Internet Address

Gopal Maharjan

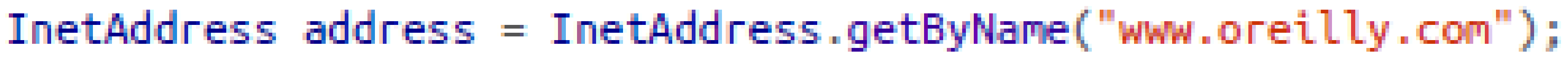
Faculty

The InetAddress Class

- The java.net.InetAddress class is Java’s high-level representation of an IP address, both IPv4 and IPv6.
- 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.

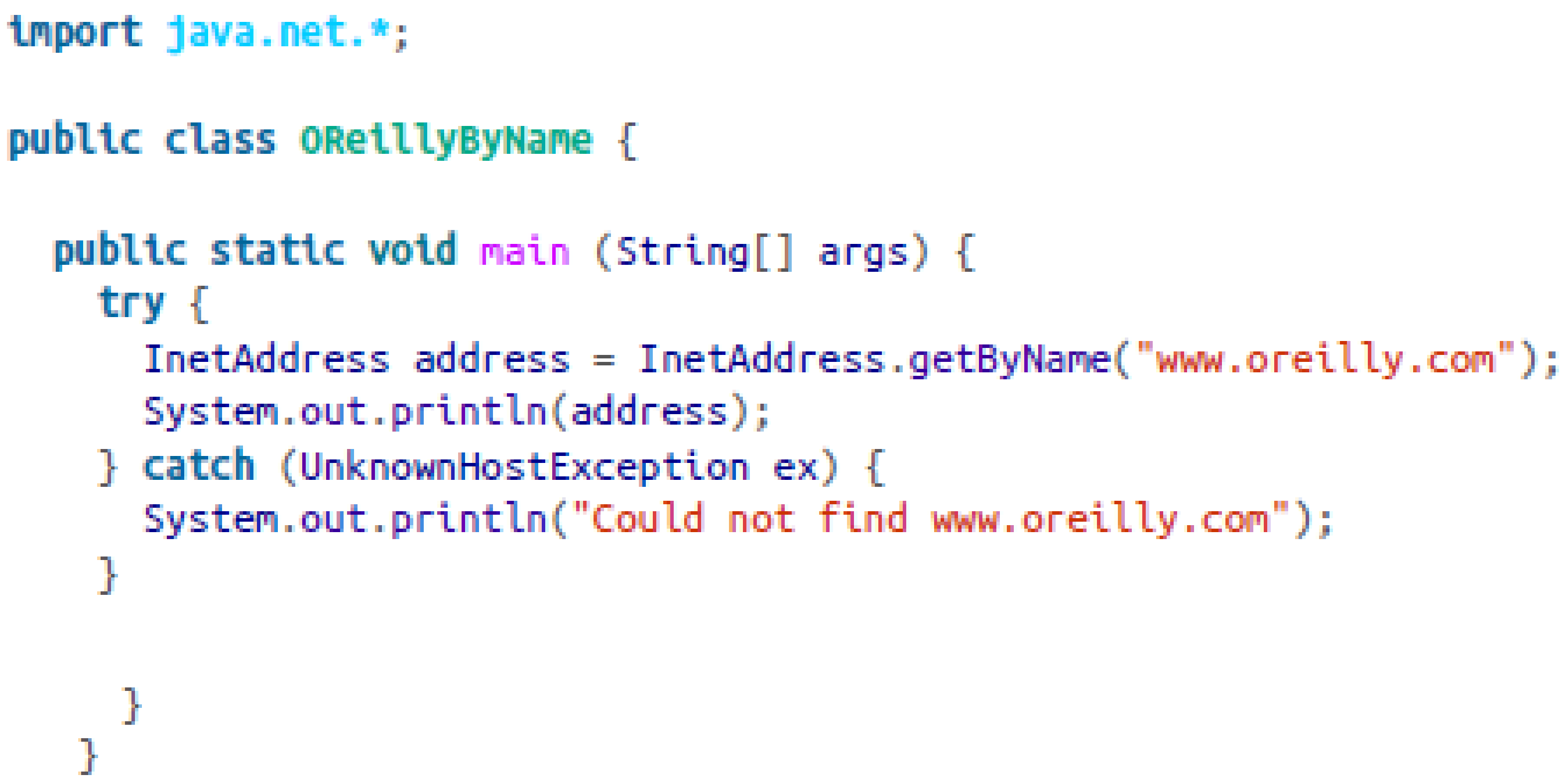
Creating New InetAddress Objects

- There are no public constructors in the InetAddress class.
- Instead, InetAddress has static factory methods that connect to a DNS server to resolve a hostname.
- The most common is InetAddress.getByName().
- For example, this is how you look up *www.oreilly.com*:
- It actually makes a connection to the local DNS server to look up the name and the numeric address. (If you've looked up this host previously, the information may be cached locally, in which case a network connection is not required.)
- If the DNS server can't find the address, this method throws an UnknownHostException, a subclass of IOException.

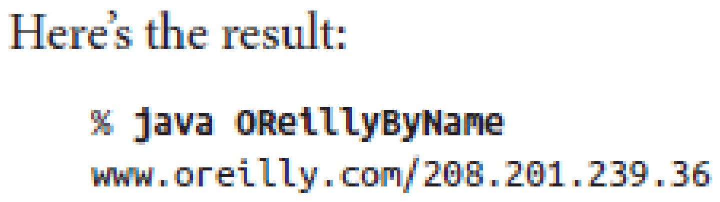


```java
InetAddress address = InetAddress.getByName("www.oreilly.com");
```

- *A program that prints the address of www.oreilly.com*



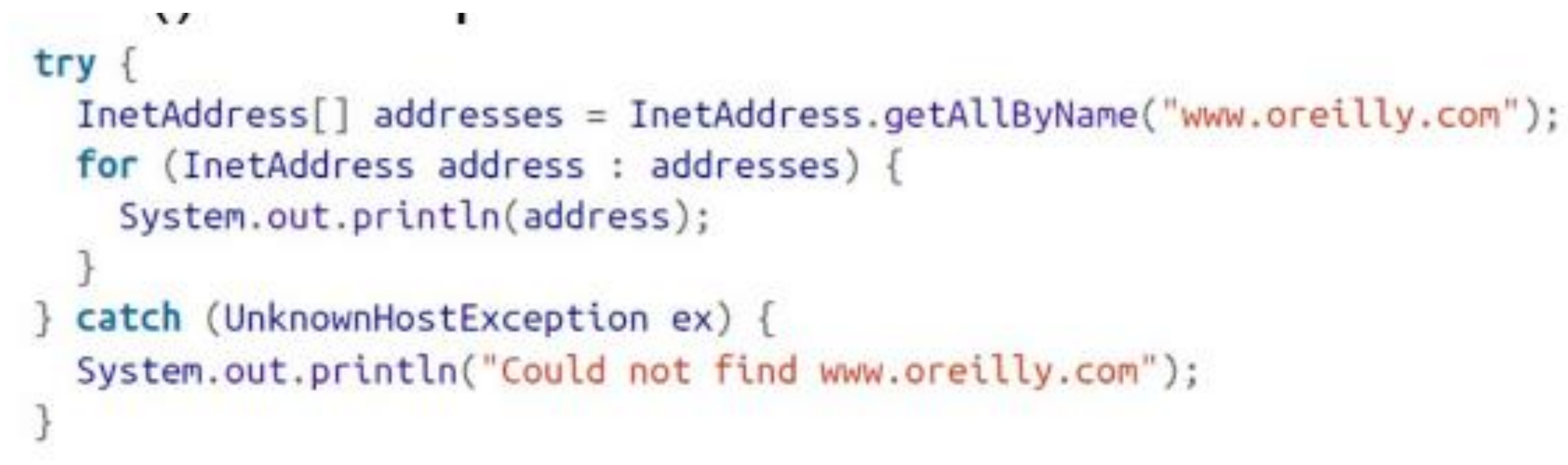
```
import java.net.*;

public class OReillyByName {

  public static void main (String[] args) {
    try {
      InetAddress address = InetAddress.getByName("www.oreilly.com");
      System.out.println(address);
    } catch (UnknownHostException ex) {
      System.out.println("Could not find www.oreilly.com");
    }


     }
   }
```



