CSE 686 Internet Programming

Week 3: Network Programming in Java

Edmund Yu, PhD
Associate Teaching Professor
esyu@syr.edu

January 29, 2018

Reading Assignments

Required:

- Textbook #1: **TCP/IP Sockets in Java**, Chapter 1: Introduction.
- **Reading Assignment #2:** The Java Tutorials at Oracle.com: http://docs.oracle.com/javase/tutorial/, at a minimum:
  - All sections under the **Getting Started** trail
  - All sections under **Learning the Java Language** trail
  - The **Exceptions** section under the **Essential Classes** trail

Recommended:

- Textbook #3: **Fundamentals of Web Development, 2nd Ed.,** Chapter 1: Introduction to Web Development & Chapter 2: How the Web Works
- **Reading Assignment #1 (Optional)**
  - HTML5 & JavaScript Tutorials at w3schools.com

Package java.net

Provides the classes for implementing networking applications.

See: Description

Interface Summary

Interface	Description
ContentHandlerFactory	This interface defines a factory for content handlers.
CookiePolicy	CookiePolicy implementations decide which cookies should be accepted and which should be rejected.
CookieStore	A CookieStore object represents a storage for cookie.
DatagramSocketImplFactory	This interface defines a factory for datagram socket implementations.
FileNameMap	A simple interface which provides a mechanism to map between a file name and a MIME type string.
ProtocolFamily	Represents a family of communication protocols.
SocketImplFactory	This interface defines a factory for socket implementations.
SocketOption<T>	A socket option associated with a socket.
SocketOptions	Interface of methods to get/set socket options.
URLStreamHandlerFactory	This interface defines a factory for URL stream protocol handlers.

Class Summary

Class	Description
Authenticator	The class Authenticator represents an object that knows how to obtain authentication for a network connection.

InetAddressExample.java

http://cs.ecs.baylor.edu/~donahoo/practical/JavaSockets2/code/ cs.ecs.baylor.edu

File Edit View Favorites Tools Help

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

public class InetAddressExample {

  public static void main(String[] args) {

    // Get the network interfaces and associated addresses for this host
    try {
      Enumeration<NetworkInterface> interfaceList = NetworkInterface.getNetworkInterfaces();
      if (interfaceList == null) {
        System.out.println("--No interfaces found--");
      } else {
        while (interfaceList.hasMoreElements()) {
          NetworkInterface iface = interfaceList.nextElement();
          System.out.println("Interface " + iface.getName() + ":");
          Enumeration<InetAddress> addrList = iface.getInetAddresses();
          if (!addrList.hasMoreElements()) {
            System.out.println("\t(No addresses for this interface)");
          }
          while (addrList.hasMoreElements()) {
            InetAddress address = addrList.nextElement();
            System.out.print("\tAddress "
                + ((address instanceof Inet4Address ? "(v4)"
                    : (address instanceof Inet6Address ? "(v6)" : "(?)"))));
            System.out.println(": " + address.getHostAddress());
          }
        }
      }
    } catch (SocketException se) {
      System.out.println("Error getting network interfaces:" + se.getMessage());
    }

    // Get name(s)/address(es) of hosts given on command line
    for (String host : args) {
      try {
        System.out.println(host + ":");
        InetAddress[] addressList = InetAddress.getAllByName(host);
        for (InetAddress address : addressList) {
          System.out.println("\t" + address.getHostName() + "/" + address.getHostAddress());
        }
      } catch (UnknownHostException e) {
        System.out.println("\tUnable to find address for " + host);
      }
    }
  }
}
```

The InetAddress Class

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, etc.

- Usually, it includes both a hostname and an IP address.

compact1, compact2, compact3

java.net

Class InetAddress

java.lang.Object
    java.net.InetAddress

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

Inet4Address, Inet6Address

```
public class InetAddress
extends Object
implements Serializable
```

This class represents an Internet Protocol (IP) address.

An IP address is either a 32-bit or 128-bit unsigned number used by IP, a lower-level protocol on which protocols like UDP and TCP are built. The IP address architecture is defined by *RFC 790: Assigned Numbers*, *RFC 1918: Address Allocation for Private Internets*, *RFC 2365: Administratively Scoped IP Multicast*, and *RFC 2373: IP Version 6 Addressing Architecture*. An instance of an InetAddress consists of an IP

Creating New InetAddress Objects

- There are no public constructors in the InetAddress class.
- It has static (factory) methods that connect to a DNS server to resolve a hostname.
- The most common one is InetAddress.**getByName**():

```
InetAddress address = InetAddress.getByName("www.syr.edu");
```

- This method does not only set a private String field in the InetAddress class, it also makes a connection to the local DNS server to look up the name and its numeric address.
- If the DNS server can't find the address, this method throws an **UnknownHostException**, a subclass of IOException.

SUByName.java

```
import java.net.*;

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

Creating New InetAddress Objects

- You can also pass the **dotted quad** address to InetAddress.getByName():

```
InetAddress address = InetAddress.getByName("128.230.18.198");
```

- See **SUByName2.java** on the next slide

SUByName2.java

```
import java.net.*;

public class SUByName2 {
  public static void main (String[] args) {
    try {
          InetAddress address = InetAddress.getByName("128.230.18.198");
          System.out.println(address);
    } catch (UnknownHostException ex) {
          System.out.println("Could not find www.syr.edu");
    }
  }
```

getByName

```
public static InetAddress getByName(String host)
                            throws UnknownHostException
```

Determines the IP address of a host, given the host's name.

The host name can either be a machine name, such as "`java.sun.com`", or a textual representation of its IP address. If a literal IP address is supplied, only the validity of the address format is checked.

