PGT: INTERNET ADDRESSES

Mohamed Elshaikh

Faculty of Electronics Engineering Technology – UniMAP (FTKEN-UniMAP)

Objectives

- **DISCUSS** and **SHOW** Java programs interact with the Domain Name System through the InetAddress class.
  - The InetAddress Class
  - Inet4Address and Inet6Address
  - The NetworkInterface Class
  - Spam Check

Nodes in a network

nodes
All devices that are connected on a network.

host
A node which is a computer

Node/host 2
IP address

Node/host 3
IP address

Node/host 4
IP address

Node/host 1
IP address

Node 0
IP address

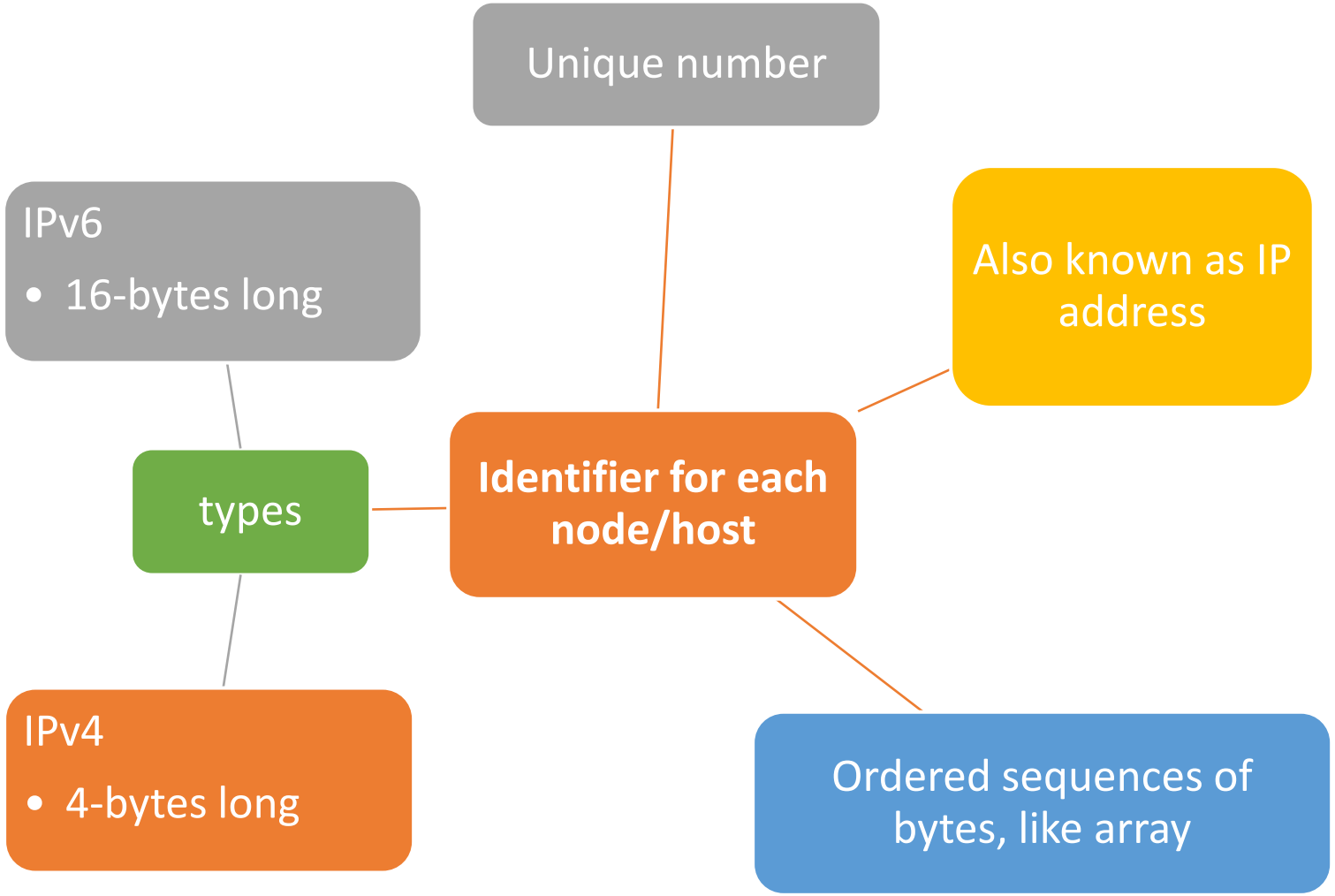
Node/host 5
IP address

Node/host 8
IP address

Node 7 IP address

Node/host 6
IP address

Internet Addresses



IPv4 - Internet Protocol Version 4

protocol in data communication	fourth revision of Internet Protocol (IP)
	widely used over different kinds of networks.
address is normally written as	four unsigned bytes,
	each ranging from 0 to 255,
	with the most significant byte first.
Bytes are separated by periods	for the convenience of human eyes
	dotted quad format
	Eg: *login.ibiblio.org* = 152.19.134.132.
MSB: most significant bit	a.k.a high-order bit, left-most bit
	Usually the bit that is transmitted first in a sequence
	e.g.: in the binary number 1000, MSB is 1. In 0111, MSB is 0.