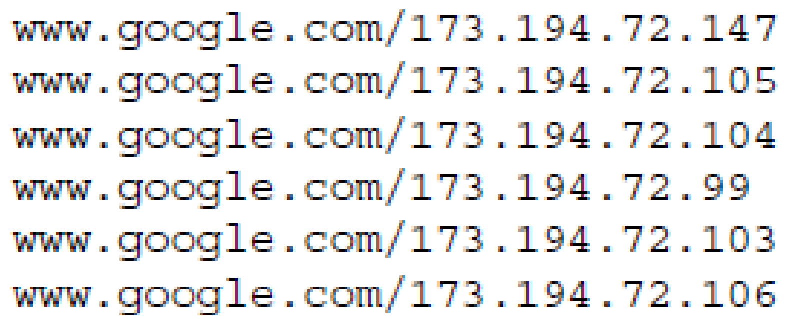
Here's the result:

```
% java OReillyByName
www.oreilly.com/208.201.239.36
```

getAllByName(): lookup all the addresses of a host

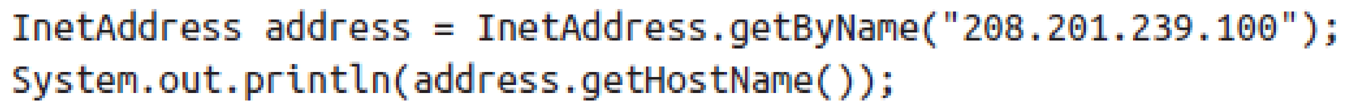


```
try {
  InetAddress[] addresses = InetAddress.getAllByName("www.oreilly.com");
  for (InetAddress address : addresses) {
    System.out.println(address);
  }
} catch (UnknownHostException ex) {
  System.out.println("Could not find www.oreilly.com");
}
```



```
www.google.com/173.194.72.147
www.google.com/173.194.72.105
www.google.com/173.194.72.104
www.google.com/173.194.72.99
www.google.com/173.194.72.103
www.google.com/173.194.72.106
```

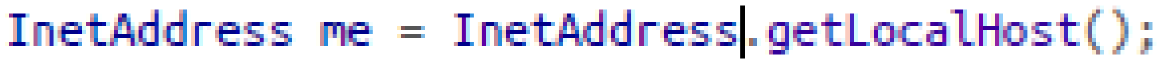
- You can also do a reverse lookup by IP address
- For example, if you want the hostname for the address 208.201.239.100, pass the dotted quad address to InetAddress.getBy Name():



```
InetAddress address = InetAddress.getByName("208.201.239.100");
System.out.println(address.getHostName());
```

- If the address you look up does not have a hostname, getHostName() simply returns the dotted quad address you supplied.

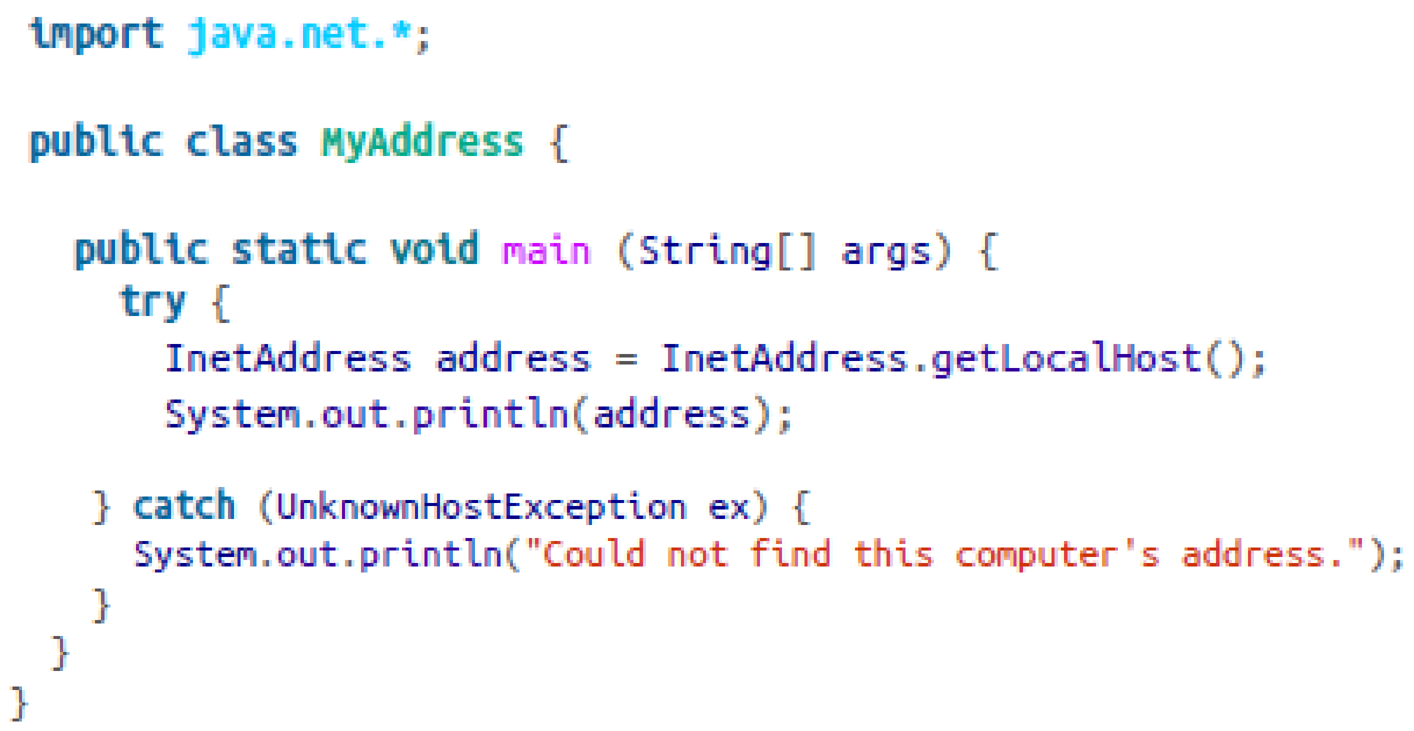
- Finally, the getLocalHost() method returns an InetAddress object for the host on which your code is running:



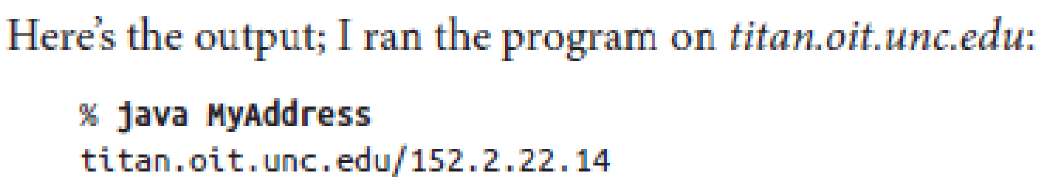
```
InetAddress me = InetAddress.getLocalHost();
```

- This method tries to connect to DNS to get a real hostname and IP address such as “elharo.laptop.corp.com” and “192.1.254.68”; but if that fails it may return the *loopback* address instead. This is the hostname “localhost” and the dotted quad address “127.0.0.1”.

- Print the address of the machine it’s run on
- Find the address of the local machine



```java
import java.net.*;

public class MyAddress {

  public static void main (String[] args) {
    try {
      InetAddress address = InetAddress.getLocalHost();
      System.out.println(address);

    } catch (UnknownHostException ex) {
      System.out.println("Could not find this computer's address.");
    }
  }
}
```



Here’s the output; I ran the program on *titan.oit.unc.edu*:

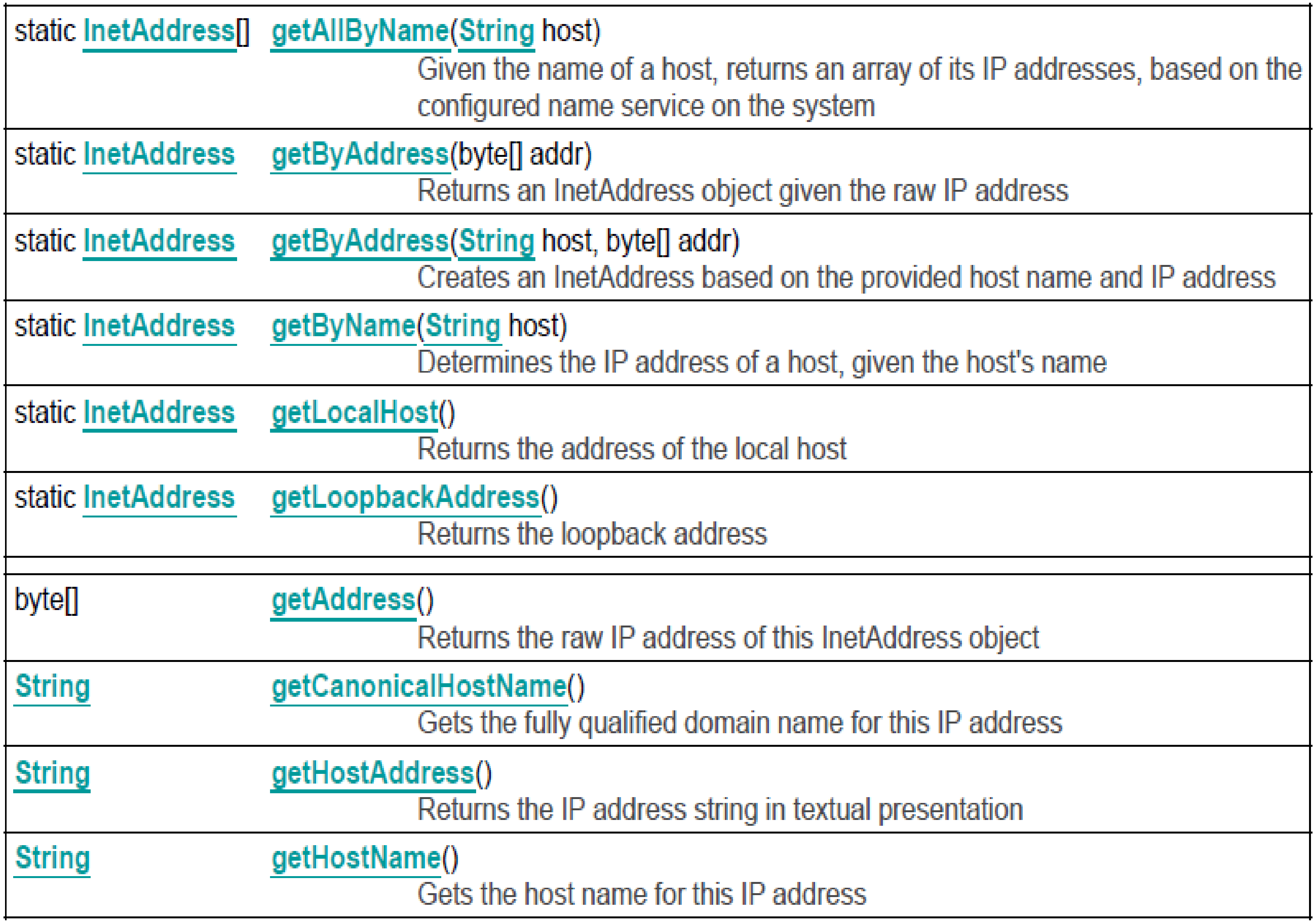
```
% java MyAddress
titan.oit.unc.edu/152.2.22.14
```

