Java Networking

- INTRODUCTION TO NETWORKING
- TERMINOLOGIES IN NETWORKING
- JAVA NETWORKING

Introduction to Networking

Network: Group or system of connected entities (things, people).

Networking: The actual process of communicating or interacting with others in a network.

Networking in programming: Writing programs that execute across a group of devices in the same network.

Networking Terminologies

Internet Protocol (IP) Address

A number assigned to an entity in a network. Logical, identifies connection of a device.

- Unique.
- Not randomly generated; mathematically produced and allocated by IANA.
- Different types – private, public (static & dynamic).
- Different versions – IPv4, IPv6
  - IPv4 version – set of four numbers, each within the range of 0 and 255.
  - IPv6 – set of eight hexadecimal digits.

Examples:

- 257.108.0.25 😆
- 127.0.0.1 👌
- 2447:odb8:84ed:44e4:0000:8a3e:0270:5774 👌

Networking Terminologies (contd.)

Protocol

Set of communication rules.

Some examples:

- Transmission Control Protocol (TCP): enables programs and device exchange messages over a network.
- File Transfer Protocol (FTP): manages file transfer between devices over a network.
- Teletype Network (Telnet): manages remote device communication.
- Simple Mail Transfer Protocol (SMTP): manages electronic mail/messages over a network.
- Internet Protocol (IP) – yes, IP. Uses TCP. TCP/IP seems to be the most popular.
- Hyper Text Transfer Protocol (HTTP): manages hypertext communication over a network.
- User Datagram Protocol (UDP): an alternative to TCP.

Networking Terminologies (contd.)

Port Number

A port is a communication endpoint/process/service.

To identify a specific port, port number is used.

- Unique
- 16-bit unsigned number, thus range is 0 to 65535.

Media Access Control (MAC) Number

Unique number identifying active devices. Physical, identifies the device.

Socket

A communication link endpoint. Provides communication techniques between devices using TCP.

Java Networking

Why?

Share resources, communicate…

Okay, how?

Over the network; usually same network.

Where?

The ***application layer*** (one of the 7 layer architecture, closest to end-user that indicates protocols and interfaces used by devices in a communications network)

Java Networking (contd.)

The base networking functionalities are provided in the `java.net` package.

The `java.net` package provides support for TCP and UDP

Java offers

- Stream-based communications – enables data streams communications.
- Packet-based communications – enables packet (usually images, audios and video) communications.

Our main focus is on stream-based communication.

Java Networking: stream-based

Communication usually entails request and response, and should be presented in an orderly format. Thus, the **client/server communication model**.

The Client ***requests*** for information to be received, processed or stored. The server receives this request and actions it, then ***responds*** with the result of process to the client.

In this communication model, the client is seen as dumb while the server is the hub of processing, thus, seen as smart.

One of the ways to implement this communication model in Java is the use of **socket-based communication** (socket programming).

In Java, a socket represents an endpoint of connection.

For stream-based communications, **stream sockets** come in handy; uses TCP.

Java Networking: stream-based (contd.)

To implement socket streams in the communication *request-response* model. A socket needs to be implement on the server, as well as on the client.

Steps to implement a server stream socket.

1. Create a `ServerSocket` object.

   ```
   ServerSocket server = new ServerSocket(portNumber);
   ```

2. Listen/Wait for a client connection.

   ```
   Socket connection = server.accept();
   ```

3. Get I/O Streams from the Socket.
   - Receive stream objects from client.
   - Send response to client.

Java Networking: stream-based (contd.)

4. Action the request.
5. Close the connection.
   - Once processing is done, and transmission is complete, the server should close the I/O streams and socket connection.

Steps to implement a client stream socket.

1. Create a socket connection to the server socket.
2. Get I/O streams from the socket.
   - Send stream to server.
   - Receive response from server.
3. Close connections.

Java Networking: stream-based (contd.)

Practical demonstration

A practical will be done using NetBeans IDE. To do the following:

1. Demonstrate a simple communication between a client and a server.
2. Extend (1) above to mirror an obtainable real situation that can manipulate a Person java object.

Questions?

Let’s get hands-on!