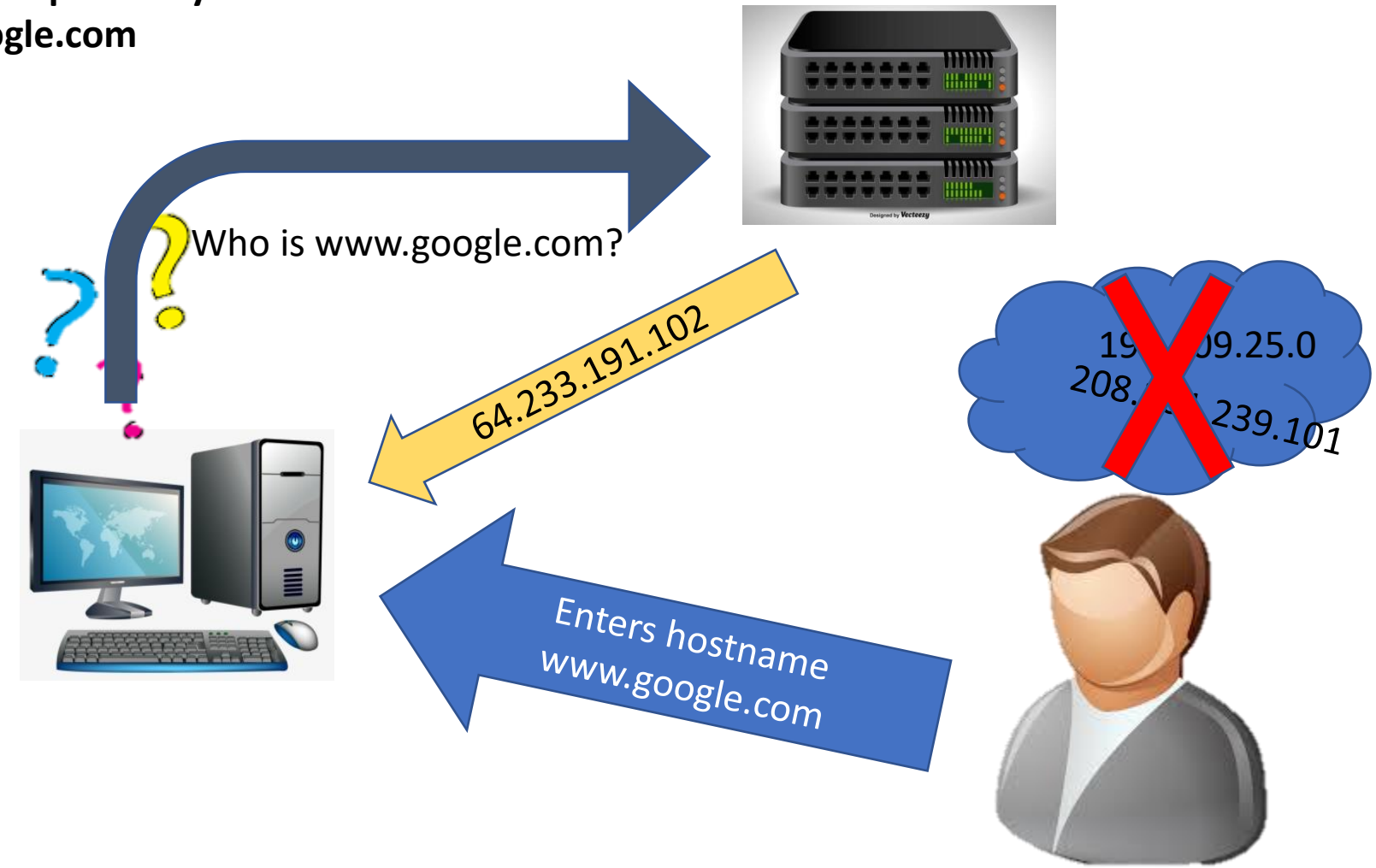
IPv6 - Internet Protocol Version 6

Protocol in data communication	sixth revision of Internet Protocol (IP)
	successor to **IPv4**.
Address normally written as	eight blocks of four hexadecimal digits
Bytes are separated by colons (:)	Eg: `www.hamiltonweather.tk` = **`2400:cb00:2048:0001:0000:0000:6ca2:c665`**
	Leading zeros do not need to be written.
	Therefore, the address above can be written as **`2400:cb00:2048:1:0:0:6ca2:c665`**.
A double colon indicates multiple zero blocks.	Eg: **`2001:4860:4860:0000:0000:0000:0000:8888`**
	can be written more compactly as **`2001:4860:4860::8888`**

DNS (Domain Name Servers)

humans can remember **hostnames**
e.g.
www.unimap.edu.my
www.google.com

Computers only recognizes **IP address**
e.g. 165.200.124.20



DNS (Domain Name Servers)

- Internet's equivalent of a phone book.
  - maintain a directory of domain names
  - translate to Internet Protocol (IP) addresses.

Servers hostname	At least one hostname
Clients hostname,	Often do not have hostnames
	especially if IP address is dynamically assigned at startup.
One hostname, multiple IP addresses.	Not always, but sometimes can exist
	the DNS server randomly choose machines to respond to each request.
	Mostly used for very high-traffic websites, which has several machines.

The `InetAddress` Class

In Java programming, we use **InetAddress** class

- represents an IP address,
- both IPv4 and IPv6.

Provides methods to get the IP of any hostname.

It is used by most of the other networking classes

- Socket,
- ServerSocket,
- URL,
- DatagramSocket,
- DatagramPacket, and more.

Usually, it includes:

- a hostname
- and an IP address

Creating New InetAddress Objects

- **no public constructors** in the InetAddress class.
- Instead, InetAddress has static factory methods
  - connect to a DNS server **to resolve a hostname**.
  - The most common is `InetAddress.getByName()`.
- Eg:

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

- It makes a connection to the local DNS server