- Whether you see a fully qualified domain name like *titan.oit.unc.edu* or a partial name like *titan* depends on what the local DNS server returns for hosts in the local domain. If you’re not connected to the Internet, and the system does not have a fixed IP address or domain name, you’ll probably see *localhost* as the domain name and 127.0.0.1 as the IP address.

Caching

- Because DNS lookups can be relatively expensive (on the order of several seconds for a request that has to go through several intermediate servers, or one that's trying to resolve an unreachable host) the InetAddress class caches the results of lookups.
- Once it has the address of a given host, it won't look it up again, even if you create a new InetAddress object for the same host. As long as IP addresses don't change while your program is running, this is not a problem.
- Java only caches unsuccessful DNS queries for 10 seconds by default
- Times can be controlled and specified in Java security properties
  - networkaddress.cache.ttl: specifies the number of seconds a successful DNS lookup will remain in Java's cache
  - networkaddress.cache.negative.ttl: specifies the number of seconds an unsuccessful lookup will be cached

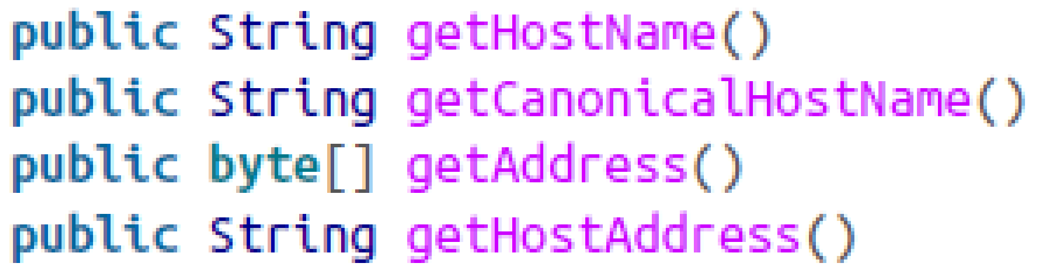
InetAddress: Create Objects and Getter Methods



static InetAddress[]	getAllByName(String host) Given the name of a host, returns an array of its IP addresses, based on the configured name service on the system
static InetAddress	getByAddress(byte[] addr) Returns an InetAddress object given the raw IP address
static InetAddress	getByAddress(String host, byte[] addr) Creates an InetAddress based on the provided host name and IP address
static InetAddress	getByName(String host) Determines the IP address of a host, given the host's name
static InetAddress	getLocalHost() Returns the address of the local host
static InetAddress	getLoopbackAddress() Returns the loopback address
byte[]	getAddress() Returns the raw IP address of this InetAddress object
String	getCanonicalHostName() Gets the fully qualified domain name for this IP address
String	getHostAddress() Returns the IP address string in textual presentation
String	getHostName() Gets the host name for this IP address

Getter Methods

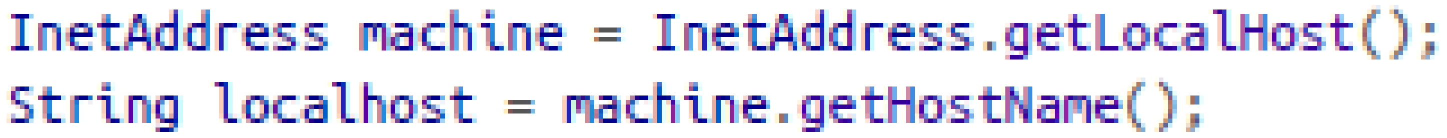
- The InetAddress class contains four getter methods that return the hostname as a string and the IP address as both a string and a byte array:



```java
public String getHostName()
public String getCanonicalHostName()
public byte[] getAddress()
public String getHostAddress()
```

getHostName()

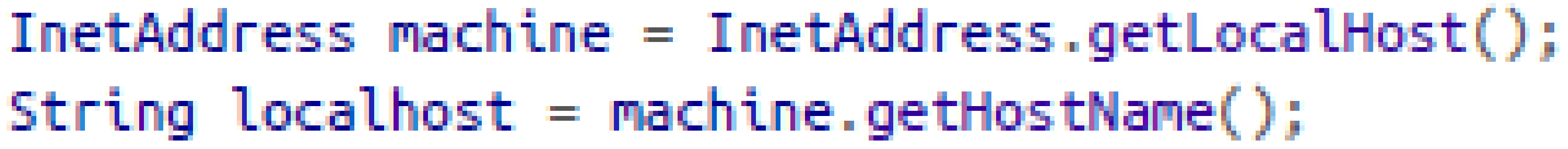
- The getHostName() method returns a String that contains the name of the host with the IP address represented by this InetAddress object.
- If the machine in question doesn’t have a hostname or if the security manager prevents the name from being determined, a dotted quad format of the numeric IP address is returned.



```
InetAddress machine = InetAddress.getLocalHost();
String localhost = machine.getHostName();
```

getCanonicalHostName()

- The getCanonicalHostName() method is similar, but it's a bit more aggressive about contacting DNS.
- getHostName() will only call DNS if it doesn't think it already knows the hostname
- getCanonicalHostName() calls DNS if it can, and may replace the existing cached hostname. For example:
- The getCanonicalHostName() method is particularly useful when you're starting with a dotted quad IP address rather than the hostname.



```
InetAddress machine = InetAddress.getLocalHost();
String localhost = machine.getHostName();
```

- Given the address, find the hostname

```java
import java.net.*;

public class ReverseTest {

  public static void main (String[] args) throws UnknownHostException {
    InetAddress ia = InetAddress.getByName("208.201.239.100");
    System.out.println(ia.getCanonicalHostName());
  }
}
```

Here's the result:

```
% java ReverseTest
oreilly.com
```

getHostAddress()

- The getHostAddress() method returns a string containing the dotted quad format of the IP address.

- *Find the IP address of the local machine*

```
import java.net.*;

public class MyAddress {

  public static void main(String[] args) {
    try {
      InetAddress me = InetAddress.getLocalHost();
      String dottedQuad = me.getHostAddress();
      System.out.println("My address is " + dottedQuad);
    } catch (UnknownHostException ex) {
      System.out.println("I'm sorry. I don't know my own address.");
    }
  }
}
```

Here's the result:

```
% java MyAddress
My address is 152.2.22.14.
```

- If you want to know the IP address of a machine (and you rarely do), then use the getAddress() method, which returns an IP address as an array of bytes in network byte order.