For `host` specified in literal IPv6 address, either the form defined in RFC 2732 or the literal IPv6 address format defined in RFC 2373 is accepted. IPv6 scoped addresses are also supported. See here for a description of IPv6 scoped addresses.

If the host is `null` then an `InetAddress` representing an address of the loopback interface is returned. See RFC 3330 section 2 and RFC 2373 section 2.5.3.

Parameters:

`host` - the specified host, or null.

Returns:

an IP address for the given host name.

Throws:

`UnknownHostException` - if no IP address for the host could be found, or if a scope_id was specified for a global IPv6 address.

`SecurityException` - if a security manager exists and its checkConnect method doesn't allow the operation

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
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.

Methods inherited from class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait, wait

SUByName3.java

```
import java.net.*;

public class SUByName3 {
  public static void main (String[] args) {  // getAllByName
    try {
      InetAddress[] addresses = InetAddress.getAllByName("www.syr.edu");
          for (InetAddress address : addresses) {
              System.out.println(address);
          }
    } catch (UnknownHostException ex) {
          System.out.println("Could not find www.syr.edu");
    }
}
```

Creating New InetAddress Objects

- The **getLocalHost()** method also 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 <u>real hostname</u> and IP address, but if that fails it may return the **loopback** address instead.
- The loopback address is the dotted quad address "127.0.0.1".
  - This is the hostname "**localhost**"

MyAddress.java

```
import java.net.*;

public class MyAddress {
  public static void main (String[] args) {
    try {
          InetAddress me = InetAddress.getLocalHost();
          System.out.println(me);
    } catch (UnknownHostException ex) {
          System.out.println("Could not find this computer's address.");
    }
  }
}
```

❖ **This program is from Java Network Programming (Textbook #2)**

Creating New InetAddress Objects

- If you know a numeric address, you can create an InetAddress object from that address <u>without talking to DNS,</u> using:

  InetAddress.getByAddress()

- This method may create addresses for hosts that do not exist or cannot be resolved:

public static InetAddress getByAddress(**byte**[] addr)
  throws UnknownHostException

- creates an InetAddress object with an IP address and no hostname.

public static InetAddress getByAddress(**String** hostname, **byte** [] addr)
  throws UnknownHostException

- creates an InetAddress object with an IP address and a hostname.

SUByAddress.java

```
import java.net.*;

public class SUByAddress {
  public static void main (String[] args) {
    byte[] address = {(byte)128,(byte)230, (byte)18, (byte)198};

    try {
      InetAddress syr = InetAddress.getByAddress(address);
      InetAddress syrWithName = InetAddress.getByAddress("www.syr.edu", address);
      System.out.println(syr); // uses toString()
      System.out.println(syrWithName); // uses toString()
    } catch (UnknownHostException ex) {
      System.out.println("Could not find that address");
    }
  }
}
```

Getters

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

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

- There are no **setters** in the InetAddress class. (Why?)

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
boolean	**equals**(Object obj) Compares this object against the specified object.
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.
int	**hashCode**() Returns a hashcode for this IP address.
boolean	**isAnyLocalAddress**() Utility routine to check if the InetAddress in a wildcard address.
boolean	**isLinkLocalAddress**() Utility routine to check if the InetAddress is an link local address.

Getters

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

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

getHostName()

- The getHostName() method returns a String that contains the name of the host, for the IP address represented by this InetAddress object.

- If the machine doesn't have a hostname, a dotted quad format of the numeric IP address is returned.

- Usage:

```
InetAddress address = InetAddress.getByName("128.230.18.198");
System.out.println(address.getHostName()); // a Getter
```

- If the address you look up does not have a hostname, **getHostName**() simply returns the dotted quad address you supplied. (Exercise: Add all 4 getters to SUByName.java)

Getters

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

public String **getHostName**()

public String **getCanonicalHostName**()

public String **getHostAddress**()

public byte[] **getAddress**()

getCanonicalHostName()

- The getCanonicalHostName() method is 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.
- Usage:

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

- Revise **MyAddress.java**

getCanonicalHostName()

- The getCanonicalHostName() method is particularly useful when you start with a dotted quad IP address rather than the hostname.

```
import java.net.*;

public class ReverseTest { // from Textbook #2
  public static void main (String[] args) throws
          UnknownHostException {
      InetAddress ia = InetAddress.getByName("208.201.239.100");
      System.out.println(ia.getCanonicalHostName());
  }
}
```

Getters

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

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

getHostAddress()

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

```
import java.net.*;

public class MyAddress { // revised again
   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.");
      }
   }
}
```

Getters

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

public String **getHostName**()

public String **getCanonicalHostName**()

public String **getHostAddress**()

public byte[] **getAddress**()

getAddress()

- getAddress() method returns an IP address as <u>an array of bytes</u> in network byte order.
  - The most significant byte (i.e., the first byte in the address's dotted quad form) is the first byte in the array, or element zero.
  - To be ready for IPv6 addresses, try not to assume anything about the length of this array.
  - If you need to know the length of the array, use the array's <u>length field</u>.