  - to look up the name and the numeric address.
- If the DNS server can't find the address,
  - this method throws an **UnknownHostException** (a subclass of IOException)

Example :

Create an InetAddress object

```
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");

       }

   }

}
```

Output:

```
www.oreilly.com/208.201.239.36
```

Reverse Lookup

- 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.getByName()`
- Eg:

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

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

Hostname with Multiple Addresses

- If the hostname supplied has multiple addresses (referring to more than one machine),which one `getHostName()` returns is indeterminate.
- If needed, call `getAllByName()` to get all the addresses of a host, which returns in an array.
- Eg:

```
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");
}
```

Address of the Local Machine

- The `getLocalHost()` method returns an InetAddress object for the host on which your code is running.
- Eg:

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

- This method tries to connect to DNS to get a real hostname and IP address but if that fails it may return the loop-back address instead.
- Without internet connection, the result would probably be → hostname “localhost” and address “127.0.0.1”.

Example :

Find the address of the local machine

```
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.");

        }

    }

}
```

Output (when this program is run on titan.oit.unc.edu server) :

```
titan.oit.unc.edu/152.2.22.14
```

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:

Method	Description
`public String getHostName()`	Returns a String that contains the name of the host with the IP address represented by this InetAddress object.
`public String getCanonicalHostName()`	Gets the fully qualified domain name for this IP address.
`public byte[] getAddress()`	Returns an IP address as an array of bytes in network byte order.
`public String getHostAddress()`	Returns a string containing the dotted quad format of the IP address.