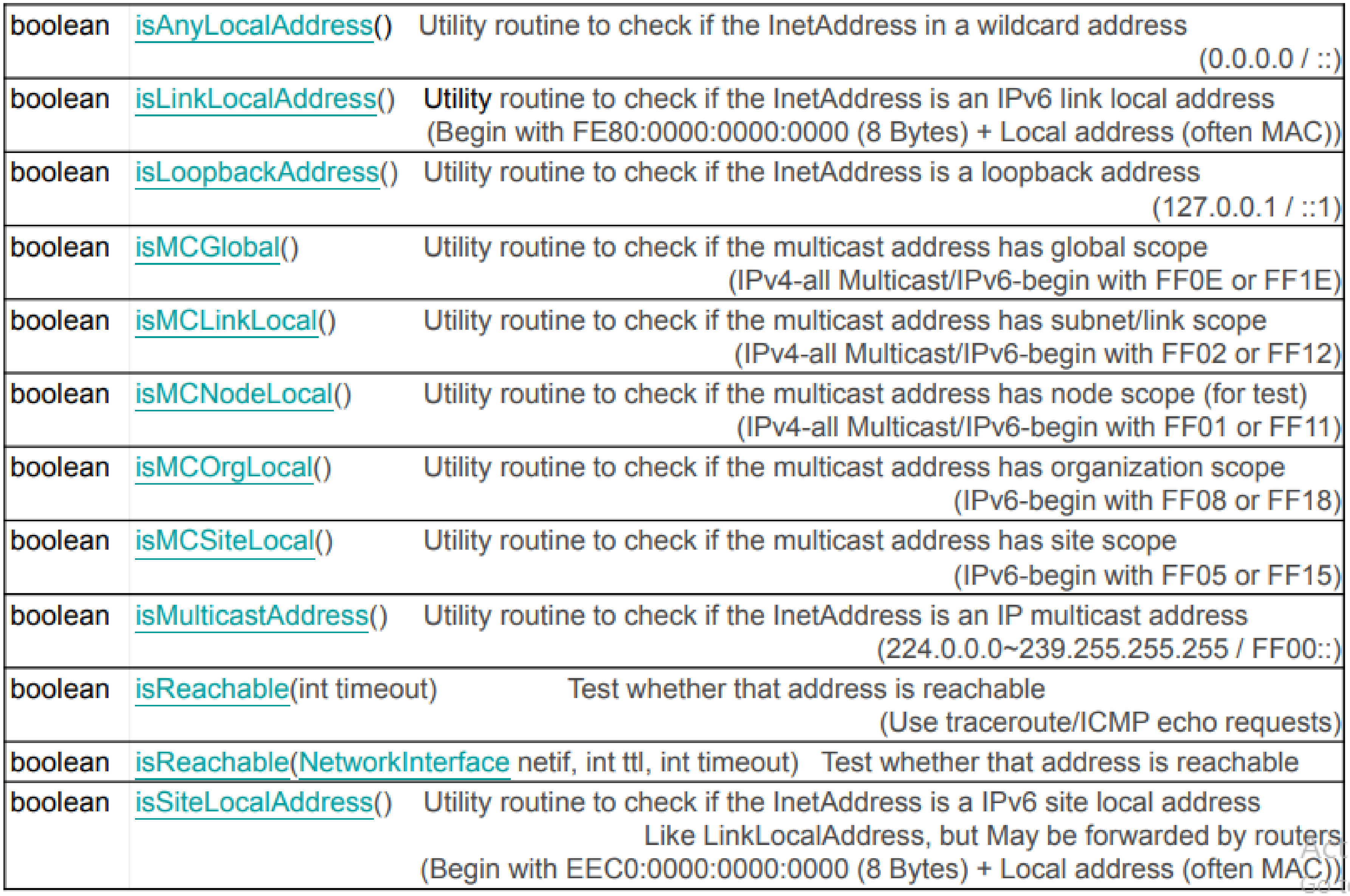
Address Types

- 127.0.0.1 is the local loopback address.
- IPv4 addresses in the range 224.0.0.0 to 239.255.255.255 are multicast addresses that send to several subscribed hosts at once. Java includes 10 methods for testing whether an InetAddress object meets any of these criteria:

Address Types

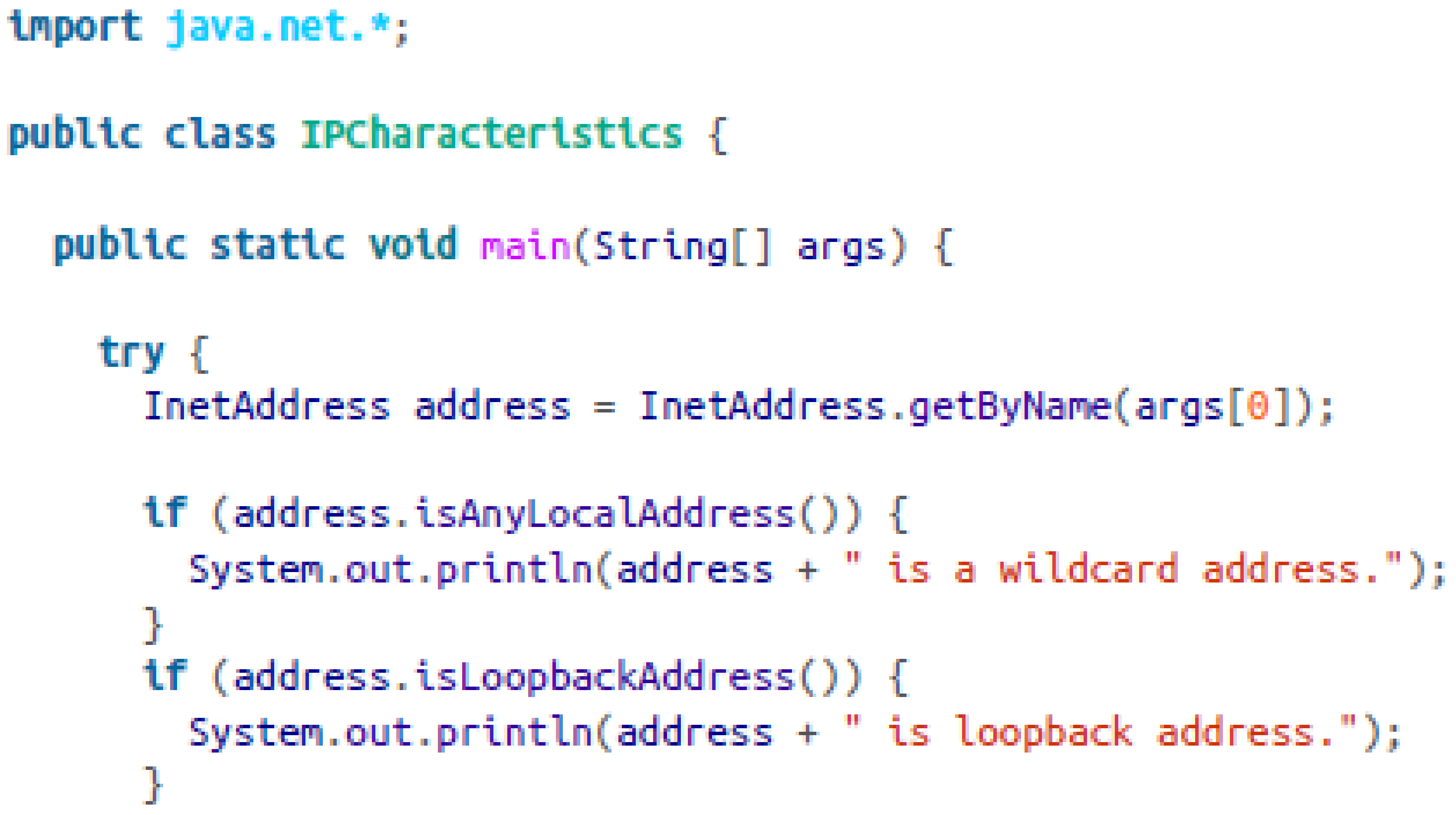
- **public boolean** isAnyLocalAddress()
- **public boolean** isLoopbackAddress()
- **public boolean** isLinkLocalAddress()
- **public boolean** isSiteLocalAddress()
- **public boolean** isMulticastAddress()
- **public boolean** isMCGlobal()
- **public boolean** isMCNodeLocal()
- **public boolean** isMCLinkLocal()
- **public boolean** isMCSiteLocal()
- **public boolean** isMCOrgLocal()

Address Types



boolean	isAnyLocalAddress()	Utility routine to check if the InetAddress in a wildcard address (0.0.0.0 / ::)
boolean	isLinkLocalAddress()	Utility routine to check if the InetAddress is an IPv6 link local address (Begin with FE80:0000:0000:0000 (8 Bytes) + Local address (often MAC))
boolean	isLoopbackAddress()	Utility routine to check if the InetAddress is a loopback address (127.0.0.1 / ::1)
boolean	isMCGlobal()	Utility routine to check if the multicast address has global scope (IPv4-all Multicast/IPv6-begin with FF0E or FF1E)
boolean	isMCLinkLocal()	Utility routine to check if the multicast address has subnet/link scope (IPv4-all Multicast/IPv6-begin with FF02 or FF12)
boolean	isMCNodeLocal()	Utility routine to check if the multicast address has node scope (for test) (IPv4-all Multicast/IPv6-begin with FF01 or FF11)
boolean	isMCOrgLocal()	Utility routine to check if the multicast address has organization scope (IPv6-begin with FF08 or FF18)
boolean	isMCSiteLocal()	Utility routine to check if the multicast address has site scope (IPv6-begin with FF05 or FF15)
boolean	isMulticastAddress()	Utility routine to check if the InetAddress is an IP multicast address (224.0.0.0~239.255.255.255 / FF00::)
boolean	isReachable(int timeout)	Test whether that address is reachable (Use traceroute/ICMP echo requests)
boolean	isReachable(NetworkInterface netif, int ttl, int timeout)	Test whether that address is reachable
boolean	isSiteLocalAddress()	Utility routine to check if the InetAddress is a IPv6 site local address Like LinkLocalAddress, but May be forwarded by routers (Begin with EEC0:0000:0000:0000 (8 Bytes) + Local address (often MAC))