- Usage

```
InetAddress me = InetAddress.getLocalHost();
byte[] address = me.getAddress();
```

getAddress()

- The bytes returned by **getAddress()** are unsigned, which poses a problem.
- Unlike C, Java doesn't have an <u>unsigned byte</u> primitive data type.
- Bytes with values higher than 127 are treated as negative numbers.
- Therefore, if you want to do anything with the bytes returned by getAddress(), you need to <u>promote</u> the bytes to int's and make appropriate adjustments:

```
int unsignedByte = signedByte < 0 ? signedByte + 256 : signedByte;
```

getAddress()

- One reason to look at the raw bytes of an IP address is to determine the type of the address.
- Test the number of bytes in the array returned by getAddress() to determine whether you're dealing with an IPv4 or IPv6 address:

```
import java.net.*;
public class AddressTests { // from Textbook #2
    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;
    }
    // Exercise: add a main() here
}
```

Address Types

- Some IP addresses and some patterns of addresses have special meanings.
- Java includes 10 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 isMCNodeLocal()
public boolean isMCLinkLocal()
public boolean isMCSiteLocal()
public boolean isMCOrgLocal()
```

boolean	**isAnyLocalAddress()** Utility routine to check if the InetAddress in a wildcard address.
boolean	**isLinkLocalAddress()** Utility routine to check if the InetAddress is an link local address.
boolean	**isLoopbackAddress()** Utility routine to check if the InetAddress is a loopback address.
boolean	**isMCGlobal()** Utility routine to check if the multicast address has global scope.
boolean	**isMCLinkLocal()** Utility routine to check if the multicast address has link scope.
boolean	**isMCNodeLocal()** Utility routine to check if the multicast address has node scope.
boolean	**isMCOrgLocal()** Utility routine to check if the multicast address has organization scope.
boolean	**isMCSiteLocal()** Utility routine to check if the multicast address has site scope.
boolean	**isMulticastAddress()** Utility routine to check if the InetAddress is an IP multicast address.
boolean	**isReachable(int timeout)** Test whether that address is reachable.
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 site local address.

isAnyLocalAddress()

- The isAnyLocalAddress() method returns true if the address is a **wildcard address**, false otherwise.
  - A wildcard address matches any address of the local system.
  - This is important if the system has multiple network interfaces, as might be the case on a system with multiple Ethernet cards or an Ethernet card and an 802.11 WiFi interface.
  - In IPv4, the wildcard address is 0.0.0.0.
  - In IPv6, this address is 0:0:0:0:0:0:0:0 (a.k.a. ::).

isLoopbackAddress()

- The isLoopbackAddress() method returns true if the address is the loopback address, false otherwise.
  - The loopback address connects to the same computer directly in the IP layer without using any physical hardware.
  - Thus, connecting to the loopback address enables tests to bypass potentially buggy or nonexistent Ethernet, PPP, and other drivers, helping to isolate problems.
  - Connecting to the loopback address is not the same as connecting to the system's normal IP address from the same system.
    - In IPv4, this address is 127.0.0.1
    - In IPv6, this address is 0:0:0:0:0:0:0:1 (a.k.a. ::1).

isLinkLocalAddress()

- The isLinkLocalAddress() method returns true if the address is an **IPv6** link-local address, false otherwise.
  - This is an address used to help IPv6 networks self-configure, much like **DHCP** on IPv4 networks but without necessarily using a server.
  - Routers do not forward packets addressed to a link-local address beyond the local subnet.
  - 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 assigned by the Ethernet card manufacturer.

isSiteLocalAddress()

- 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 <u>within a site or campus</u> 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.

isMulticastAddress()

- The isMulticastAddress() method returns true if the address is a multicast address, false otherwise.
  - Multicasting broadcasts content to all subscribed computers rather than to one particular computer.
  - 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 FF.

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, 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
```

- They use **traceroute** to find out if the specified address is reachable, which in turn uses **Echo**.
- If the host responds within timeout milliseconds,the methods return true; otherwise, they return false.

isReachable

```
public boolean isReachable(int timeout)
                    throws IOException
```

Test whether that address is reachable. Best effort is made by the implementation to try to reach the host, but firewalls and server configuration may block requests resulting in a unreachable status while some specific ports may be accessible. A typical implementation will use ICMP ECHO REQUESTs if the privilege can be obtained, otherwise it will try to establish a TCP connection on port 7 (Echo) of the destination host.

The timeout value, in milliseconds, indicates the maximum amount of time the try should take. If the operation times out before getting an answer, the host is deemed unreachable. A negative value will result in an IllegalArgumentException being thrown.

Parameters:

`timeout` - the time, in milliseconds, before the call aborts

Returns:

a `boolean` indicating if the address is reachable.

Throws:

`IOException` - if a network error occurs

`IllegalArgumentException` - if `timeout` is negative.

Since:

1.5

isReachable

```
public boolean isReachable(NetworkInterface netif,
                    int ttl
```

isReachable

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

Test whether that address is reachable. Best effort is made by the implementation to try to reach the host, but firewalls and server configuration may block requests resulting in a unreachable status while some specific ports may be accessible. A typical implementation will use ICMP ECHO REQUESTs if the privilege can be obtained, otherwise it will try to establish a TCP connection on port 7 (Echo) of the destination host.

The `network interface` and `ttl` parameters let the caller specify which network interface the test will go through and the maximum number of hops the packets should go through. A negative value for the `ttl` will result in an IllegalArgumentException being thrown.

The timeout value, in milliseconds, indicates the maximum amount of time the try should take. If the operation times out before getting an answer, the host is deemed unreachable. A negative value will result in an IllegalArgumentException being thrown.

Parameters:

`netif` - the NetworkInterface through which the test will be done, or null for any interface

`ttl` - the maximum numbers of hops to try or 0 for the default

`timeout` - the time, in milliseconds, before the call aborts

Returns:

a `boolean`indicating if the address is reachable.

Throws:

`IllegalArgumentException` - if either `timeout` or `ttl` are negative.

`IOException` - if a network error occurs

Since:

Object Methods

- Like every other class, java.net.InetAddress inherits from java.lang.Object, and hence 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**()

Equals() in InetAddress

- An object is equal to an InetAddress object only if it is itself an instance of the InetAddress class and it has the <u>same IP address</u>.
  - It does not need to have the same hostname.
  - Thus, an InetAddress object for *www.syr.edu* is equal to an InetAddress object for *syr-prod-web.syracuse.edu* because both names refer to the same IP address.

