

# The InetAddress Class

By

Khushbu Kumari Sarraf

EMBA,BE

The bottom right corner of the slide features several decorative concentric circles in a lighter shade of blue, resembling ripples on water.

# The InetAddress Class

- InetAddress class provides methods to get the IP address of any hostname. An IP address is represented by 32-bit or 128-bit unsigned number. InetAddress can handle both IPv4 and IPv6 addresses.
- Java InetAddress class represents an IP address. The java.net.InetAddress class provides methods to get the IP of any host name for example [www.javatpoint.com](http://www.javatpoint.com), [www.google.com](http://www.google.com), [www.facebook.com](http://www.facebook.com), etc.

# The InetAddress Class

The `java.net.InetAddress` class is Java's encapsulation of an IP address. It is used by most of the other networking classes, including `Socket`, `ServerSocket`, `URL`, `DatagramSocket`, `DatagramPacket`, and more. Usually, it includes both a hostname and an IP address.

`InetAddress` class represents an Internet address as two fields: `hostName` (a `String`) and `address` (an `int`). `hostName` contains the name of the host; for example, [www.javapoint.com](http://www.javapoint.com), 104.21.79.8 address contains the 32-bit IP address.

## Creating new InetAddress Objects

There are no public constructors in `InetAddress` class. Instead, `InetAddress` has static factory methods that connect to a DNS server to resolve a hostname. The most common is `InetAddress.getByName()`.

```
InetAddress address= InetAddress.getByName("www.javapoint.com")
```

## A program that prints the address of [www.javapoint.com](http://www.javapoint.com)

```
import java.net.*;
public class JavaInternetAddressByName {
public static void main (String[] args) {
try {
InetAddress
address=InetAddress.getByName("www.javapoint.com");
System.out.println(address);
} catch (UnknownHostException ex) {
System.out.println("Could not find www.javapoint.com");
}
```

# The InetAddress Class

If we want hostname for the address 104.21.79.8, pass the dotted address to `InetAddress.getByName()`.

```
InetAddress address =InetAddress.getByName("104.21.79.8");  
System.out.println(address.getHostName());
```

**Note:** If the address you look up does not have a hostname, `getHostName()` simply returns the dotted address you supplied.

# The InetAddress Class

- `getLocalHost(). getHostAddress()` method of `InetAddress` to get the IP Address of our machine in our local network.
- `getByName()` method of `InetAddress` to get the IP Address of a specific Domain Name.

# Java InetAddress Class Methods

Method	Description
<code>public static InetAddress getByName(String host) throws UnknownHostException</code>	It returns the instance of InetAddress containing LocalHost IP and name.
<code>public static InetAddress getLocalHost() throws UnknownHostException</code>	It returns the instance of InetAddress containing local host name and address.
<code>public String getHostName()</code>	It returns the host name of the IP address.
<code>public String getHostAddress()</code>	It returns the IP address in string format.

# InetAddress: Address Types

- Some IP address have special meanings. For instance, 127.0.0.1 is the local loopback address. IPv4 address in the range 224.0.0.0 to 239.255.255.255 are multicast address. Java includes different methods for testing whether an InetAddress object meets any of these criteria.
- `public boolean IsAnyLocalAddress()`
- `public boolean isLoopbackAddress()`
- `public boolean IsLinkLocalAddress()`
- `Public boolean isSiteLocalAddress()`
- `public boolean isMulticastAddress()`
- `public boolean isMCGlobal()`
- `public boolean is MCNodeLocal()`
- `Public boolean is MCLinkLocal()`
- `Public boolean is MCSiteLocal()`
- `Public boolean is MCOrgLocal()`



```
public class InetDemo2
{
    public static void main(String[] arg) throws Exception
    {
        InetAddress ip = InetAddress.getByName("www.javapoint.com");
        InetAddress ip1[] = InetAddress.getAllByName("www.javapoint.com");
        byte addr[]={72, 3, 2, 12};
        System.out.println("ip : "+ip);
        System.out.print("\nip1 : "+ip1);
        InetAddress ip2 = InetAddress.getByAddress(addr);
        System.out.print("\nip2 : "+ip2);
        System.out.print("\nAddress : " +Arrays.toString(ip.getAddress()));
        System.out.print("\nHost Address : " +ip.getHostAddress());
        System.out.print("\nisAnyLocalAddress : " +ip.isAnyLocalAddress());
        System.out.print("\nisLinkLocalAddress : " +ip.isLinkLocalAddress());
        System.out.print("\nisLoopbackAddress : " +ip.isLoopbackAddress());
        System.out.print("\nisMCGlobal : " +ip.isMCGlobal());
        System.out.print("\nisMCLinkLocal : " +ip.isMCLinkLocal());
        System.out.print("\nisMCNodeLocal : " +ip.isMCNodeLocal());
        System.out.print("\nisMCOrgLocal : " +ip.isMCOrgLocal());
        System.out.print("\nisMCSiteLocal : " +ip.isMCSiteLocal());
        System.out.print("\nisMulticastAddress : " +ip.isMulticastAddress());
        System.out.print("\nisSiteLocalAddress : " +ip.isSiteLocalAddress());
    }
}
```

# InetAddress Class

- InetAddress can handle both IPv4 and IPv6 address.
- InetAddress Class is used to encapsulate both the numerical IP address and the domain name for the address.