Determine IP version

- 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 {
  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;
  }
}
```

Address Types

- Java includes 10 methods for testing whether an InetAddress object meets any of these criteria:

Method	Description
public boolean isAnyLocalAddress()	returns true if the address is a wildcard address
public boolean isLoopbackAddress()	returns true if the address is the loopback address
public boolean isLinkLocalAddress()	returns true if the address is an IPv6 link-local address
public boolean isSiteLocalAddress()	returns true if the address is an IPv6 site-local address
public boolean isMulticastAddress()	returns true if the address is a multicast address
public boolean isMCGlobal()	returns true if the address is a global multicast address
public boolean isMCNodeLocal()	returns true if the address is an interface-local multicast address
public boolean isMCLinkLocal()	returns true if the address is a subnet-wide multicast address
public boolean isMCSiteLocal()	returns true if the address is a site-wide multicast ad- dress
public boolean isMCOrgLocal()	returns true if the address is an organization-wide multicast 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
```

- `Inet4Address` overrides several of the methods in `InetAddress` but doesn't change their behavior in any public way.

- `Inet6Address` is similar, with one new method not present in the superclass, `isIPv4CompatibleAddress()` → returns true if and only if the address is an IPv4 address stuffed into an IPv6 container—which means only the last four bytes are nonzero.

The `NetworkInterface` Class

- The `NetworkInterface` provides configuration and statistical information for a network interface.
- Represents a Network Interface made up of a name, and a list of IP addresses assigned to this interface. Interfaces are normally known by names such as "le0".
- This can either be
  - a **physical interface** such as an additional Ethernet card (common on firewalls and routers) , or
  - a **virtual interface** bound to the same physical hardware as the machine's other IP addresses.
- Provides methods to list all the local addresses and to create InetAddress objects from them.
- These InetAddress objects can then be used to create sockets, server sockets, and so forth.

The `NetworkInterface` Class
Methods

- Because NetworkInterface objects represent physical hardware and virtual addresses, **they cannot be constructed arbitrarily.**

- There are several static factory methods that return the `NetworkInterface` object associated with a particular network interface :
    - `getByName()`
    - `getByInetAddress()`
    - `getNetworkInterfaces()`

- You can ask for a NetworkInterface by IP address, by name, or by enumeration.

The `getByName()` method

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

- 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.
- Eg – Finding network interface on UNIX machine :

```
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.");
}
```

The `getByInetAddress()` method

```
public static NetworkInterface getByInetAddress(InetAddress address) throws SocketException
```

- 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`.
- E.g. – Find network interface for the local loopback address

```
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.");
}
```

The `getNetworkInterfaces()` method

```
public static Enumeration getNetworkInterfaces() throws SocketException
```

- Returns a `java.util.Enumeration` listing all the network interfaces on the local host.

- Eg – list all network interfaces on the local host

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

Output (the result of running this on the IBiblio login server):

```
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;
```

The `NetworkInterface` Class
Getter Methods

- Having NetworkInterface object, you can inquire about its IP address and name.

Method	Description
`public Enumeration getInetAddresses()`	Returns a java.util.Enumeration containing an InetAddress object for each IP address the interface is bound to.
`public String getName()`	Returns the name of a particular NetworkInterface object, such as eth0 or lo.
`public String getDisplayName()`	Returns a more human-friendly name for the particular NetworkInterface—E.g: “Ethernet Card 0” ,“Local Area Connection” or “Local Area Connection 2.”

SpamCheck

- Monitor spammer using DNS
- To find out if a certain IP address is a known spammer:
  1. reverse the bytes of the address,
  2. add the domain of the blackhole service, and
  3. look it up.
- **If the address is found, it is a spammer**.
- E.g.: 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.spamhaus.org*.
- If the DNS query succeeds (returns 127.0.0.2) → the host is known to be a spammer.
- If the lookup fails → it throws an `UnknownHostException` (also means it is not a spammer)

Example : SpamCheck

```
package lec4s;
import java.net.*;
public class Lec4s {
 public static final String BLACKHOLE = "sbl.spamhaus.org";
 public static void main(String[] args) throws UnknownHostException
{
String[] addList = {"207.34.56.23","125.12.32.4","130.130.130.130"};
        for (String octet : addList)
        {
          if (isSpammer(octet)) {
              System.out.println(octet + " is a known spammer.");
            } else {
              System.out.println(octet + " appears legitimate.");
            }
        }
    }
 private static boolean isSpammer(String arg) {…}
}
```

```
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;
          }
     }
```

```
$ 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.
```

Sample Output:

THANK YOU