Equals() in InetAddress

```
import java.net.*;
public class SUAliases {
   public static void main (String args[]) {
      try {
         InetAddress syr = InetAddress.getByName("www.syr.edu");
         InetAddress its = InetAddress.getByName(“syr-prod-web.syracuse.edu");
         if (syr.equals(its)) {
            System.out.println("www.syr.edu is the same as syr-prod-web.syracuse.edu ");
         } else {
            System.out.println("www.syr.edu is not the same as syr-prod-web.syracuse.edu");
         }
      } catch (UnknownHostException ex) {
            System.out.println("Host lookup failed.");
      }
   }
}
```

InetAddressExample.java

http://cs.ecs.baylor.edu/~donahoo/practical/JavaSockets2/code/ cs.ecs.baylor.edu

File Edit View Favorites Tools Help

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

public class InetAddressExample {

  public static void main(String[] args) {

    // Get the network interfaces and associated addresses for this host
    try {
      Enumeration<NetworkInterface> interfaceList = NetworkInterface.getNetworkInterfaces();
      if (interfaceList == null) {
        System.out.println("--No interfaces found--");
      } else {
        while (interfaceList.hasMoreElements()) {
          NetworkInterface iface = interfaceList.nextElement();
          System.out.println("Interface " + iface.getName() + ":");
          Enumeration<InetAddress> addrList = iface.getInetAddresses();
          if (!addrList.hasMoreElements()) {
            System.out.println("\t(No addresses for this interface)");
          }
          while (addrList.hasMoreElements()) {
            InetAddress address = addrList.nextElement();
            System.out.print("\tAddress "
                + ((address instanceof Inet4Address ? "(v4)"
                    : (address instanceof Inet6Address ? "(v6)" : "(?)"))));
            System.out.println(": " + address.getHostAddress());
          }
        }
      }
    } catch (SocketException se) {
      System.out.println("Error getting network interfaces:" + se.getMessage());
    }

    // Get name(s)/address(es) of hosts given on command line
    for (String host : args) {
      try {
        System.out.println(host + ":");
        InetAddress[] addressList = InetAddress.getAllByName(host);
        for (InetAddress address : addressList) {
          System.out.println("\t" + address.getHostName() + "/" + address.getHostAddress());
        }
      } catch (UnknownHostException e) {
        System.out.println("\tUnable to find address for " + host);
      }
    }
  }
}
```

Textbook #1, pp 10-11
Get it from the book site

Check out this part first

InetAddressExample.java

http://cs.ecs.baylor.edu/~donahoo/practical/JavaSockets2/code/ cs.ecs.baylor.edu

File Edit View Favorites Tools Help

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

public class InetAddressExample {

  public static void main(String[] args) {

    // Get the network interfaces and associated addresses for this host
    try {
      Enumeration<NetworkInterface> interfaceList = NetworkInterface.getNetworkInterfaces();
      if (interfaceList == null) {
        System.out.println("--No interfaces found--");
      } else {
        while (interfaceList.hasMoreElements()) {
          NetworkInterface iface = interfaceList.nextElement();
          System.out.println("Interface " + iface.getName() + ":");
          Enumeration<InetAddress> addrList = iface.getInetAddresses();
          if (!addrList.hasMoreElements()) {
            System.out.println("\t(No addresses for this interface)");
          }
          while (addrList.hasMoreElements()) {
            InetAddress address = addrList.nextElement();
            System.out.print("\tAddress "
                + ((address instanceof Inet4Address ? "(v4)"
                      : (address instanceof Inet6Address ? "(v6)" : "(?)"))));
            System.out.println(": " + address.getHostAddress());
          }
        }
      }
    } catch (SocketException se) {
      System.out.println("Error getting network interfaces:" + se.getMessage());
    }

    // Get name(s)/address(es) of hosts given on command line
    for (String host : args) {
      try {
        System.out.println(host + ":");
        InetAddress[] addressList = InetAddress.getAllByName(host);
        for (InetAddress address : addressList) {
          System.out.println("\t" + address.getHostName() + "/" + address.getHostAddress());
        }
      } catch (UnknownHostException e) {
        System.out.println("\tUnable to find address for " + host);
      }
    }
  }
}
```

To understand this part, we need to learn Enumeration & NetworkInterface

compact1, compact2, compact3

java.util

Interface Enumeration<E>

All Known Subinterfaces:

NamingEnumeration<T>

All Known Implementing Classes:

StringTokenizer

---

public interface **Enumeration<E>**

An object that implements the Enumeration interface generates a series of elements, one at a time. Successive calls to the `nextElement` method return successive elements of the series.

For example, to print all elements of a `Vector<E>` *v*:

```
for (Enumeration<E> e = v.elements(); e.hasMoreElements();)
    System.out.println(e.nextElement());
```

Methods are provided to enumerate through the elements of a vector, the keys of a hashtable, and the values in a hashtable. Enumerations are also used to specify the input streams to a `SequenceInputStream`.

compact1, compact2, compact3

java.util

Interface Iterator<E>

Type Parameters:

E - the type of elements returned by this iterator

All Known Subinterfaces:

ListIterator<E>, PrimitiveIterator<T,T_CONS>, PrimitiveIterator.OfDouble, PrimitiveIterator.OfInt, PrimitiveIterator.OfLong, XMLEventReader

All Known Implementing Classes:

BeanContextSupport.BCSIterator, EventReaderDelegate, Scanner

---

public interface **Iterator<E>**

An iterator over a collection. Iterator takes the place of Enumeration in the Java Collections Framework. Iterators differ from enumerations in two ways:

- Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
- Method names have been improved.