Factory Method	Instance Method
<ul style="list-style-type: none"><li>• <code>getLocalHost()</code></li><li>• <code>getByName()</code></li><li>• <code>getAllByName()</code></li><li>• <code>getByAddress()</code></li></ul>	<ul style="list-style-type: none"><li>• <code>getAddress()</code></li><li>• <code>getHostAddress()</code></li><li>• <code>getHostName()</code></li></ul>

# Factory Method

```
Import java.net.*;  
Class InetAddressTest{  
    public static void main(String args[]) throws  
UnknownHostException{  
        //To get and print InetAddress of LocalHost  
        InetAddress ad1=InetAddress.getLocalHost();  
        System.out.println(ad1);  
  
        //To get and print  InetAddress of Named Host  
        InetAddress ad2 =InetAddress.getByName("www.microsoft.com");  
        System.out.println(ad2);
```

## Factory Method Cont.

//To get and print All InetAddress of Named Host

```
InetAddress ad3[]=InetAddress.getAllByName("www.google.com");  
for(int i=0;i<ad3.length;i++)  
{  
System.out.println(ad3[i]);  
}
```

//To get and print InetAddress of Host with specified IP Address

```
byte ipAddr[] ={127,0,0,1};  
InetAddress ad4 =InetAddress.getByAddress(ipAddr);  
System.out.println(ad4);  
}
```

# Instance Method

- The `InetAddress` class has plenty of instance methods that can be called using object.

```
Import java.net.*;
Class InetAddressDemo{
    public static void main(String args[]) throws UnknownHostException {
        InetAddress ad1 = InetAddress.getByName("www.microsoft.com");
        InetAddress ad2 = InetAddress.getByName("www.microsoft.com");
        InetAddress ad3 = InetAddress.getByName("www.google.com");
        System.out.println(ad1.equals (ad2));
        System.out.println(ad1.equals (ad3));
        System.out.println(ad1.getHostAddresss());
        System.out.println(ad1.getHostName());
        System.out.println(ad1.getAddress());
    }
}
```

# Testing Reachability

The `InetAddress` class has two `isReachable()` methods that enable apps to test whether a particular node is reachable from the current host; that is, whether a network connection can be made. 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. The `isReachable()` methods allow you to test the connection:

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

# Testing Reachability

**isReachable()** : Returns true if this address is reachable. This method is used generally as a pre-condition in various programs, to avoid Host Unreachable exceptions in future.

**Syntax** : `public boolean isReachable(int timeout)` throws `IOException`

**Parameters :**

`timeout` : time after which the call aborts, resulting in false value.

**Throws :**

`IOException` : if network error occurs

# Testing Reachability

Another overloaded `isReachable()` method specify the network interface to be used while checking for reachability and the `ttl` parameter specifies the number of hops the echo packet makes before exiting the network.

## Syntax :

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

## Parameters :

`netif` : Network interface to use

`ttl` : time to live in milliseconds

`timeout` : time after which the call aborts, resulting in false value.

## Throws :

`IOException` : if network error occurs



# Object Methods

Like every other class, `java.net.InetAddress` inherits from `java.lang.Object`.

Thus, it has access to all the methods of that class.

It overrides three methods to provide more specialized behavior:

```
public boolean equals(Object o)
```

```
public int hashCode( )
```

```
public String toString( )
```

# Object Methods Cont.

```
public boolean equals(Object o)
```

An Object is equal to an InetAddress object only if it is itself an instance of the InetAddress class and it has the same IP address. It does not need to have the same hostname.

```
public int hashCode( )
```

The hashCode( ) method returns an int calculated from the IP address. It does not take the hostname into account. If two InetAddress objects have the same address, then they have the same hashCode, even if their hostnames are different.

```
public String toString( )
```

This method returns a short representation of the object.  
i.e hostname/ dotted quad address

# 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. In the application layer where Java programs reside, you simply don't need to know this (and even if you do need to know, it's quicker to check the size of the byte array returned by `getAddress()` than to use `instanceof` to test which subclass you have.