- *Testing the characteristics of an IP address*



```
import java.net.*;

public class IPCharacteristics {

  public static void main(String[] args) {

    try {
      InetAddress address = InetAddress.getByName(args[0]);

      if (address.isAnyLocalAddress()) {
        System.out.println(address + " is a wildcard address.");
      }
      if (address.isLoopbackAddress()) {
        System.out.println(address + " is loopback address.");
      }
```

```
if (address.isLinkLocalAddress()) {
  System.out.println(address + " is a link-local address.");
} else if (address.isSiteLocalAddress()) {
  System.out.println(address + " is a site-local address.");
} else {
  System.out.println(address + " is a global address.");
}

if (address.isMulticastAddress()) {
  if (address.isMCGlobal()) {
    System.out.println(address + " is a global multicast address.");
  } else if (address.isMCOrgLocal()) {
    System.out.println(address
     + " is an organization wide multicast address.");
  } else if (address.isMCSiteLocal()) {
    System.out.println(address + " is a site wide multicast
                       address.");
  } else if (address.isMCLinkLocal()) {
    System.out.println(address + " is a subnet wide multicast
                       address.");
  } else if (address.isMCNodeLocal()) {
    System.out.println(address
     + " is an interface-local multicast address.");
  } else {
    System.out.println(address + " is an unknown multicast
                       address type.");
  }
} else {
  System.out.println(address + " is a unicast address.");
}
```

```
    } catch (UnknownHostException ex) {
      System.err.println("Could not resolve " + args[0]);
    }
  }
}
```

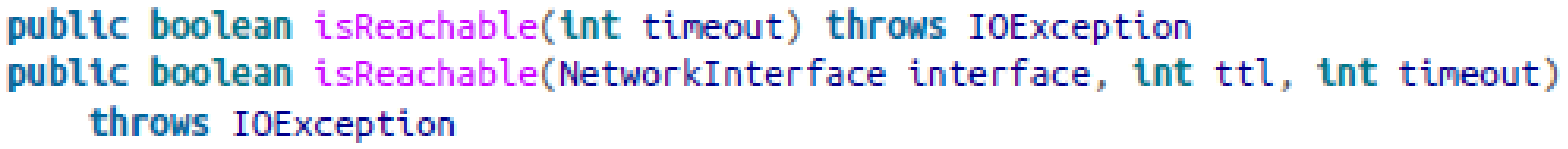
Here's the output from an IPv4 and IPv6 address:

```
$ java  IPCharacteristics 127.0.0.1
/127.0.0.1 is loopback address.
/127.0.0.1 is a global address.
/127.0.0.1 is a unicast address.
$ java  IPCharacteristics 192.168.254.32
/192.168.254.32 is a site-local address.
/192.168.254.32 is a unicast address.
$ java  IPCharacteristics www.oreilly.com
www.oreilly.com/208.201.239.37 is a global address.
www.oreilly.com/208.201.239.37 is a unicast address.
$ java  IPCharacteristics 224.0.2.1
/224.0.2.1 is a global address.
/224.0.2.1 is a global multicast address.
$ java  IPCharacteristics FF01:0:0:0:0:0:0:1
```

```
/ff01:0:0:0:0:0:0:1 is a global address.
/ff01:0:0:0:0:0:0:1 is an interface-local multicast address.
$ java  IPCharacteristics FF05:0:0:0:0:0:0:101
/ff05:0:0:0:0:0:0:101 is a global address.
/ff05:0:0:0:0:0:0:101 is a site wide multicast address.
$ java  IPCharacteristics 0::1
/0:0:0:0:0:0:0:1 is loopback address.
/0:0:0:0:0:0:0:1 is a global address.
/0:0:0:0:0:0:0:1 is a unicast address.
```

Testing Reachability

- The InetAddress class has two isReachable() methods that test whether a particular node is reachable from the current host (i.e., 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.



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

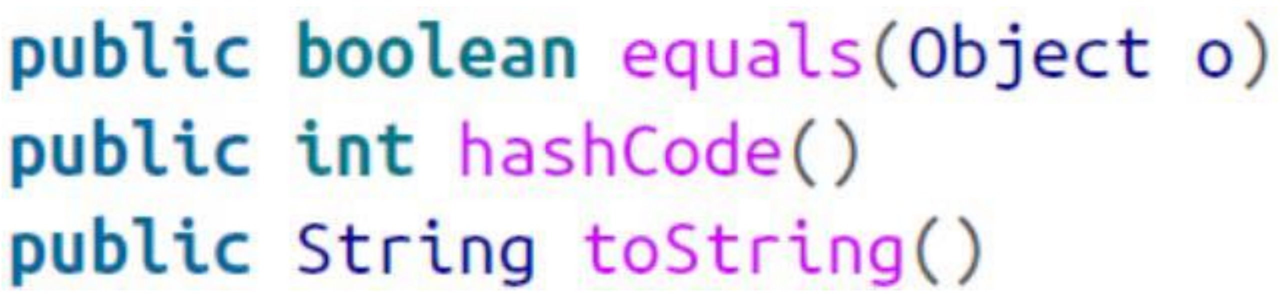
Testing Reachability

- The two testing reachability methods attempt to use traceroute (more specifically, ICMP echo requests) to find out if the specified address is reachable.
- If the host responds within timeout milliseconds, the methods return true; otherwise, they return false. An IOException will be thrown if there’s a network error. The second variant also lets you specify the local network interface the connection is made from and the “time-to-live” (the maximum number of network hops the connection will attempt before being discarded).

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:

Object Methods



```
public boolean equals(Object o)
public int hashCode()
public String toString()
```

- equals(): both of InetAddress with the same IP address (not same hostname)
- hashCode(): solely from the IP address; consistent with the equals()
- toString(): has the form of hostname/dotted quad

Object Methods

- 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.
- Thus, an InetAddress object for *www.ibiblio.org* is equal to an InetAddress object for *www.cafeaulait.org* because both names refer to the same IP address.
- Example in next slide creates InetAddress objects for *www.ibiblio.org* and *helios.ibiblio.org* and then tells you whether they’re the same machine.

- *Are www.ibiblio.org and helios.ibiblio.org the same?*

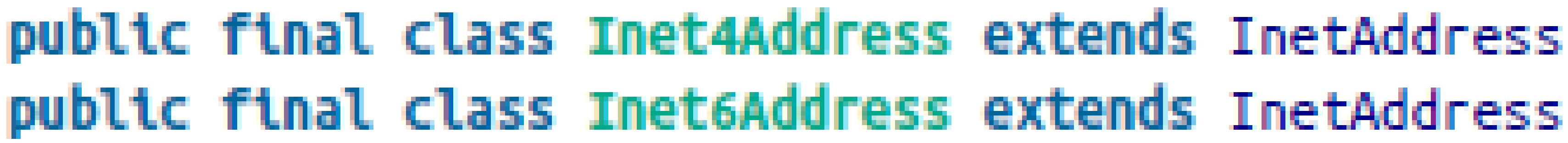
```
import java.net.*;

public class IBiblioAliases {

  public static void main (String args[]) {
    try {
      InetAddress ibiblio = InetAddress.getByName("www.ibiblio.org");
      InetAddress helios = InetAddress.getByName("helios.ibiblio.org");
      if (ibiblio.equals(helios)) {
        System.out.println
            ("www.ibiblio.org is the same as helios.ibiblio.org");
      } else {
        System.out.println
            ("www.ibiblio.org is not the same as helios.ibiblio.org");
      }
    } catch (UnknownHostException ex) {
      System.out.println("Host lookup failed.");
    }
  }
}
```

When you run this program, you discover:

```
% java IBiblioAliases
www.ibiblio.org is the same as helios.ibiblio.org
```

Inet4Address and Inet6Address

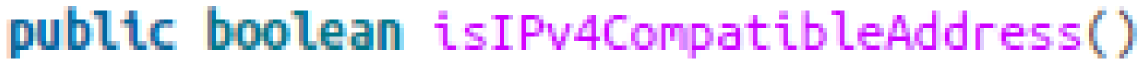
- Java uses two classes, Inet4Address and Inet6Address, in order to distinguish IPv4 addresses from IPv6 addresses:



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

Inet4Address and Inet6Address

- Most of the time, you really shouldn’t be concerned with whether an address is an IPv4 or IPv6 address.
- 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 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():

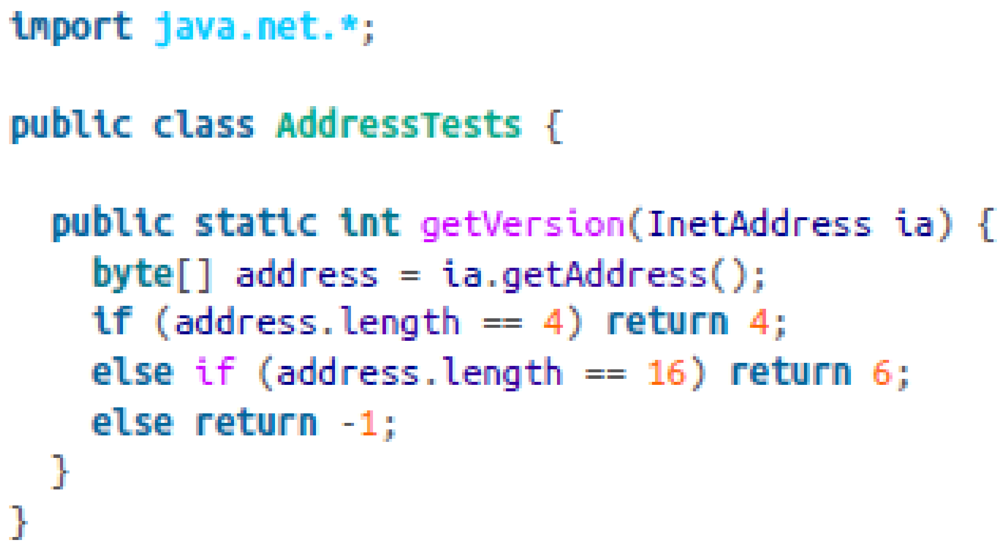


```
public boolean isIPv4CompatibleAddress()
```

Inet4Address and Inet6Address

- This method returns true if and only if the address is essentially an IPv4 address stuffed into an IPv6 container—which means only the last four bytes are nonzero.
- That is, the address has the form *0:0:0:0:0:0:0:xxxx*. If this is the case, you can pull off the last four bytes from the array returned by getBytes() and use this data to create an Inet4Address instead. However, you rarely need to do this.

