




Application Program Development

Segment : Networking and Sockets

Presenter Name




Agenda for Week

- Lecture
 - Java Socket
 - Client and Server Paradigm
 - JDBC Pooling
 - JDBC Transaction System
 - Lab
 - Part 1 : In-Lab - Design and create a GUI based Application
 - Part 2 : DIY - using the GUI designed in-lab to write events
- 



Outcomes

- Understanding of Java Networking
 - Networking protocols
 - Java Sockets and their programming
 - Client and Server Paradigm
 - Multi-threaded Server
- 

Network

- A network is a system of computers connected together so they can share resources and communicate with each other.
- **Networking** refers to how the connected computers communicate.

Intranet



- Computers can also communicate across a private network called **Intranet**.
- Still common in today's businesses.
- Networking base comes from intranets.
- For examples, printers shared in a business environment over the business's intranet.
- Business can have multiple intranets one for each department.

Host

- You can also use networking when you want applications running on the same machine to communicate with each other.
- In networking a machine is usually referred as a **host**.



Client/ Server

- A common network configuration that one probably heard of is **client/server**, meaning that one (or more) hosts on the network are acting as servers, and the other hosts are clients that connect to the server.
- The browser is a client, and when you type in a web address, it connects to a server that has the files for the website address. In simple that's how internet works.

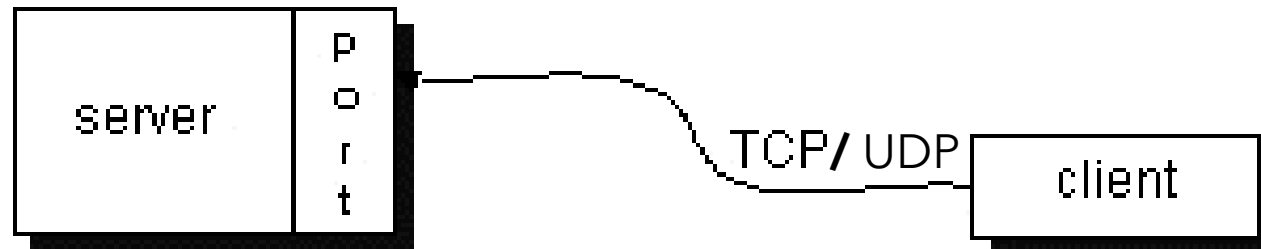
- 
- You can have client/server interaction on the same host, for example, the MySQL database comes with a workbench you can use to perform database operations. The **workbench** is a client, and it connects to the MySQL database server.
 - Quite often what web developers do when they are working on a website, is possibly to run an apache or IIS server on computer and connect to it using a web browser on the computer.
- 

Network Communications

- Computers on a network (including internet), communicates with each other using a **transport protocol**,
 1. TCP (Transmission Control Protocol)
 2. UDP (User Datagram Protocol)

- 
- Generally, a computer will have one **physical connection** to a network, and anything sent to the computer from other hosts on the network will arrive through that connection.
 - Sometimes the same computer will be running multiple applications that want data from the network. For example, at any one time, you might have browser open, a chat application open, streaming music etc.
- 

- Once data arrives at one of the physical connection to the network, how does it get to the targeted application?
- *Answer:* Ports



- Each application that needs data from the network is assigned a port.

TCP and UDP

- TCP (*Transmission Control Protocol*): is a connection-based protocol that provides a reliable flow of data between two computers.
- UDP (*User Datagram Protocol*): is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival. UDP is not connection-based like TCP.

TCP (Transmission Control Protocol)

- **Definition:** A connection based protocol, provides a reliable flow of data between two computers.
- Like making a Telephone call.
- Provides point-to-point channel for applications that require reliable communications.
- Like HTTP, FTP and telnet.
- The key of a successful transfer is in the order of data.
- **Problem:**
 - Prior to information transmission, a connection must be established.
 - Connection must be rigidly held during the whole communication.
 - The connection must be closed explicitly at the end.
 - Holding the bandwidth even if there is no communication happening.

Communication steps TCP

- When communicating TCP/IP, the sequence of events is as follows,
 1. The client opens a connection to the server.
 2. The client sends a request to the server.
 3. The server sends a response to the client.
 4. The client closes the connection to the server.
- Steps 2 and 3 may be repeated multiple times before the connection is closed.



