Unit 2

Java InetAddress class

Java InetAddress class represents an IP address. The java.net.InetAddress class provides methods to get the IP of any host name *for example* www.google.com, www.facebook.com, etc. An IP address is represented by 32-bit or 128-bit unsigned number. An instance of InetAddress represents the IP address with its corresponding host name. There are two types of addresses: Unicast and Multicast. The Unicast is an identifier for a single interface whereas Multicast is an identifier for a set of interfaces. Moreover, InetAddress has a cache mechanism to store successful and unsuccessful host name resolutions.

IP Address

- An IP address helps to identify a specific resource on the network using a numerical representation.
- Most networks combine IP with TCP (Transmission Control Protocol). It builds a virtual bridge among the destination and the source.

Uses Of Network Programming

1. Identifying hosts: The **InetAddress** class provides methods to get the IP address and hostname of a host. This is useful for identifying remote hosts on a network when establishing network connections.
2. Resolving hostnames: The **InetAddress** class provides methods to resolve a hostname to an IP address, or an IP address to a hostname. This is important for establishing connections between machines on a network, where the IP address of the remote host is required.
3. Creating sockets: The **InetAddress** class is used to create a socket for a given IP address and port number. This is useful for establishing communication channels between machines on a network.
4. Network programming: The **InetAddress** class is a fundamental class in Java's networking package and is used extensively in network programming. It is used to represent the local machine's IP address as well as remote machine's IP addresses. With the help of **InetAddress** class, network programs can identify the host machines and establish communication channels between them.
5. Checking connectivity: The **InetAddress** class provides methods to check whether a remote host is reachable over the network. This is useful for verifying connectivity between machines before attempting to establish a connection.

Getter Method

In network programming, a getter method is a method used to retrieve information about network-related objects. These methods allow programmers to obtain information such as the IP address, hostname, port number, or other properties of a network connection or device. Getter methods are typically defined as part of a class that represents a network-related object, and they have a return type that corresponds to the type of information being retrieved. These methods are used extensively in Java networking programming to retrieve information about objects such as **InetAddress**, **Socket**, **ServerSocket**, **DatagramSocket**, **URL**, and other network-related objects. Overall, getter methods are a key tool for network programmers to effectively manage and utilize network connections and devices. Here are some of the commonly used getter methods and their uses:

1. **getHostAddress()**: Returns the IP address of a **InetAddress** object as a string.
2. **getHostName()**: Returns the hostname of a **InetAddress** object as a string.
3. **getCanonicalHostName()**: Returns the fully qualified domain name of a **InetAddress** object as a string.
4. **getAddress()**: Returns the raw IP address of a **InetAddress** object as a byte array.

5. **getByName()**: Returns an **InetAddress** object that represents the IP address or hostname specified as a string.
6. **getLocalHost()**: Returns an **InetAddress** object that represents the local host.
7. **getPort()**: Returns the port number of a socket.
8. **getInputStream()**: Returns an input stream for reading data from a socket.
9. **getOutputStream()**: Returns an output stream for writing data to a socket.
10. **getLocalSocketAddress()**: Returns the local socket address of a socket.
11. **getRemoteSocketAddress()**: Returns the remote socket address of a socket.

```
import java.io.IOException;
import java.net.*;
public class NetworkGetterExample {
      public static void main(String[] args) {
 try {
     // Get InetAddress object for localhost
     InetAddress localhost = InetAddress.getLocalHost();
     System.out.println("Localhost IP address: " + localhost.getHostAddress());
     System.out.println("Localhost hostname: " + localhost.getHostName());
       System.out.println("Localhost fully qualified domain name: " + localhost.getCanonicalHostName());
       System.out.println("Localhost raw IP address: " + new String(localhost.getAddress()));
     // Get InetAddress object for a specific hostname or IP address
     InetAddress google = InetAddress.getByName("www.google.com");
     System.out.println("Google IP address: " + google.getHostAddress());
     System.out.println("Google hostname: " + google.getHostName());
       System.out.println("Google fully qualified domain name: " + google.getCanonicalHostName());
     System.out.println("Google raw IP address: " + new String(google.getAddress()));
    // Create a socket and get its properties
     Socket socket = new Socket("www.google.com", 80);
     System.out.println("Socket local address: " + socket.getLocalSocketAddress());
     System.out.println("Socket remote address: " + socket.getRemoteSocketAddress());
     System.out.println("Socket port number: " + socket.getPort());
     System.out.println("Socket input stream: " + socket.getInputStream());
     System.out.println("Socket output stream: " + socket.getOutputStream());
     // Close the socket
     socket.close();
 } catch (IOException e) {
     e.printStackTrace();
 }
   }
}
```