Iterators

- An **iterator** is an object that allows you to process a collection of items one at a time. (See next slide)
  - It lets you step through each item in turn and process it as needed
  - An iterator has a **hasNext** method that returns true if there is at least one more item to process
  - The **next** method returns the next item
  - **Iterator** objects are defined using the Iterator **interface**.

ORACLE Java Documentation

Hide TOC

The Java™ Tutorials

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases.

The Collection Interface

A `Collection` represents a group of objects known as its elements. The `Collection` interface is used to pass around collections of objects where maximum generality is desired. For example, by convention all general-purpose collection implementations have a constructor that takes a `Collection` argument. This constructor, known as a *conversion constructor*, initializes the new collection to contain all of the elements in the specified collection, whatever the given collection's subinterface or implementation type. In other words, it allows you to *convert* the collection's type.

Suppose, for example, that you have a `Collection<String> c`, which may be a `List`, a `Set`, or another kind of `Collection`. This idiom creates a new `ArrayList` (an implementation of the `List` interface), initially containing all the elements in `c`.

```
List<String> list = new ArrayList<String>(c);
```

Or — if you are using JDK 7 or later — you can use the diamond operator:

```
List<String> list = new ArrayList<>(c);
```

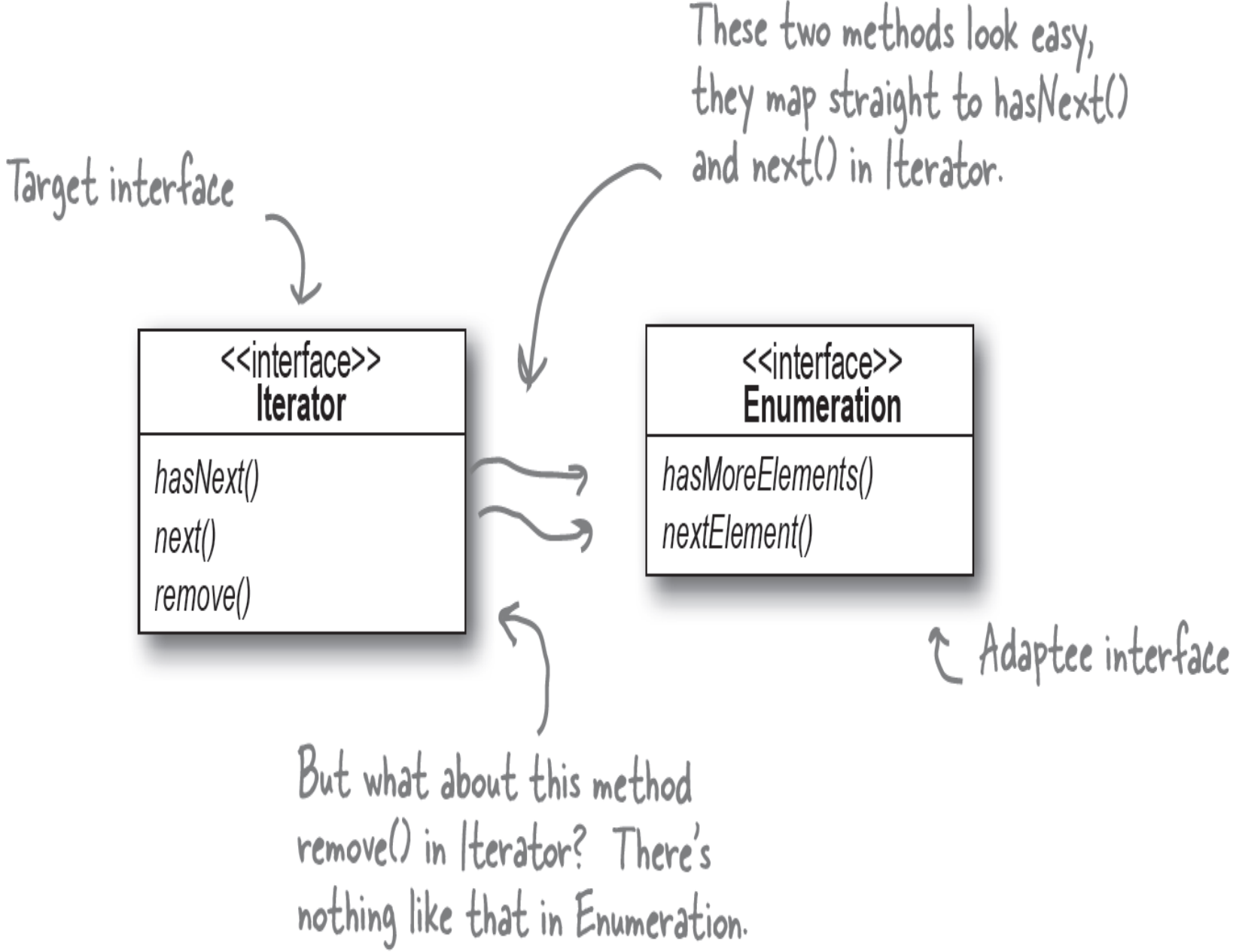
Iterators

- Several classes in the Java standard class library are iterators
- The **Scanner** class is an iterator
  - the **hasNext** method returns true if there is more data to be scanned
  - the **next** method returns the next scanned token as a string
- The Scanner class also has variations on the hasNext method for specific data types (such as **hasNextInt**)