UDP (User Datagram Protocol)



- **Definition:** is a non-connection based protocol that sends independent packets of data, called *datagrams*.
- No guarantee of data arrival.
- Like sending a letter through postal service. i.e. the order of delivery is not important and is not guaranteed as well each message is independent of each other.
- Example: clock server that sends current time to its client when requested to do so.

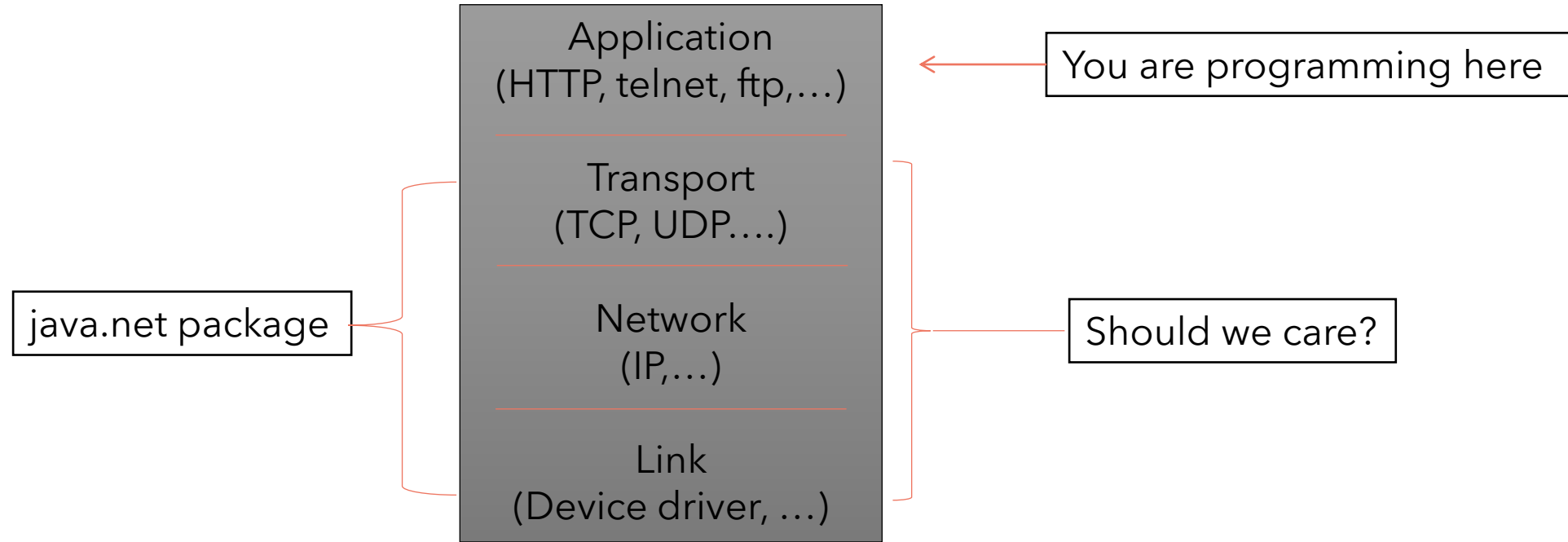
IPV4 and IPV6

- **IPv4:** Internet Protocol Version 4, uses 32-bit address scheme, allows over 4 billion unique addresses.
- Addresses four integers, separate by dots.
- Not enough in today's time, as we have computers, tablets, game consoles, smart TVs, smart phones, smart appliances etc, all connected to internet, and each of device needs a unique IP address.
- **IPv6:** Internet Protocol version 6, got birth. Uses 128-bit address scheme, lots and lots of unique IP addresses.
- Written in hexadecimal and separated by colons.

www.whatismyip.com

- 
- Data that is transmitted over the internet also carries information regarding which computer and port it will deliver to.
 - 32-bit IP address identifies which computer to deliver the message.
 - 16-bit number identify the Port, which TCP and UDP uses to deliver the data to the correct application.
- 

- 
- Port numbers range from 0 to 65,535 because ports are represented by 16-bit numbers.
 - **Well-known:** port numbers ranging from 0 - 1023 are restricted; reserved for use by well-known services such as HTTP and FTP and other system services.
 - Your applications should not attempt to bind to them.
- 



- Why do we need to know about TCP, UDP, IP, Ports, ... ?
- To make better decision about which Java Classes to use in the program.