Address Types

In network programming, Address Types refer to the type of addressing scheme used to identify a network resource. There are several types of addresses used in network programming, including IP addresses, MAC addresses, and port numbers. An IP address is a unique numerical label assigned to each device connected to a network, used for identifying and communicating with that device. MAC addresses, on the other hand, are unique hardware identifiers assigned to each network interface card, used for identifying devices on a local network. Port numbers are used to identify specific services running on a device, enabling multiple services to run simultaneously on the same device. Some InetAddress class Address methods are:

1. isAnyLocalAddress(): This method tests whether the InetAddress object represents any local address. It returns true if the address is a wildcard or loopback address, and false otherwise. This method is useful for determining if a particular IP address is local to the current host.
2. isLoopbackAddress(): This method tests whether the InetAddress object represents a loopback address. Loopback addresses are used for communication within a single

device. This method is useful for identifying loopback addresses and handling them differently than other addresses.

3. isLinkLocalAddress(): This method tests whether the InetAddress object represents a link-local address. Link-local addresses are used for communication within a single network segment. This method is useful for identifying link-local addresses and handling them differently than other addresses.
4. isSiteLocalAddress(): This method tests whether the InetAddress object represents a site-local address. Site-local addresses are used for communication within a single organization. This method is useful for identifying site-local addresses and handling them differently than other addresses.
5. isMCGlobal(): This method tests whether the InetAddress object represents a global multicast address. Global multicast addresses are used for one-to-many communication across different networks. This method is useful for identifying global multicast addresses and handling them differently than other addresses.
6. isMCOrgLocal(): This method tests whether the InetAddress object represents an organization-local multicast address. Organization-local multicast addresses are used for one-to-many communication within a single organization. This method is useful for identifying organization-local multicast addresses and handling them differently than other addresses.
7. isMCSiteLocal(): This method tests whether the InetAddress object represents a site-local multicast address. Site-local multicast addresses are used for one-to-many communication within a single site or network. This method is useful for identifying site-local multicast addresses and handling them differently than other addresses.
8. isMCLinkLocal(): This method tests whether the InetAddress object represents a link-local multicast address. Link-local multicast addresses are used for one-to-many communication within a single network segment. This method is useful for identifying link-local multicast addresses and handling them differently than other addresses.
9. isMCNodeLocal(): This method tests whether the InetAddress object represents a node-local multicast address. Node-local multicast addresses are used for one-to-many communication within a single device. This method is useful for identifying node-local multicast addresses and handling them differently than other addresses.

```
import java.net.InetAddress;
public class IPAddressTest {
  public static void main(String[] args) {
    try {
      InetAddress address = InetAddress.getByName("localhost");
      System.out.println("Host Address: " + address.getHostAddress());
      if (address.isAnyLocalAddress()) {
        System.out.println("This is a wildcard or loopback address");
      }
      if (address.isLoopbackAddress()) {
        System.out.println("This is a loopback address");
      }
      if (address.isLinkLocalAddress()) {
        System.out.println("This is a link-local address");
      }
      if (address.isSiteLocalAddress()) {
        System.out.println("This is a site-local address");
      }
      if (address.isMCGlobal()) {
        System.out.println("This is a global multicast address");
      }
```

```
            if (address.isMCOrgLocal()) {
                System.out.println("This is an organization-local multicast address");
            }
            if (address.isMCSiteLocal()) {
                System.out.println("This is a site-local multicast address");
            }
            if (address.isMCLinkLocal()) {
                System.out.println("This is a link-local multicast address");
            }
            if (address.isMCNodeLocal()) {
                System.out.println("This is a node-local multicast address");
            }
        } catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

Testing Reachability

In the InetAddress class of Java, the method isReachable() is used to test the reachability of a particular IP address. This method can be used to determine whether a particular host is available on the network, and whether it is possible to connect to it using a socket. The method takes an optional timeout value, which specifies the maximum amount of time to wait for a response from the host. If the host is reachable, the method returns true; otherwise, it returns false. This method is useful for testing the availability of hosts before attempting to connect to them, and for detecting network issues such as connectivity problems or routing failures. It can also be used to measure network latency by timing the duration of the method call.

