Basics of Network Programming

1. Socket Overview (Client/Server Architecture)

Java uses the **`java.net`** package for socket programming. A **socket** is one endpoint of a two-way communication link between two programs running on the network.

Client/Server Model

- **Server**: Waits for client requests using a `ServerSocket`.
- **Client**: Initiates communication by connecting to the server using a `Socket`.

Server-side Example:

```
java
CopyEdit
ServerSocket serverSocket = new ServerSocket(5000); //
Listen on port 5000
Socket socket = serverSocket.accept(); // Accept
connection
```

Client-side Example:

```
java
CopyEdit
Socket socket = new Socket("localhost", 5000); //
Connect to server on port 5000
```

2. Reserved Sockets (Port Numbers)

Certain port numbers are **reserved** by the system or by standard protocols:

- **0–1023**: *Well-known ports* (e.g., HTTP: 80, FTP: 21, SSH: 22).
- **1024–49151**: *Registered ports* (for user-defined services).
- **49152–65535**: *Dynamic/private ports* (used for temporary or client-side communication).

Java allows you to bind to a specific port, but ports below 1024 often require admin privileges.

3. Proxy Servers

A **proxy server** acts as an intermediary between the client and the target server, often used for:

- Content filtering
- Security
- Caching

In Java, you can configure a socket to use a proxy using the `Proxy` class:

```
java
CopyEdit
Proxy proxy = new Proxy(Proxy.Type.HTTP, new
InetSocketAddress("proxy.example.com", 8080));
Socket socket = new Socket(proxy);
```

4. Internet Addressing in Java

Java uses the `InetAddress` class to handle IP addressing:

- Represents an IP address (IPv4 or IPv6).
- Can be resolved from hostnames and vice versa.

Examples:

```
java
CopyEdit
InetAddress ip =
InetAddress.getByName("www.google.com");
System.out.println("IP Address: " +
ip.getHostAddress());

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

