the address 127.0.0.2 then the host is known to be a spammer. Otherwise it isn't.

Example:

```
public class SpamCheckIp {

   public static boolean isSpam(String ipAddress) {
       // Reverse the IP address and append the blacklist
domain
       String blacklistDomain = "sbl.spamhaus.org";
       String reversedIpAddress = new
StringBuilder(ipAddress).reverse().toString();
       String query = reversedIpAddress + "." +
blacklistDomain;

       try {
           // Perform a DNS lookup on the query string
           InetAddress address =
InetAddress.getByName(query);

           // Return true if the IP address is listed in the
blacklist
```

```
            return
address.getHostAddress().equals("127.0.0.2");
        } catch (UnknownHostException ex) {
            // The DNS lookup failed, so assume the IP
address is not listed
            return false;
        }
    }

    public static void main(String[] args) {
        String ipAddress = "192.0.2.1";
        boolean isSpam = isSpam(ipAddress);
        System.out.println(“Is spam \n”+isSpam);
    }
}
```

Output:

```
Is spam?

False
```

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.
- They can be used to analyze user behavior, track website performance, and identify potential security issues.

Example:

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

public class WebLogTest {

public static void main(String[] args) {
String file = "logfile.txt";
try (FileInputStream fin = new FileInputStream(file);
Reader in = new InputStreamReader(fin);
BufferedReader bin = new BufferedReader(in);) {

for (String entry = bin.readLine(); entry != null; entry = bin.readLine())
{
// separate out the IP address
int index = entry.indexOf(' '); // position of the first space
String ip = entry.substring(0, index); // read IP
String theRest = entry.substring(index); // reads remain (extra
information)

// Ask DNS for the hostname and print it out
try {
InetAddress address = InetAddress.getByName(ip);
System.out.println(address.getHostName() + theRest);
} catch (UnknownHostException ex) {
System.err.println(entry);
}
}
} catch (IOException ex) {
System.out.println("Exception: " + ex);
}
}
}
```

Sample Log:

```
205.160.186.76      unknown

[11/Mar/2023:22:53:76      -0500]      "GET

/bgs/greenbg.gif HTTP 1.0"   200  50
```