```
import java.net.InetAddress;
public class ReachabilityTest {
    public static void main(String[] args) {
        try {
            String hostname = "www.google.com";
            InetAddress address = InetAddress.getByName(hostname);

            if (address.isReachable(5000)) {
                System.out.println("Host " + hostname + " is reachable");
            } else {
                System.out.println("Host " + hostname + " is not reachable");
            }
        } catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

Object Method

In network programming, object methods are functions that are associated with an object and are used to interact with the network. For example, the Socket class in Java provides object methods such as connect(), getInputStream(), and getOutputStream() that are used to establish a connection to a server, read data from the server, and write data to the server, respectively. Object methods are important in network programming because they provide a structured and organized way to interact with the network. By using object methods, developers can create robust and scalable networked applications that can communicate with other applications over the network. Object methods also provide a level of abstraction that simplifies network programming and allows developers to focus on the high-level functionality of their applications rather than the low-level details of network communication.

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

```
public class Client {
    public static void main(String[] args) {
        try {
            // Create a new socket object and connect to the server at localhost on port 8080
            Socket socket = new Socket("localhost", 8080);
            // Get the input and output streams of the socket
            InputStream inputStream = socket.getInputStream();
            OutputStream outputStream = socket.getOutputStream();
            // Send a message to the server by writing to the output stream
            String message = "Hello, server!";
            outputStream.write(message.getBytes());
            // Read the response from the server by reading from the input stream
            byte[] buffer = new byte[1024];
            int length = inputStream.read(buffer);
            String response = new String(buffer, 0, length);
            // Print the response to the console
            System.out.println("Server response: " + response);
            // Close the socket to release the resources
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Network Interface Class

NetworkInterface class that 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

Factory Method

The network interface class factory method is a design pattern that provides a way to create instances of network interface classes without exposing the implementation details to the client. This pattern involves using a static method, typically called a factory method, within the network interface class to create and return instances of the class. The factory method allows the client to create objects without having to know the specific class or implementation being used, which makes the code more modular and easier to maintain. Additionally, the factory method can provide a centralized location for managing object creation and can enforce any required constraints or policies on object creation. The advantages of using the network interface class factory method pattern include increased flexibility and maintainability, as well as improved encapsulation and code reuse. It can also improve performance by reducing the overhead associated with object creation and initialization. Overall, this design pattern is a useful tool for managing object creation in complex software systems.

```
import java.net.*;
public class NetworkInterfaceFactoryExample {
    public static void main(String[] args) throws SocketException, UnknownHostException {
        InetAddress address = InetAddress.getByName("192.168.1.100");
        NetworkInterface iface = NetworkInterface.getByInetAddress(address);
        if (iface == null) {
            System.out.println("No interface found for address: " + address.getHostAddress());
        } else {
            System.out.println("Interface found: " + iface.getName());
        }
    }
}
```

SpamChecker

SpamChecker is a tool used in network programming to identify and filter out unwanted messages, commonly known as spam, from incoming data streams. It analyzes the content of each message using various algorithms and rules to detect characteristics that indicate spam, such as suspicious keywords, patterns, or sender information. Once identified, SpamChecker can take action to block or mark the spam message for further review, preventing it from reaching its intended recipient and reducing the risk of security breaches, malware infections, or unwanted advertising. The effectiveness of a SpamChecker depends on its accuracy, speed, and flexibility to adapt to new spamming techniques and patterns. Here are some of the key uses of SpamChecker:

1. Email filtering: SpamChecker is widely used in email systems to filter out spam messages from users' email inboxes. The tool helps to reduce the risk of phishing attacks, malware infections, and unwanted advertising. SpamChecker can also mark suspicious emails as spam, move them to a spam folder, or block them altogether.
2. Messaging platform filtering: Messaging platforms, such as chat apps and social media platforms, use SpamChecker to prevent users from receiving spam messages. The tool can scan messages for spam characteristics, such as suspicious links or keywords, and block or quarantine them.
3. Website form filtering: Websites that allow users to fill out forms, such as contact forms or comment sections, use SpamChecker to prevent spam submissions. The tool can analyze the content of the form submissions and block or flag submissions that contain spam characteristics, such as links or suspicious keywords.
4. Network security: SpamChecker can also be used as part of a network security solution to prevent spam-related security threats, such as phishing attacks, malware infections, or spam-based botnet attacks. The tool can scan network traffic for spam characteristics and block or quarantine suspicious traffic.
5. User protection: SpamChecker helps protect users from unwanted and potentially harmful messages, such as phishing emails or malware-infected attachments. By filtering out spam, the tool reduces the risk of users falling victim to spam-related scams or attacks.