FileReadDemo.java - Notepad

File Edit Format View Help

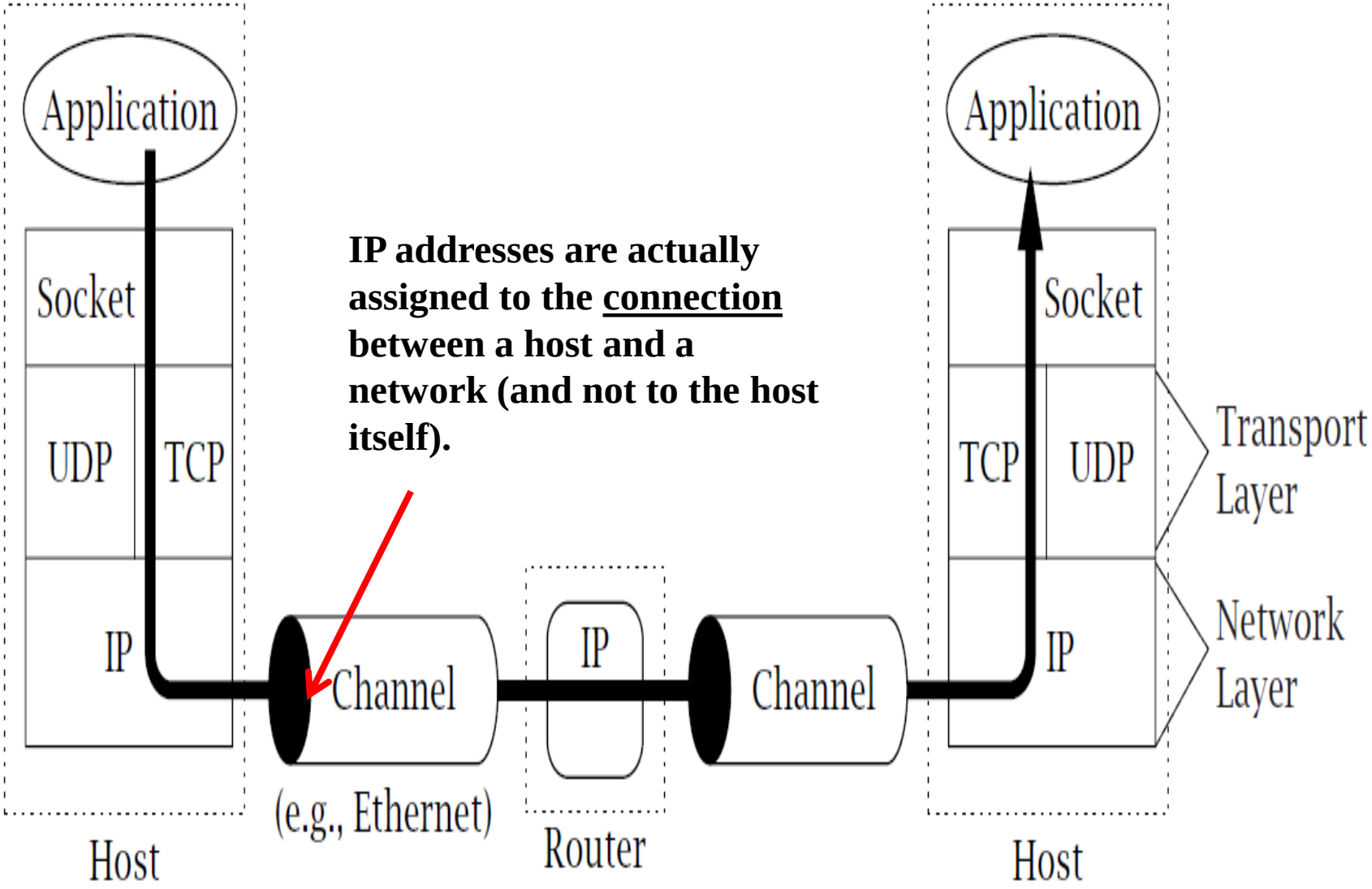
```
public class FileReadDemo
{
   public static void main(String[] args) throws IOException
   {
      // Create a Scanner object for keyboard input.
      Scanner keyboard = new Scanner(System.in);

      // Get the filename.
      System.out.print("Enter the filename: ");
      String filename = keyboard.nextLine();

      // Open the file.
      File file = new File(filename);
      Scanner inputFile = new Scanner(file);

      // Read lines from the file until no more are left.
      while (inputFile.hasNext())
      {
         // Read the next name.
         String friendName = inputFile.nextLine();

         // Display the last name read.
         System.out.println(friendName);
      }

      // Close the file.
      inputFile.close();
   }
}
```


These two methods look easy, they map straight to hasNext() and next() in Iterator.
Target interface
<<interface>>
Iterator
hasNext()
next()
remove()
<<interface>>
Enumeration
hasMoreElements()
nextElement()
Adaptee interface
But what about this method remove() in Iterator? There's nothing like that in Enumeration.

The NetworkInterface Class

The NetworkInterface Class

- The **NetworkInterface** class represents a local IP address assigned to a connection.
  - IP addresses are actually assigned to the connection between a host and a network, and not to the host itself. (See next slide.)
  - That connection is called an **interface**.