Determining whether an IP address is v4 or v6



```java
import java.net.*;

public class AddressTests {

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

The NetworkInterface Class

- 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, and to create InetAddress objects from them.
- These InetAddress objects can then be used to create sockets, server sockets, and so forth
  - ➢ NetworkInterface: Factory Methods
  - ➢ NetworkInterface: Getter Methods

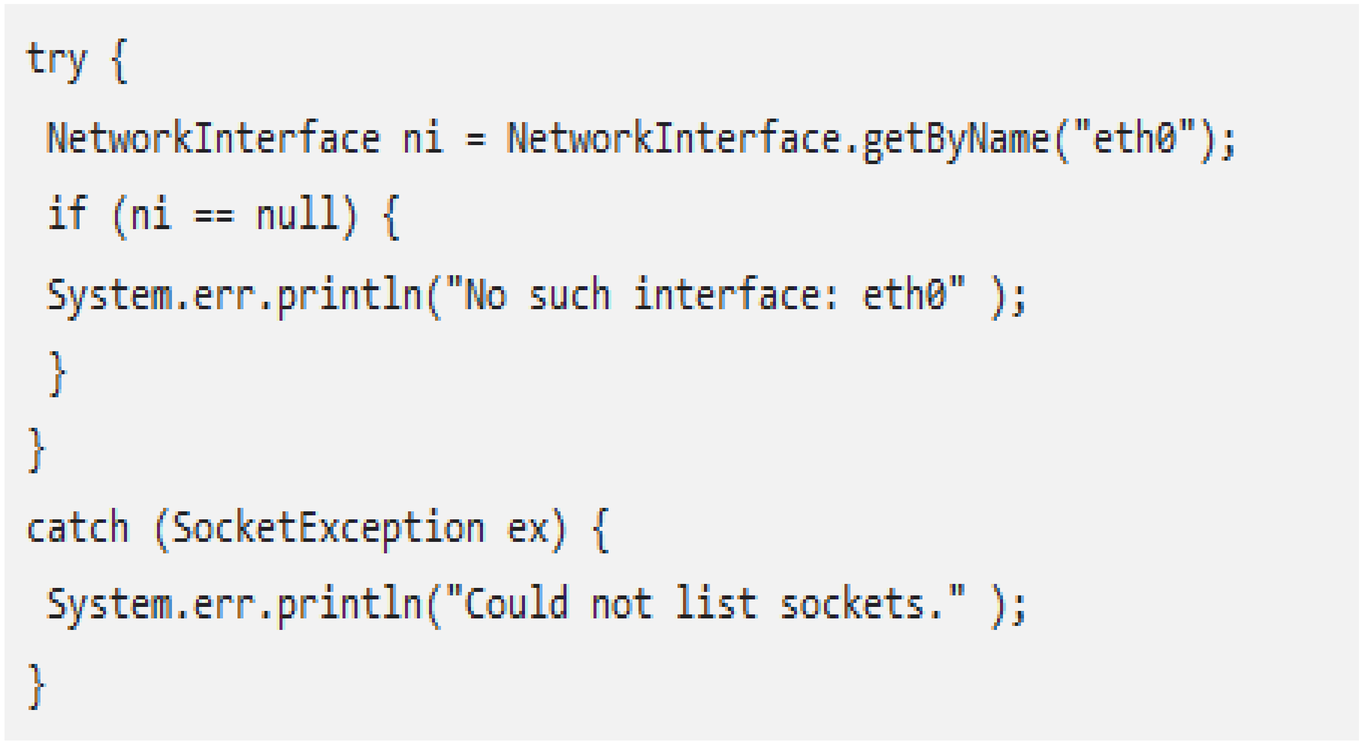
NetworkInterface: Factory Methods

- Because 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.

NetworkInterface: Factory Methods

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.
- If the underlying network stack encounters a problem while locating the relevant network interface, a SocketException is thrown, but this isn’t too likely to happen.

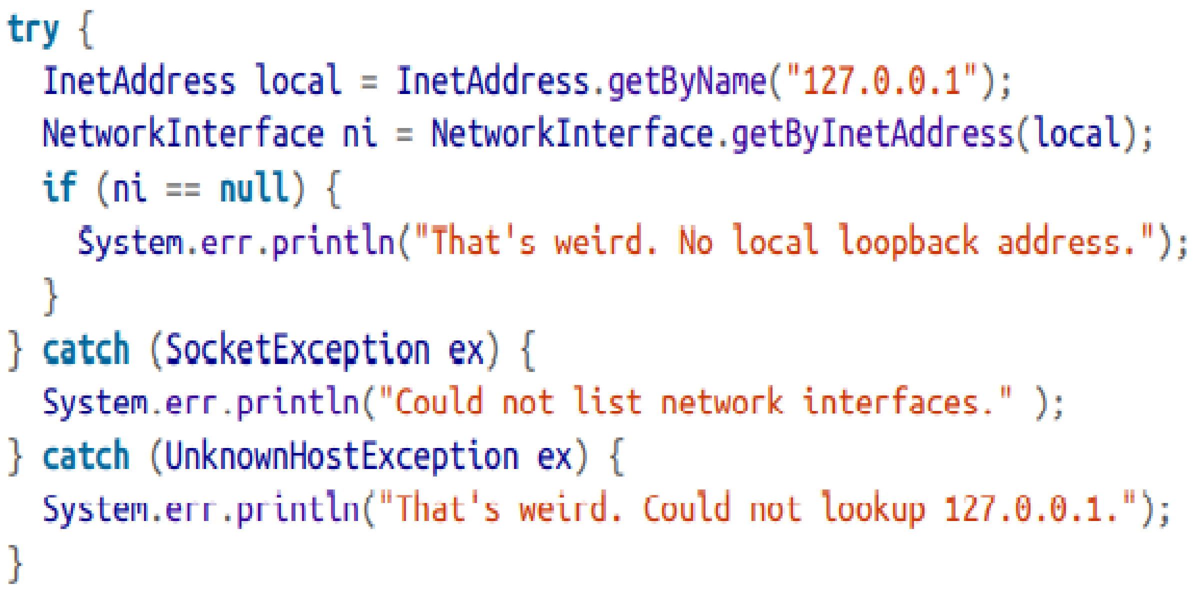


```
try {
 NetworkInterface ni = NetworkInterface.getByName("eth0");
 if (ni == null) {
 System.err.println("No such interface: eth0" );
 }
}
catch (SocketException ex) {
 System.err.println("Could not list sockets." );
}
```

NetworkInterface: Factory Methods

```
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.