```
import java.io.*;
import java.net.*;
public class SpamChecker {
  public static void main(String[] args) throws Exception {
    String host = "example.com"; // replace with your host name or IP address
    int port = 80; // replace with the port number used by your application
    String message = "Hello World!"; // replace with the message to be sent
    boolean isSpam = checkSpam(message); // check if the message is spam
    if (!isSpam) {
      sendMessage(host, port, message); // send the message if it is not spam
    } else {
      System.out.println("The message is identified as spam and will not be sent.");
    }
  }
  private static boolean checkSpam(String message) {
    return false; // replace with your spam detection result
  }
  private static void sendMessage(String host, int port, String message) throws Exception {
    Socket socket = new Socket(host, port); // create a socket to connect to the host
    OutputStream output = socket.getOutputStream(); // get the output stream to send data
    PrintWriter writer = new PrintWriter(output, true); // wrap the output stream with a writer
    writer.println(message); // send the message
    System.out.println("The message has been sent successfully.");
    writer.close(); // close the writer and the socket
```

```
        output.close();
        socket.close();
    }
}
```
Processing web log files involves analyzing the data generated by web servers when they interact with visitors. When a visitor requests a webpage, the web server generates a log file that contains information about the request, including the visitor's IP address, the date and time of the request, the requested URL, the HTTP status code, and other details. These log files can be used to gain insights into website traffic, visitor behavior, and website performance. The process of analyzing web log files typically involves parsing the data to extract relevant information, transforming the data into a more usable format, and loading the data into a database or visualization tool for further analysis. Techniques such as data mining, machine learning, and statistical analysis can be applied to web log files to identify patterns and trends, optimize website performance, and improve the visitor experience.

Process web log files can provide valuable insights about website visitors and their behavior. Here are some of the common uses of processing web log files:

1. Website performance analysis: Web log files can provide data on website performance, such as page load times, error rates, and server response times. This information can help website owners and developers identify areas for improvement and optimize the website for better user experience.
2. Traffic analysis: Web log files contain information about website visitors, including their IP addresses, location, device type, and browser type. Analyzing this data can provide insights into visitor demographics, behavior patterns, and traffic sources.
3. Search engine optimization (SEO): Web log files can help website owners and marketers optimize their content for search engines by providing information on keywords used by visitors to find the site, the pages they visit, and the time spent on each page.
4. Security analysis: Web log files can be used to detect and investigate security incidents, such as hacking attempts, malware infections, and denial of service attacks. By analyzing web log files, security experts can identify patterns of suspicious behavior and take appropriate action to protect the website.
5. Advertising analysis: Web log files can provide valuable data for advertising campaigns, such as click-through rates, conversion rates, and visitor demographics. This information can help marketers optimize their ad targeting and messaging to reach their desired audience.
6. Compliance monitoring: Web log files can be used to monitor compliance with regulatory requirements, such as data privacy laws and industry standards. By analyzing web log files, organizations can ensure that their website and associated systems are meeting the necessary standards and requirements.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
public class WebLogServer {
    public static void main(String[] args) throws IOException {
        String fileName = "access.log"; // change this to the path of your log file
        int portNumber = 1234; // change this to the port number you want to use
        try (ServerSocket serverSocket = new ServerSocket(portNumber);
             Socket clientSocket = serverSocket.accept(); // wait for a client to connect
             BufferedReader br = new BufferedReader(new FileReader(fileName)); // read from the log file
             PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true)) { // send data to client
```

```
            String line;
            while ((line = br.readLine()) != null) { // read each line of the log file
                String[] tokens = line.split(" "); // split the line into tokens
                String ip = tokens[0]; // get the IP address from the first token
                out.println(ip); // send the IP address to the client over the socket connection
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```