# Inet4Address and Inet6Address

## 1. IPv4

- IPv4 is the primary Internet protocol. It is the first version of IP deployed for production in the ARPANET in 1983. It is a widely used IP version to differentiate devices on network using an addressing scheme. A 32-bit addressing scheme is used to store  $2^{32}$  addresses that is more than 4 billion addresses.

## 2. IPv6

- IPv6 is the latest version of Internet protocol. It aims at fulfilling the need of more internet addresses. It provides solutions for the problems present in IPv4. It provides 128-bit address space that can be used to store  $2^{128}$  addresses. IPv6 is also identified with a name IPng (Internet Protocol next generation).

# Inet4Address and Inet6Address

```
public static int getVersion(InetAddress ia){  
    byte[] address = ia.getAddress();  
    if (address.length==4)  
        return 4;  
    else if (address.length ==16) return 6;  
    return 16;  
    else return -1;
```

# Network Interface

- A network interface can be thought of as a point at which your computer connects to the network. It is not necessarily a piece of hardware but can also be implemented in a software. For example a loopback interface which is used for testing purposes. The loop back interface (127.0.0.1 for IPv4 and ::1 for IPv6) is not a physical device but a piece of software simulating a network interface. The loop back interface is commonly used in test environments.
- Network interface class represents a Network Interface made up of a name, and a list of IP addresses assigned to this interface. It is used to identify the local interface. Interfaces are normally known by names such as "lo0".

# Network Interface

- The `NetworkInterface` class represents a local IP address. This can either be a physical interface such as an additional Ethernet card (common on firewalls and routers) or it can be a virtual interface bound to the same physical hardware as the machine's other IP addresses. The `NetworkInterface` class provides methods to enumerate all the local addresses, regardless of interface.

# Factory Methods

- Since `NetworkInterface` objects represent physical hardware and virtual addresses, they cannot be constructed arbitrarily. As with the `InetAddress` class, there are static factory methods that return the `NetworkInterface` object associated with a particular network interface. You can ask for a `NetworkInterface` by IP address, by name, or by enumeration.



# Network Interface : Methods

**1.getName() :** Returns the name of this network interface.

*Syntax : public String getName()*

**2.getInetAddresses() :** Returns an enumeration of all Inetaddresses bound to this network interface, if security manager allows it.

*Syntax : public Enumeration getInetAddresses()*

# Network Interface : Methods

**3.getInterfaceAddresses()** : Returns a list of all interface addresses on this interface.

*Syntax :public List getInterfaceAddresses()*

**4.getSubInterfaces()** : Returns an enumeration of all the sub or virtual interfaces of this network interface. For example, eth0:2 is a sub interface of eth0.

*Syntax :public Enumeration getSubInterfaces()*

The background of the slide features several decorative concentric circles in a lighter blue shade, located primarily in the bottom right and bottom center areas.

# Network Interface : Methods

**5.getParent() :** In case of a sub interface, this method returns the parent interface. If this is not a subinterface, this method will return null.

*Syntax :public NetworkInterface getParent()*

**6.getIndex() :** Returns the index assigned to this network interface by the system. Indexes can be used in place of long names to refer to any interface on the device.

*Syntax :public int getIndex()*

# Network Interface : Methods

**7.getDisplayName() :** This method returns the name of network interface in a readable string format.

*Syntax :public String getDisplayName()*

**8.getByName() :** Finds and returns the network interface with the specified name, or null if none exists.

*Syntax :public static NetworkInterface getByName(String name)*  
*throws SocketException*

**Parameters :name :** name of network interface to search for.

**Throws : SocketException :** if I/O error occurs.

# 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.
- The nature of the problem requires that the response be fast, and ideally it should be cacheable. Furthermore, the load should be distributed across many servers, ideally ones located around the world. Although this could conceivably be done using a web server, SOAP, UDP, a custom protocol, or some other mechanism, this service is in fact cleverly implemented using DNS and DNS alone.

# SpamCheck

- To find out if a certain IP address is a known spammer, reverse the bytes 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 `207.87.34.17` is a spammer, you would look up the hostname `17.34.87.207.sbl.spam-haus.org`. (Note that despite the numeric component, this is a hostname ASCII string, not a dotted quad IP address.)
- If the DNS query succeeds, then the host is known to be a spammer. If the lookup fails—that is, it throws an `UnknownHostException`—it isn't.