```
System.out.println("Local Host: " +
localHost.getHostName());
```

Java and the Net: Theory of Networking Classes and Interfaces

Java supports **network communication** using the `java.net` package. This package provides classes and interfaces to handle both **TCP (connection-oriented)** and **UDP (connectionless)** communication.

Purpose of Networking in Java:

- To allow Java programs to **communicate over the internet or a local network**.
- To implement client-server architectures.
- To support modern applications such as browsers, chat apps, and web services.

Key Networking Classes and Interfaces

1. **`Socket`**: Represents a client-side endpoint for TCP communication.
2. **`ServerSocket`**: Represents a server-side socket that listens for client connections.
3. **`DatagramSocket`**: Supports UDP communication, which is connectionless.
4. **`InetAddress`**: Represents an IP address, and provides methods to resolve hostnames and access address details.
5. **`URL` and `URLConnection`**: Enable working with web addresses and establishing connections over HTTP/FTP.
6. **`Proxy`**: Allows configuring network communication through proxy servers.
7. **`NetworkInterface`**: Represents a network interface like Ethernet or Wi-Fi on the local machine.

`InetAddress` Class – Theoretical Concept

What is `InetAddress`?

- It is a class used to represent **IP addresses** (IPv4 and IPv6) in Java.
- It helps in converting between **hostnames (like www.google.com)** and **IP addresses (like 142.250.190.68)**.
- Objects of `InetAddress` are **not directly created** but are **obtained using factory methods**.

Factory Methods of `InetAddress` (Static Methods)

These are used to **create or obtain** `InetAddress` objects:

Method	Description
`getByName(String host)`	Resolves a hostname to an IP address (DNS lookup).
`getAllByName(String host)`	Returns an array of all IP addresses associated with a domain.
`getLocalHost()`	Returns the IP address of the local computer.
`getByAddress(byte[] addr)`	Returns an `InetAddress` from a raw IP byte array.
`getLoopbackAddress()`	Returns the loopback address (127.0.0.1), used for testing within the same machine.

These methods often throw `UnknownHostException` if the hostname cannot be resolved.

Instance Methods of `InetAddress`

These methods allow you to **work with an `InetAddress` object** and get specific information:

Method	Purpose
`getHostAddress()`	Returns the IP address in string format (e.g., "192.168.1.1").
`getHostName()`	Returns the hostname associated with the IP address.
`isReachable(int timeout)`	Checks if the address is reachable (like a ping).
`isLoopbackAddress()`	Checks if the address is a loopback address (e.g., 127.0.0.1).
`isAnyLocalAddress()`	Checks if the address is a wildcard address (0.0.0.0).
`isMulticastAddress()`	Checks if the address is for multicast communication.

Importance of `InetAddress` in Networking

- Allows Java applications to be **network-aware** by resolving and working with hosts.
- Provides an **object-oriented abstraction of IP addresses**.
- Essential for creating **sockets**, connecting to servers, and validating network connections.

Java Networking Theory: TCP/IP and UDP Concepts

Java supports **two main types of communication protocols**:

- **TCP (Transmission Control Protocol)** – Reliable, connection-based.
- **UDP (User Datagram Protocol)** – Fast, connectionless, and unreliable.

These are part of the **TCP/IP protocol suite**, which underlies all internet communication.

1. TCP/IP Client and Server Sockets (Connection-Oriented)

Concept of TCP Sockets

TCP communication uses **sockets** that establish a reliable connection between two endpoints:

- **Client Socket**: Initiates the connection.
- **Server Socket**: Listens for incoming client connections.

Java Classes Used:

- `Socket` – Represents the client-side endpoint.
- `ServerSocket` – Represents the server-side socket that listens for incoming connections.

Working Process (Theory)

1. The **server** creates a `ServerSocket` bound to a specific port (e.g., 5000).
2. The server calls `accept()` which waits for a client to connect.
3. The **client** creates a `Socket` and connects to the server's IP and port.
4. Once connected, both sides can **send and receive data** using input and output streams.
5. TCP ensures that data is **delivered reliably and in order**.

Key Features of TCP in Java:

- Reliable delivery.
- Connection must be established before communication.
- Stream-based data transmission.

2. Datagram Sockets (Connectionless - UDP)

Concept of UDP Sockets

- UDP is a **connectionless** protocol.
- Does **not guarantee delivery**, order, or data integrity.
- Suitable for applications where **speed is more important than reliability** (e.g., streaming, online gaming).

Java Classes Used:

- **`DatagramSocket`** – Used for sending and receiving datagrams.
- **`DatagramPacket`** – Represents the data being sent or received.

Working Process (Theory)

1. A `DatagramSocket` is created on both client and server sides.
2. Data is wrapped in a `DatagramPacket`.
3. The sender uses `send()` to transmit the packet.
4. The receiver uses `receive()` to get the packet.
5. No formal connection is established; messages are sent independently.

Key Features of UDP in Java:

- Lightweight and fast.
- Suitable for broadcast and multicast.
- Useful where occasional data loss is acceptable.

3. DatagramPacket (Used with UDP)

Role of DatagramPacket

- Acts as a **container for data** that is being sent or received.
- Contains:
  - **Byte array** for data
  - **Length of data**

- **Destination/source IP address and port**

Use in UDP Communication

- For sending: the packet is prepared with destination IP and port.
- For receiving: the packet acts as a buffer to store incoming data.

Summary Table

Protocol	Java Class	Type	Features
TCP	`Socket`	Client	Reliable, ordered, stream-based
TCP	`ServerSocket`	Server	Accepts connections from clients
UDP	`DatagramSocket`	Client/Server	Unreliable, fast, connectionless
UDP	`DatagramPacket`	Data Container	Holds data and destination/source address info

Use Case Comparison

Feature	TCP (`Socket`)	UDP (`DatagramSocket`)
Reliability	High	Low
Connection Required	Yes	No
Speed	Slower	Faster
Data Format	Stream (continuous)	Packet (individual chunks)
Suitable For	Chat apps, file transfer	Streaming, games, DNS

1. The `URL` Class (Uniform Resource Locator)

□ Purpose:

The `URL` class represents a **pointer to a resource** on the World Wide Web, such as:

- Web pages (`http://example.com`)
- Files (`ftp://example.com/file.txt`)
- JAR files, or even local resources

□ Concept:

A `URL` object encapsulates:

- **Protocol** (e.g., `http`, `ftp`, `file`)
- **Host name** (e.g., `www.google.com`)
- **Port** (optional)
- **File/path** (e.g., `/index.html`)
- **Query** and **reference** (optional)

□ Key Methods:

Method	Description
`getProtocol()`	Returns the protocol name (e.g., `http`)
`getHost()`	Returns the host name
`getPort()`	Returns the port number (or `-1` if unspecified)
`getFile()`	Returns the file path and query string
`openConnection()`	Opens a connection to the URL
`openStream()`	Returns an input stream to read from the URL

□ Example Concept:

When you create a `URL` object, Java parses the string into its components:

```
java
CopyEdit
URL url = new
URL("https://www.example.com:443/index.html?name=java")
;
```

- Protocol: `https`
- Host: `www.example.com`
- Port: `443`
- File: `/index.html?name=java`

2. The `URLConnection` Class

Purpose:

The `URLConnection` class represents a **communication link** between a Java application and a URL. It provides methods to:

- Read/write data to/from the resource.
- Set request headers.
- Access response metadata (e.g., content type, length).

How it Works:

- You create a `URL` object.
- Call `openConnection()` on the URL to get a `URLConnection` object.
- Then you can configure the connection or read/write data.

Key Features:

- Supports both **input** (reading from the resource) and **output** (sending data to the server).
- Can work with various protocols (`http`, `ftp`, `file`, etc.).
- Allows setting headers, timeouts, and request properties.

□ Important Methods:

Method	Description
`connect()`	Establishes the connection (optional)
`getInputStream()`	Gets an input stream to read from the URL
`getOutputStream()`	Gets an output stream to send data
`setDoOutput(boolean)`	Enables output (used in POST requests)
`getContentType()`	Returns the MIME type of the response
`getContentLength()`	Returns the length of the content
`setRequestProperty(key, val)`	Sets request headers
`getHeaderField(String name)`	Gets the value of a header field

? Summary Comparison

Feature	`URL` Class	`URLConnection` Class
Purpose	Represents the web resource	Represents the connection to the resource
Main Use	Parsing and pointing to resources	Reading/writing data, managing request
Protocol Support	Yes	Yes
Can Send Data	No	Yes (with `setDoOutput(true)`)

Feature	`URL` Class	`URLConnection` Class
Common Use Cases	Create links, resolve addresses	Download files, submit forms, access APIs

-