Java Networking

- java.net package contains the classes you will use to establish connection between computers and then send messages between them.
- The package contains two sets of APIs,
 - Low-level API. (socket programming).
 - High-level API. (web-oriented, URL's, URI's)
- Java makes network coding much easier, by letting developers write code using abstract concepts and taking care of implementation detail under cover.

Sockets

- **Definition:** A *socket* is one endpoint of a two-way communication link between two programs running on the network.
- Java Socket programming can be connection-oriented (TCP) or connection-less (UDP).

Why Socket Programming?

- URL's and URLConnection's provide high-level mechanism for accessing resources on internet.
- Socket programming provides low-level network communication.
- Like writing *client-server* applications.

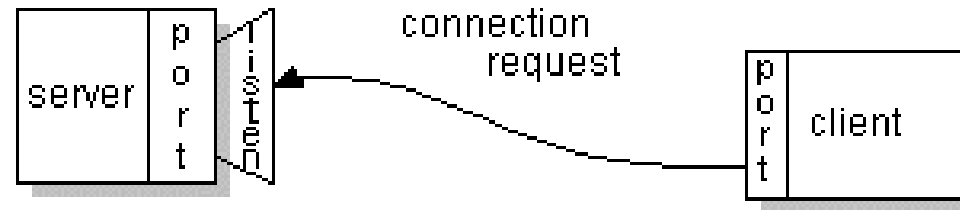
Client Server Application

- The server provides some service like performing database queries or sending out current stock prices.
- The client uses the service provided by the server.
- The communication that occurs between the client and the server must be reliable.
- No data can be dropped.
- Data must arrive on the client side in the same order in which the server sent it.



What is Socket?

- Server runs on a specific computer and has a socket that is bound to a specific port number.
- The server just waits, listening to the socket for a client to make a connection request.
- The client knows the hostname of the machine on which the server is running and the port number on which the server is listening.

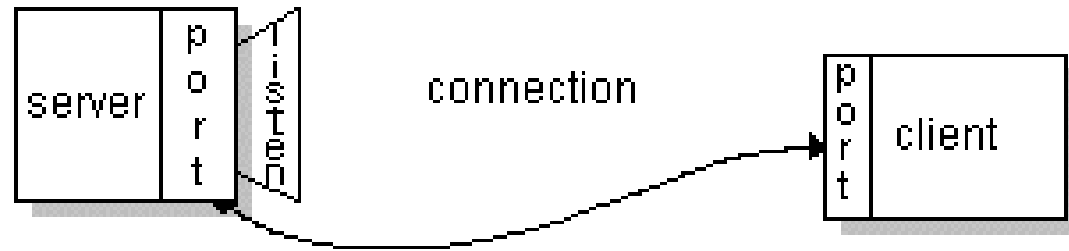
- To make connection,
 - Client exchange information with server.
 - Once client identification is done.
 - Local port is also assigned for communication during connection.



- If all goes well,
 - server accepts the connection.
 - server gets a new socket bound to the same local port.
 - its remote endpoint set to the address and port of the client

- 
- Why server needed new socket?
 - So that it can continue to listen to the original socket for connection requests.
- 

- On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.
- The client and server can now communicate by writing to or reading from their sockets.