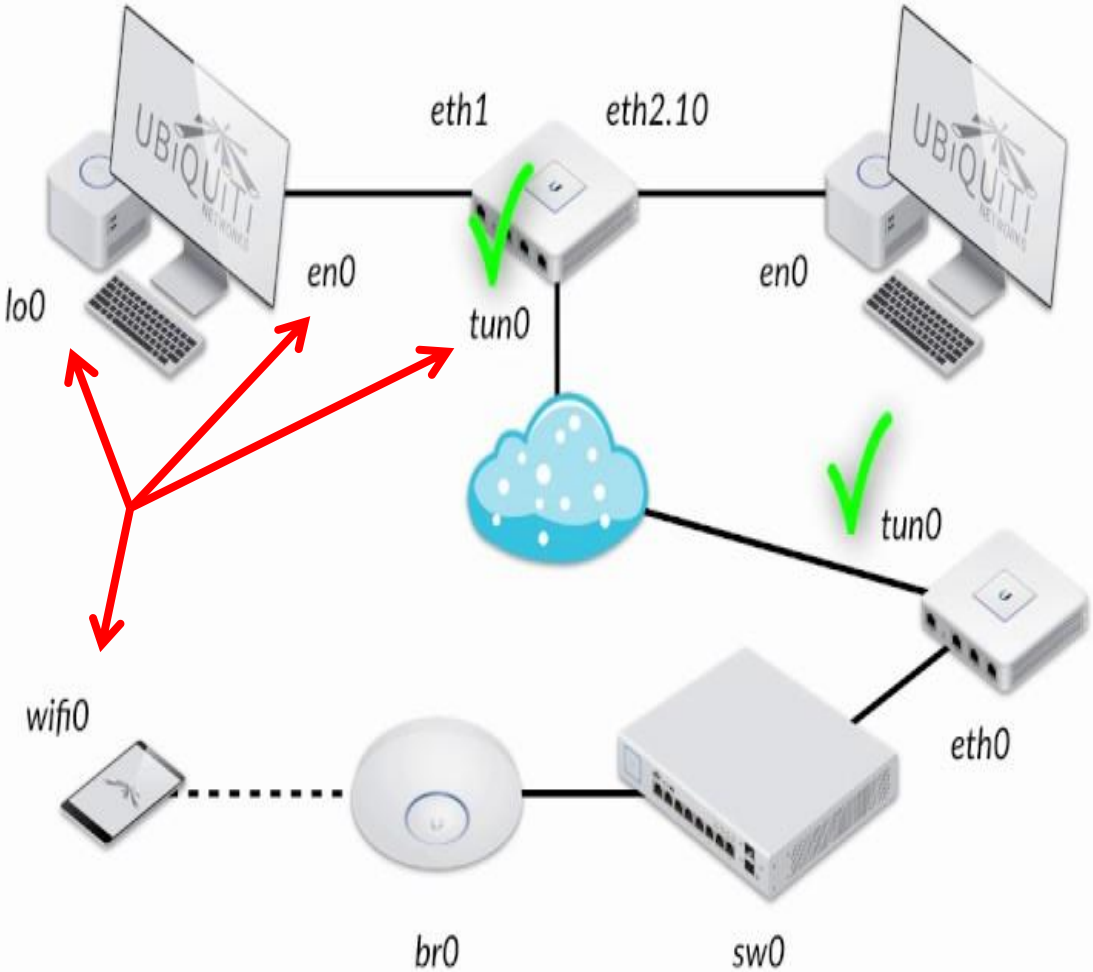


IP addresses are actually assigned to the <u>connection</u> between a host and a network (and not to the host itself).

Figure 1.1: A TCP/IP network.

Network Interfaces

- *Interface* is port on which host sends/receives
- Physical interfaces
  - Wired Ethernet (ex. eth0, en0)
  - Wireless (ex. wifi0, ath0)
  - Switch Ports (ex. sw0)
- Logical interfaces
  - Loopback (ex. lo0)
  - Bridge (ex. br0)
  - Virtual (ex. eth0.10)
  - Tunnel (ex. tun0, p2p0)



The NetworkInterface Class

- This Java class provides access to information about all of a host's interfaces.
  - It provides methods to <u>enumerate</u> all the local addresses, and to create **InetAddress** objects from them, to be used to create sockets, server sockets, etc.

getByName()

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 <u>underlying network</u> encounters a problem while locating the relevant network interface, a **SocketException** is thrown, but this isn't too likely to happen.
- The format of the names is platform dependent.
  - Unix: eth0, eth1,… for Ethernet connections; lo for the local loopback address.

getByName() Example: NITester.java

```
import java.net.*;

public class NITester {
        public static void main(String[] args) {
                try {
                  NetworkInterface ni = NetworkInterface.getByName("eth0");
                  if (ni == null) {
                          System.err.println("No such interface.");
                }
                  else System.out.println(ni); // uses toString()
                } catch (SocketException ex) {
                          System.err.println("Could not list sockets.");
                }
        }
}
```

getByInetAddress()

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.

getByInetAddress() Example: NITester2.java

```
import java.net.*;

public class NITester2 {
  public static void main(String[] args) {
    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.");
    }
  }
}
```

getNetworkInterfaces()

public static Enumeration getNetworkInterfaces()

throws SocketException

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

getNetworkInterfaces() Example: InterfaceLister

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

public class InterfaceLister { // From Textbook #2
        public static void main(String[] args) throws SocketException {
                Enumeration<NetworkInterface> interfaces =
                        NetworkInterface.getNetworkInterfaces();
                while (interfaces.hasMoreElements()) {
                        NetworkInterface ni = interfaces.nextElement();
                        System.out.println(ni);
                }
        }
}
```

isUp

```
public boolean isUp()
            throws SocketException
```

Returns whether a network interface is up and running.

Returns:

true if the interface is up and running.

Throws:

SocketException - if an I/O error occurs.

Since:

1.6

isLoopback

```
public boolean isLoopback()
                  throws SocketException
```

Returns whether a network interface is a loopback interface.

Returns:

true if the interface is a loopback interface.

Throws:

SocketException - if an I/O error occurs.

Since:

1.6

isPointToPoint

Class NetworkInterface

java.lang.Object
    java.net.NetworkInterface

```
public final class NetworkInterface
extends Object
```

This 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 on which a multicast
known by names such as "le0".

Since:

1.4

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
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.
static Enumeration<NetworkInterface>	getNetworkInterfaces() Returns all the interfaces on this machine.

Methods inherited from class java.lang.Object

getByIndex() Example: NITester3.java