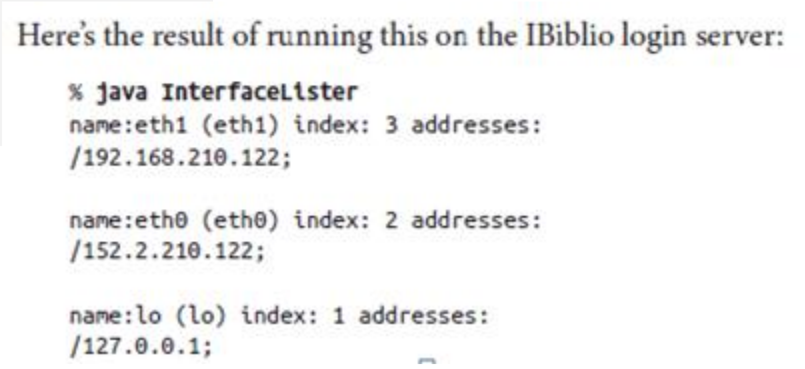
```
try {
  InetAddress local = InetAddress.getByName("127.0.0.1");
  NetworkInterface ni = NetworkInterface.getByInetAddress(local);
  if (ni == null) {
    System.err.println("That's weird. No local loopback address.");
  }
} catch (SocketException ex) {
  System.err.println("Could not list network interfaces." );
} catch (UnknownHostException ex) {
  System.err.println("That's weird. Could not lookup 127.0.0.1.");
}
```

NetworkInterface: Factory Methods

public static Enumeration getNetworkInterfaces() throws SocketException

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

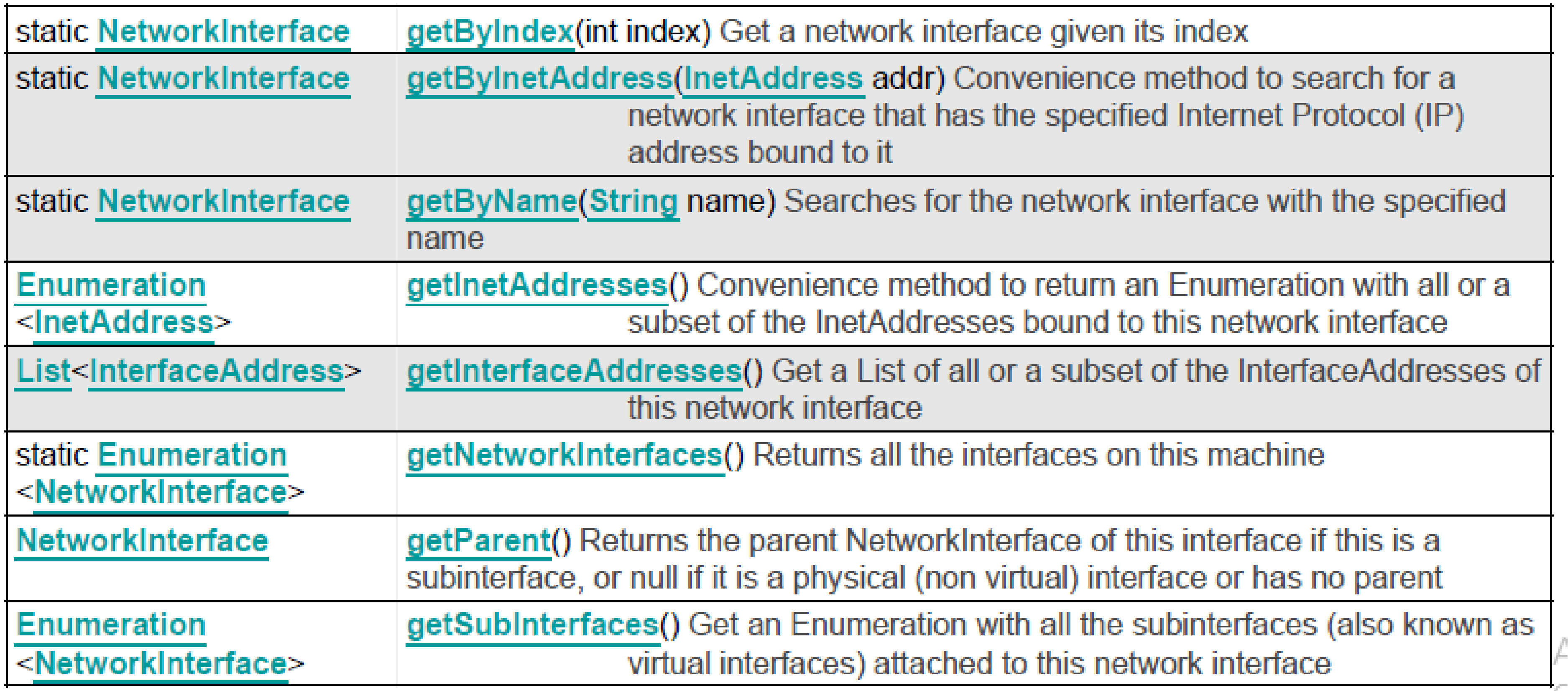
```
import java.net.*;
import java.util.*;
public class InterfaceLister {
 public static void main(String[] args) throws Exception {
 Enumeration interfaces = NetworkInterface.getNetworkInterfaces( );
 while (interfaces.hasMoreElements( )) {
 NetworkInterface ni = (NetworkInterface) interfaces.nextElement( );
 System.out.println(ni); } }
}
```



Here's the result of running this on the IBiblio login server:

```
% java InterfaceLister
name:eth1 (eth1) index: 3 addresses:
/192.168.210.122;

name:eth0 (eth0) index: 2 addresses:
/152.2.210.122;

name:lo (lo) index: 1 addresses:
/127.0.0.1;
```

NetworkInterface: Factory Methods



static NetworkInterface	getByIndex(int index) Get a network interface given its index
static NetworkInterface	getByInetAddress(InetAddress addr) Convenience method to search for a network interface that has the specified Internet Protocol (IP) address bound to it
static NetworkInterface	getByName(String name) Searches for the network interface with the specified name
Enumeration <InetAddress>	getInetAddresses() Convenience method to return an Enumeration with all or a subset of the InetAddresses bound to this network interface
List<InterfaceAddress>	getInterfaceAddresses() Get a List of all or a subset of the InterfaceAddresses of this network interface
static Enumeration <NetworkInterface>	getNetworkInterfaces() Returns all the interfaces on this machine
NetworkInterface	getParent() Returns the parent NetworkInterface of this interface if this is a subinterface, or null if it is a physical (non virtual) interface or has no parent
Enumeration <NetworkInterface>	getSubInterfaces() Get an Enumeration with all the subinterfaces (also known as virtual interfaces) attached to this network interface

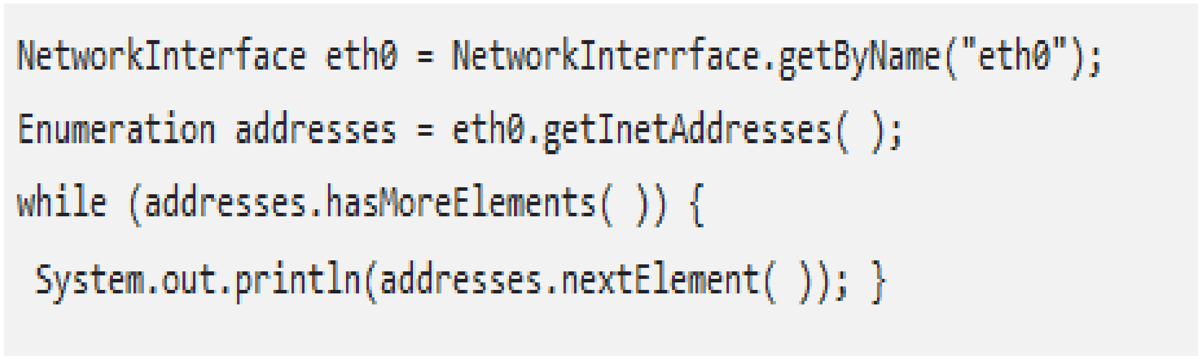
NetworkInterface Getter Methods

- Once you have a NetworkInterface object, you can inquire about its IP address and name.

NetworkInterface: GetterMethods

public Enumeration getInetAddresses()

- A single network interface may be bound to more than one IP address. This situation isn’t common these days, but it does happen.
- The getInetAddresses() method returns a java.util.Enumeration containing an InetAddress object for each IP address the interface is bound to.
- For example, this code fragment lists all the IP addresses for the eth0 interface:



```
NetworkInterface eth0 = NetworkInterrface.getByName("eth0");
Enumeration addresses = eth0.getInetAddresses( );
while (addresses.hasMoreElements( )) {
 System.out.println(addresses.nextElement( )); }
```

NetworkInterface: GetterMethods

```
public String getName()
```

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

NetworkInterface: GetterMethods

```
public String getDisplayName()
```

- The getDisplayName() method allegedly returns a more human-friendly name for the particular NetworkInterface—something like “Ethernet Card 0.”

NetworkInterface Getter Methods

boolean	equals(Object obj) Compares this object against the specified object
String	getDisplayName() Get the display name of this network interface
byte[]	getHardwareAddress() the hardware address (usually MAC) of the interface if it has one and if it can be accessed given the current privileges
int	getIndex() Returns the index of this network interface
Enumeration <InetAddress>	getInetAddresses() Convenience method to return an Enumeration with all or a subset of the InetAddresses bound to this network interface
List <InterfaceAddress>	getInterfaceAddresses() Get a List of all or a subset of the InterfaceAddresses of this network interface
int	getMTU() Returns the Maximum Transmission Unit (MTU) of this interface
String	getName() Get the name of this network interface
NetworkInterface	getParent() Returns the parent NetworkInterface of this interface if this is a subinterface, or null if it is a physical (non virtual) interface or has no parent
Enumeration <NetworkInterface>	getSubInterfaces() an Enumeration with all the subinterfaces (also known as virtual interfaces) attached to this network interface
int	hashCode() Returns a hash code value for the object.
boolean	isLoopback() Returns whether a network interface is a loopback interface.
boolean	isPointToPoint() Returns whether a network interface is a point to point interface.
boolean	isUp() Returns whether a network interface is up and running.
boolean	isVirtual() Returns whether this interface is a virtual interface (also called subinterface).
boolean	supportsMulticast() Returns whether a network interface supports multicasting or not.
String	toString() Returns a string representation of the object.

