# UNIT-2: INTERNET ADDRESS

2.1 The InetAddress Class: Creating new inetAddress object, getter

2.2 Methods, Address Types, Testing Reachability and Object Method

2.3 Inet4Address and Inet6Address

2.4 The network interface class: factory method and Getter method

2.5 Some Useful Program: spam Check, Processing Web Server Logfiles

Krishna Pd. Acharya

# INTERNET ADDRESS

- Devices connected to the Internet are called nodes. Nodes that are computers are called hosts. Each node or host is identified by at least one unique number called an Internet address or an IP address. Most current IP addresses are 4-byte-long IPv4 addresses. However, a small but growing number of IP addresses are 16-byte-long IPv6 addresses. (4 and 6 refer to the version of the Internet Protocol, not the number of the bytes in the address.) Both IPv4 and IPv6 addresses are ordered sequences of bytes, like an array. They aren't numbers, and they aren't ordered in any predictable or useful sense.
- An IPv4 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. For example, the address for www.google.com is 142.250.193.142. This is called the dotted quad format.
- An IPv6 address is normally written as eight blocks of four hexadecimal digits separated by colons. For example, at the time of this writing, the address of www.google.com is 2001:4860:4860::8844. Leading zeros do not need to be written.

## THE INETADDRESS CLASS: CREATING NEW INETADDRESS OBJECT, GETTER

- In summary, Internet addresses (IP addresses) are unique numerical labels that are assigned to each device connected to a computer network that uses the Internet Protocol for communication. They are used to identify and locate devices on a network and play a key role in routing and packet forwarding.
- The InetAddress class in Java is a part of the java.net package and it represents an Internet Protocol (IP) address. It provides methods for working with IP addresses, including getting the hostname and IP address of a remote host, and resolving IP addresses to hostnames and vice versa.
- The class is also used to check if an IP address is reachable or not. It's worth mentioning that the **InetAddress** class is an abstract class, which means you cannot create an object of this class directly, but you can use its subclasses such as Inet4Address, Inet6Address.

# CREATING NEW INETADDRESS OBJECT AND GETTER

- The InetAddress class represents an IP address. To create a new InetAddress object, you can use the static method getByName(String host) or getLocalHost().

```
InetAddress address = InetAddress.getByName(host: "www.mechicampus.edu.np");
System.out.println("IP address: " + address.getHostAddress());
```

```
InetAddress localHost = InetAddress.getLocalHost();
System.out.println("Local hostname: " + localHost.getHostName());
```

## THE INETADDRESS CLASS: CREATING NEW INETADDRESS OBJECT, GETTER

| Method | Description |
|---|---|
| ➢ **getByName**(String hostname) | Returns an InetAddress object for the specified hostname. |
| ➢ **getByAddress**(byte[] addr) | Returns an InetAddress object for the specified IP address. |
| ➢ **getLocalHost**() | Returns the local host. |
| ➢ **getHostName**() | Returns the hostname of this IP address. |
| ➢ **getHostAddress**() | Returns the IP address in string form. |
| ➢ **isReachable**(int timeout) | Tests if this InetAddress is reachable. |
| ➢ **isReachable**(NetworkInterface netif, int ttl, int timeout) | Test whether that address is reachable. |
| ➢ **getAllByName**(String host) | Returns an array of all IP addresses associated with a given host name. |
| ➢ **getByAddress**(byte[] addr, String hostname) | Returns an InetAddress object for the given IP address and hostname. |

# EXAMPLE PROGRAM

```java
import java.net.*;

public class Google {
    Run | Debug
    public static void main(String[] args) throws UnknownHostException {
        try {
            InetAddress address = InetAddress.getByName(host: "www.mechicampus.edu.np");
            System.out.println("getByName():" + address);
            InetAddress localladdr = InetAddress.getLocalHost();
            System.out.println("getLocalHost():" + localladdr);
            InetAddress[] addresses = InetAddress.getAllByName(host: "www.mechicampus.edu.np");
            for (InetAddress addritem : addresses) {
                System.out.println(addritem);
            }
        } catch (UnknownHostException ex) {
            System.out.println(x: "Could not find mechicampus.com ");
        }
    }
}
```