```
import java.net.*;

public class NITester3 {
        public static void main(String[] args) {
                try { // getByindex()
                        for (int i = 1; i <= 50; i++) {
                          NetworkInterface ni = NetworkInterface.getByIndex(i);
                          if (ni == null) {
                            System.err.println("Interface #" + i + " does not exist.");
                          }
                          else System.out.println("Interface #" + i + " : "+ ni);
                        }
                } catch (SocketException ex) {
                        System.err.println("Could not list network interfaces." );
                }
        }
}
```

Getters

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

getInetAddresses()

public Enumeration getInetAddresses()

- A single network interface may be bound to more than one IP address.
- The getInetAddresses() method returns a java.util.Enumeration containing an InetAddress object for each IP address the interface is bound to.

getInetAddresses() Example: NITester4.java

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

public class NITester4 {
        public static void main(String[] args) {
            try {
                NetworkInterface ni = NetworkInterface.getByName("lo");
                if (ni == null) System.err.println("No such interface");
                else {
                    System.out.println(ni); // uses toString()
                    Enumeration addresses = ni.getInetAddresses();
                    while (addresses.hasMoreElements())
                        System.out.println(addresses.nextElement());
                }
            } catch (SocketException ex) {
                System.err.println("Could not list sockets.");
            }
        }
}
```

getName() & getDisplayName()

public String getName()

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

public String getDisplayName()

- The getDisplayName() method returns a more <u>user-friendly name</u> for the particular NetworkInterface, such as "*Ethernet Card 0*"

- Revise InterfaceLister.java to include these 2 methods

getName

```
public String getName()
```

Get the name of this network interface.

Returns:

the name of this network interface

http://docs.oracle.com/javase/7/docs/api/java/net/NetworkInterfac
NetworkInterface (Java Plat...

getDisplayName

```
public String getDisplayName()
```

Get the display name of this network interface. A display name is a human readable String describing the network device.

Returns:

a non-empty string representing the display name of this network interface, or null if no display name is available.

getByName

```
public static NetworkInterface getByName(String name)
                                 throws SocketException
```

Searches for the network interface with the specified name.

Parameters:

`name` - The name of the network interface.

Returns:

InetAddressExample.java

http://cs.ecs.baylor.edu/~donahoo/practical/JavaSockets2/code/ cs.ecs.baylor.edu

File Edit View Favorites Tools Help

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

public class InetAddressExample {

  public static void main(String[] args) {

    // Get the network interfaces and associated addresses for this host
    try {
      Enumeration<NetworkInterface> interfaceList = NetworkInterface.getNetworkInterfaces();
      if (interfaceList == null) {
        System.out.println("--No interfaces found--");
      } else {
        while (interfaceList.hasMoreElements()) {
          NetworkInterface iface = interfaceList.nextElement();
          System.out.println("Interface " + iface.getName() + ":");
          Enumeration<InetAddress> addrList = iface.getInetAddresses();
          if (!addrList.hasMoreElements()) {
            System.out.println("\t(No addresses for this interface)");
          }
          while (addrList.hasMoreElements()) {
            InetAddress address = addrList.nextElement();
            System.out.print("\tAddress "
                + ((address instanceof Inet4Address ? "(v4)"
                      : (address instanceof Inet6Address ? "(v6)" : "(?)"))));
            System.out.println(": " + address.getHostAddress());
          }
        }
      }
    } catch (SocketException se) {
      System.out.println("Error getting network interfaces:" + se.getMessage());
    }

    // Get name(s)/address(es) of hosts given on command line
    for (String host : args) {
      try {
        System.out.println(host + ":");
        InetAddress[] addressList = InetAddress.getAllByName(host);
        for (InetAddress address : addressList) {
          System.out.println("\t" + address.getHostName() + "/" + address.getHostAddress());
        }
      } catch (UnknownHostException e) {
        System.out.println("\tUnable to find address for " + host);
      }
    }
  }
}
```

Now that we have learned Enumeration & NetworkInterface, it's time to check out this part

InetAddressExample.java

http://cs.ecs.baylor.edu/~donahoo/practical/JavaSockets2/code/ cs.ecs.baylor.edu

File Edit View Favorites Tools Help

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

public class InetAddressExample {

  public static void main(String[] args) {

    // Get the network interfaces and associated addresses for this host
    try {
      Enumeration<NetworkInterface> interfaceList = NetworkInterface.getNetworkInterfaces();
      if (interfaceList == null) {
        System.out.println("--No interfaces found--");
      } else {
        while (interfaceList.hasMoreElements()) {
          NetworkInterface iface = interfaceList.nextElement();
          System.out.println("Interface " + iface.getName() + ":");
          Enumeration<InetAddress> addrList = iface.getInetAddresses();
          if (!addrList.hasMoreElements()) {
            System.out.println("\t(No addresses for this interface)");
          }
          while (addrList.hasMoreElements()) {
            InetAddress address = addrList.nextElement();
            System.out.print("\tAddress "
                + ((address instanceof Inet4Address ? "(v4)"
                      : (address instanceof Inet6Address ? "(v6)" : "(?)"))));
            System.out.println(": " + address.getHostAddress());
          }
        }
      }
    } catch (SocketException se) {
      System.out.println("Error getting network interfaces:" + se.getMessage());
    }

    // Get name(s)/address(es) of hosts given on command line
    for (String host : args) {
      try {
        System.out.println(host + ":");
        InetAddress[] addressList = InetAddress.getAllByName(host);
        for (InetAddress address : addressList) {
          System.out.println("\t" + address.getHostName() + "/" + address.getHostAddress());
        }
      } catch (UnknownHostException e) {
        System.out.println("\tUnable to find address for " + host);
      }
    }
  }
}
```

See: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op2.html