# Spam Check Program

## Example 4-9. SpamCheck

```
import java.net.*;

public class SpamCheck {

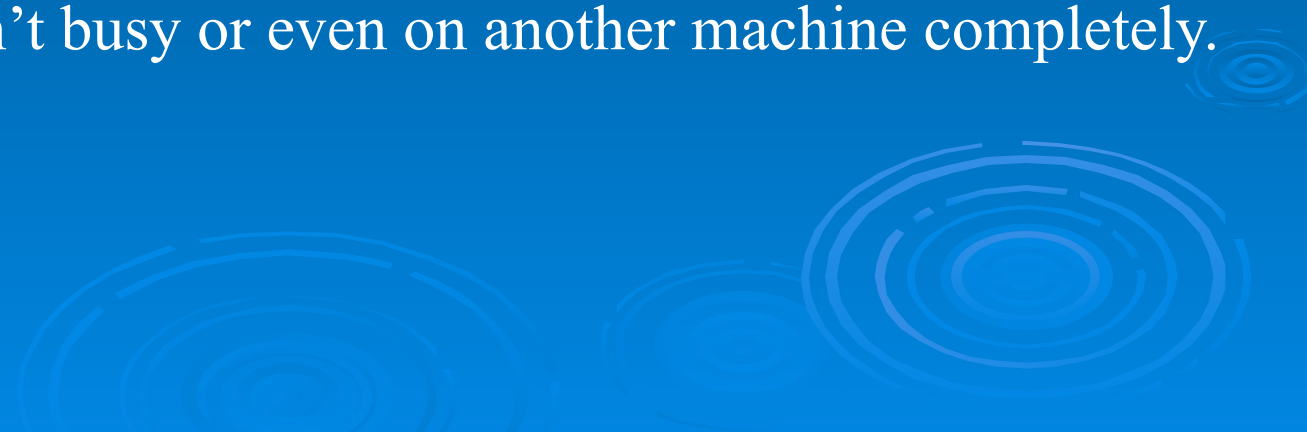
    public static final String BLACKHOLE = "sbl.spamhaus.org";

    public static void main(String[] args) throws UnknownHostException {
        for (String arg: args) {
            if (isSpammer(arg)) {
                System.out.println(arg + " is a known spammer.");
            } else {
                System.out.println(arg + " appears legitimate.");
            }
        }
    }

    private static boolean isSpammer(String arg) {
        try {
            InetAddress address = InetAddress.getByName(arg);
            byte[] quad = address.getAddress();
            String query = BLACKHOLE;
            for (byte octet : quad) {
                int unsignedByte = octet < 0 ? octet + 256 : octet;
                query = unsignedByte + "." + query;
            }
            InetAddress.getByAddress(query);
            return true;
        } catch (UnknownHostException e) {
            return 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. However, you can often get more information from the names of those sites than from their IP addresses. Most web servers have an option to store hostnames instead of IP addresses, but this can hurt performance because the server needs to make a DNS request for each hit. It is much more efficient to log the IP addresses and convert them to hostnames at a later time, when the server isn't busy or even on another machine completely.

The bottom right corner of the slide features a decorative graphic consisting of several concentric, semi-transparent circles in a lighter shade of blue, resembling ripples in water.



# Processing Web Server Logfiles

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. The first field is the IP address or, if DNS resolution is turned on, the hostname from which the connection was made. This is followed by a space. Therefore, for our purposes, parsing the logfile is easy: everything before the first space is the IP address, and everything after it does not need to be changed. The dotted quad format IP address is converted into a hostname using the usual methods of `java.net.InetAddress`.

# Processing Web Server Logfiles

- `Weblog` is more efficient than you might expect. Most web browsers generate multiple logfile entries per page served, because there's an entry in the log not just for the page itself but for each graphic on the page. And many visitors request multiple pages while visiting a site. DNS lookups are expensive and it simply doesn't make sense to look up each site every time it appears in the logfile. The `InetAddress` class caches requested addresses. If the same address is requested again, it can be retrieved from the cache much more quickly than from DNS.

process web  
server logfiles

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

public class Weblog {

    public static void main(String[] args) {
        try (FileInputStream fin = new FileInputStream(args[0]);
            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(' ');
                String ip = entry.substring(0, index);
                String theRest = entry.substring(index);

                // 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);
        }
    }
}
```

# Processing Web Server Logfiles

- The name of the file to be processed is passed to Weblog as the first argument on the command line. A `FileInputStream` `fin` is opened from this file and an `InputStreamReader` is chained to `fin`. This `InputStreamReader` is buffered by chaining it to an instance of the `BufferedReader` class. The file is processed line by line in a for loop.
- Each pass through the loop places one line in the `String` variable `entry`. `entry` is then split into two substrings: `ip`, which contains everything before the first space, and `theRest`, which is everything from the first space to the end of the string. The position of the first space is determined by `entry.indexOf(" ")`. The substring `ip` is converted to an `InetAddress` object using `getByName()`. `getHostName()` then looks up the host- name. Finally, the `hostname` and everything else on the line ( `theRest`) are printed on `System.out`. Output can be sent to a new file through the standard means for redirecting output.