Some Useful Programs

- SpamCheck: asks sbl.spamhaus.org if an IPv4 is a spammer
  - i.e. A DNS query for 17.34.87.207.sbl.spamhaus.org succeeds (/returns 127.0.0.2) if 17.34.87.207 is a spammer

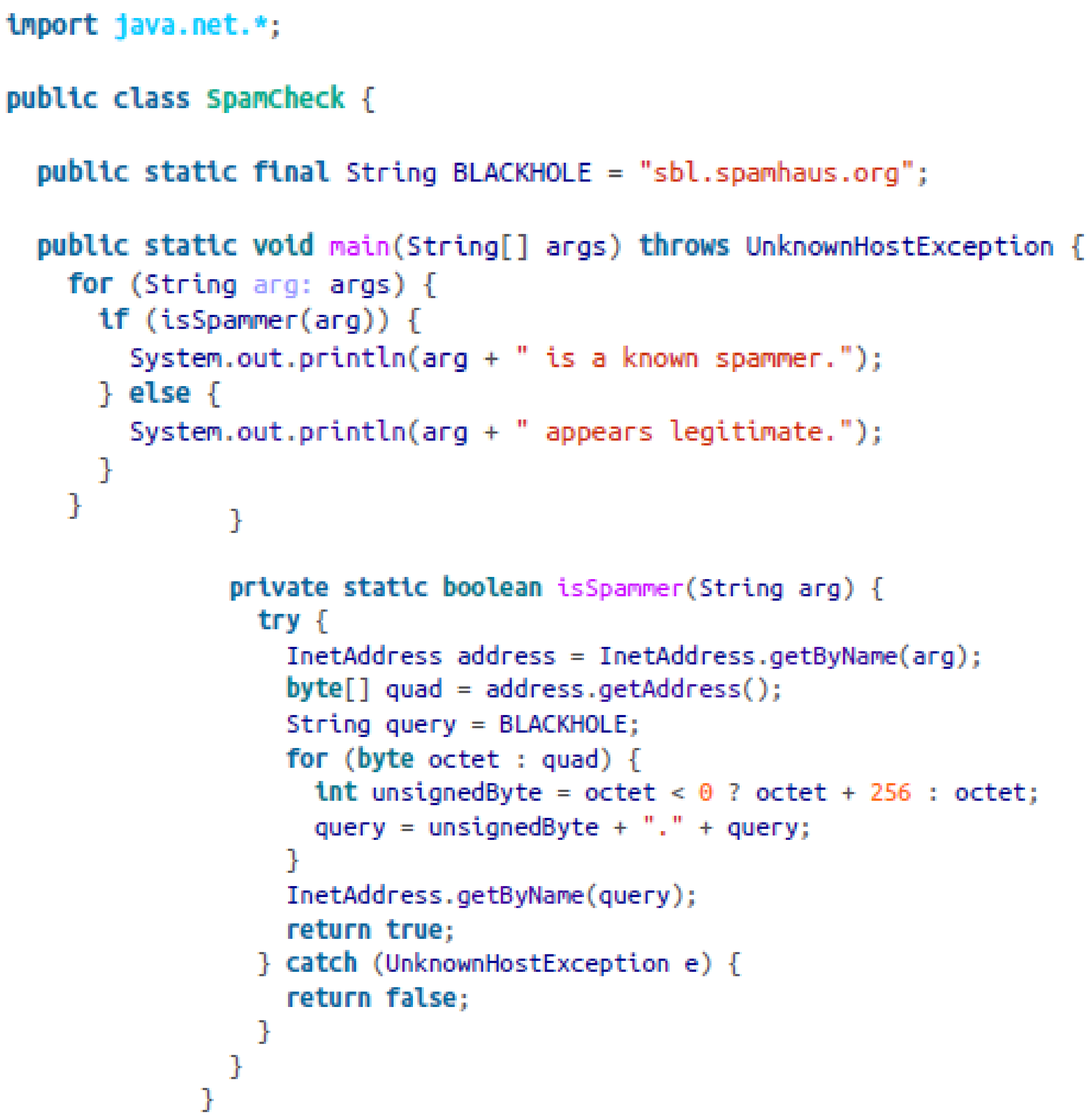
- Processing Web Server Logfiles: reads a web server logfile and prints each line with IP addresses converted to hostnames
  - Usually a Web server simply logs the IP addresses and converts them to hostnames at a later time
  - Common logfile format:

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

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.

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

Here's some sample output:

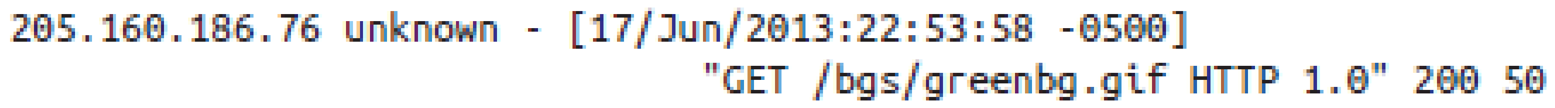
```
$ java SpamCheck 207.34.56.23 125.12.32.4 130.130.130.130
207.34.56.23 appears legitimate.
125.12.32.4 appears legitimate.
130.130.130.130 appears legitimate.
```

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.

Processing Web Server Logfiles

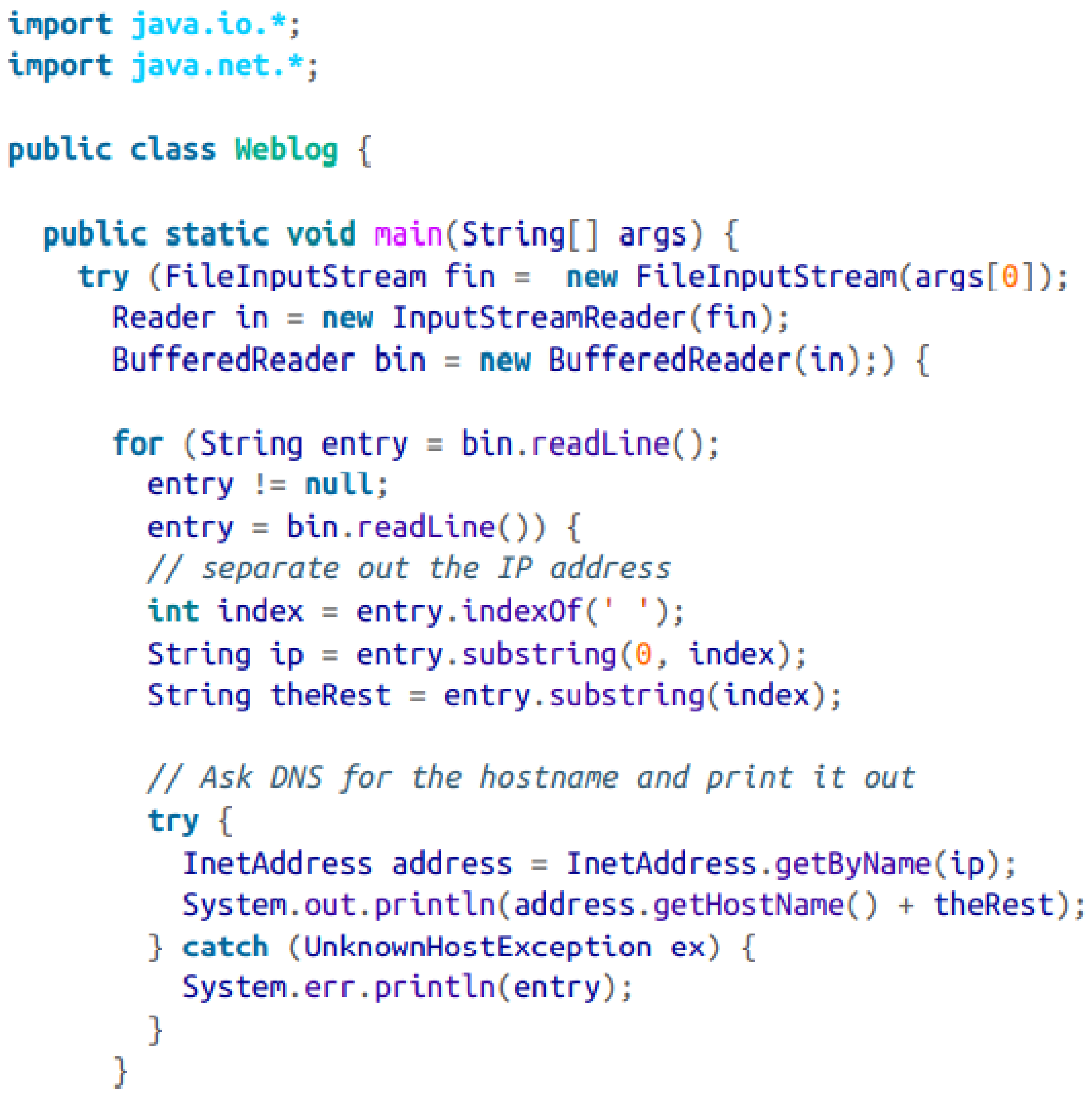
- Log Format:



```
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.

Process Web server log files



```java
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);
        }
      }
    }
```

Thank You