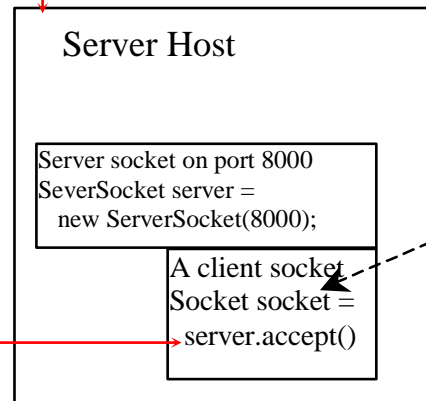


Client/Server Communications

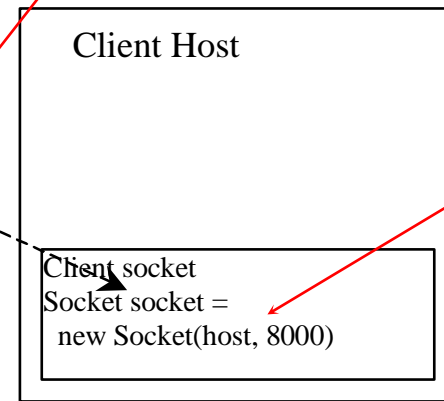
The server must be running when a client starts. The server waits for a connection request from a client. To establish a server, you need to create a server socket and attach it to a port, which is where the server listens for connections.

After the server accepts the connection, communication between server and client is conducted the same as for I/O streams.

After a server socket is created, the server can use this statement to listen for connections.



I/O Stream



The client issues this statement to request a connection to a server.

Step – 1: Creating a *ServerSocket*

- Establishing a simple server in Java requires five steps.

```
ServerSocket serverSocket = new ServerSocket(portNumber,  
queueLength)
```

- Registers an available TCP port number and specifies the maximum number of clients that can wait to connect to the server

Step – 2: Wait for a Connection

- Programs manage each client connection with a **Socket** object.
- Step 2 is now server listens for the connections indefinitely.
- To listen for a client connection, the program calls `ServerSocket` method **accept**

```
Socket socket = serverSocket.accept();
```

- This returns a `Socket` when a connection with a client established.

Step – 3: Get the Socket's I/O Streams

- Step 3 is to get the `OutputStream` and `InputStream` objects that enable the server to communicate with the client by sending and receiving bytes.
- The server sends information to the client via an `OutputStream` and receives information from the client via an `InputStream`.
- The server invokes method **`getOutputStream`** on the `Socket` to get a reference to the Socket's `OutputStream` and invokes method **`getInputStream`** on the `Socket` to get a reference to the Socket's `InputStream`.
- Often it's useful to send or receive values of primitive types (e.g., `int` and `double`) or `Serializable` objects (e.g., `Strings` or other serializable types) rather than sending bytes.

```
DataInputStream inputFromClient = new  
DataInputStream(socket.getInputStream());  
DataOutputStream outputToClient =  
new DataOutputStream(socket.getOutputStream());
```

Socket class

- A socket is simply an endpoint for communications between the machines.
- The Socket class can be used to create a socket.

```
public InputStream getInputStream()
```

returns the InputStream attached with this socket.

```
public OutputStream getOutputStream()
```

returns the OutputStream attached with this socket.

```
void close()
```

closes this socket

ServerSocket class

- The ServerSocket class can be used to create a server socket.
- This object is used to establish communication with the clients.

public Socket accept()	returns the socket and establish a connection between server and client.
void close()	closes the server socket.

The InetAddress Class

Occasionally, you would like to know who is connecting to the server. You can use the InetAddress class to find the client's host name and IP address. The InetAddress class models an IP address. You can use the statement shown below to create an instance of InetAddress for the client on a socket.

```
InetAddress inetAddress = socket.getInetAddress();
```

Next, you can display the client's host name and IP address, as follows:

```
System.out.println("Client's host name is " + inetAddress.getHostName());  
System.out.println("Client's IP Address is " + inetAddress.getHostAddress());
```

Establishing a Simple Client

Step 1 – Create Socket to connect to server

- First step is create a socket to connect to the server.

```
Socket connection = new Socket(serverAddress, portNumber);
```

- If the connection attempt is successful it will return a Socket.

Step 2 – Get the Socket's I/O Stream

- The client uses Socket methods `getInputStream` and `getOutputStream` to obtain references to the Socket's `InputStream` and `OutputStream`.
- If the server is sending information in the form of actual types, the client should receive the information in the same format. Thus, if the server sends values with an `ObjectOutputStream`, the client should read those values with an `ObjectInputStream`.

Step 3 – Perform the Processing

Step 3 is the processing phase in which the client and the server communicate via the `InputStream` and `OutputStream` objects.

Step 3 – Perform the Processing

- In *Step 4*, the client closes the connection when the transmission is complete by invoking the close method on the streams and on the Socket.
- The client must determine when the server is finished sending information so that it can call close to close the Socket connection.