# EXAMPLE PROGRAM

```java
import java.net.*;
public class InetAddressExample {
    Run | Debug
    public static void main(String[] args) throws Exception {
        try {
            // Resolve the IP address of a host
            InetAddress address = InetAddress.getByName(host: "www.mechicampus.edu.np");
            System.out.println("IP address: " + address.getHostAddress());

            // Get the hostname of the local machine
            InetAddress localHost = InetAddress.getLocalHost();
            System.out.println("Local hostname: " + localHost.getHostName());

            // Check if an IP address is reachable
            if (address.isReachable(timeout: 5000)) {
                System.out.println(x: "www.mechicampus.edu.np is reachable");
            } else {
                System.out.println(x: "www.mechicampus.edu.np is not reachable");
            }
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

# INET4ADDRESS AND INET6ADDRESS

- Inet4Address and Inet6Address
- Inet4Address and Inet6Address are subclasses of the InetAddress class and represent IPv4 and IPv6 addresses, respectively. An IPv4 address is represented as four decimal values separated by dots (e.g., 192.168.1.1), while an IPv6 address is represented as eight groups of hexadecimal values separated by colons (e.g., 2001:0db8:85a3:0000:0000:8a2e:0370:7334).
- You can use the following methods to determine the type of an InetAddress object:
- X instanceof Inet4Address
- Y instanceof Inet6Address

```
if (address instanceof Inet4Address) {
    System.out.println(x: "Type: IPv4");
} else if (address instanceof Inet6Address) {
    System.out.println(x: "Type: IPv6");
}
```

# TO CHECK IP4 OR IPV6

```java
import java.net.*;
import java.net.UnknownHostException;
public class IdentifyIPV46 {
    Run | Debug
    public static void main(String[] args) {
        try {
            InetAddress address = InetAddress.getByName(host: "www.google.com");
            System.out.println("Host Name: " + address.getHostName());
            System.out.println("IP Address: " + address.getHostAddress());

            if (address instanceof Inet4Address) {
                System.out.println(x: "Type: IPv4");
            } else if (address instanceof Inet6Address) {
                System.out.println(x: "Type: IPv6");
            }
        } catch (UnknownHostException e) {
            System.err.println(x: "Cannot resolve the host name.");
            e.printStackTrace();
        }
    }
}
```

# NETWORK INTERFACE CLASS( FACTORY VS GETTER)

- 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.
- A factory method is a design pattern used to create objects without specifying the exact class of object that will be created. In the context of the network interface class, a factory method could be used to create new instances of the network interface class without having to specify the exact type of network interface that is being created. This can be useful when dealing with different types of network interfaces, such as Ethernet, Wi-Fi, or Bluetooth, which may have different implementation details.
- A getter method, on the other hand, is a method used to retrieve the value of an attribute of an object. In the context of the network interface class, a getter method could be used to retrieve information about the network interface, such as its MAC address, IP address, or speed. Getter methods can be useful for accessing information about an object in a controlled way, allowing the programmer to ensure that the information is being accessed and used correctly.
- In summary, a factory method could be used to create new instances of the network interface class without having to specify the exact type of network interface being created, while a getter method could be used to retrieve information about the network interface object. Both of these methods can be useful for working with the network interface class and its instances.

# NETWORK INTERFACE CLASS( FACTORY VS GETTER)

CarFactory.java > CarFactory

```java
class CarFactory {
    private String make;

    public CarFactory(String make) {
        this.make = make;
    }

    public static CarFactory create(String make) {
        if (make.equals(anObject:"Ford")) {
            return new CarFactory(make);
        } else if (make.equals(anObject:"Netavi")) {
            return new CarFactory(make);
        } else {
            throw new IllegalArgumentException("Unsupported make: " + make);
        }
    }

    Run | Debug
    public static void main(String[] args) {
        CarFactory mustang = CarFactory.create(make:"Ford");
        System.out.println(mustang.make);
        CarFactory cg = CarFactory.create(make:"Netavi");
        System.out.println(cg.make);
    }
}
```

# NETWORK INTERFACE CLASS( FACTORY VS GETTER)