Networking

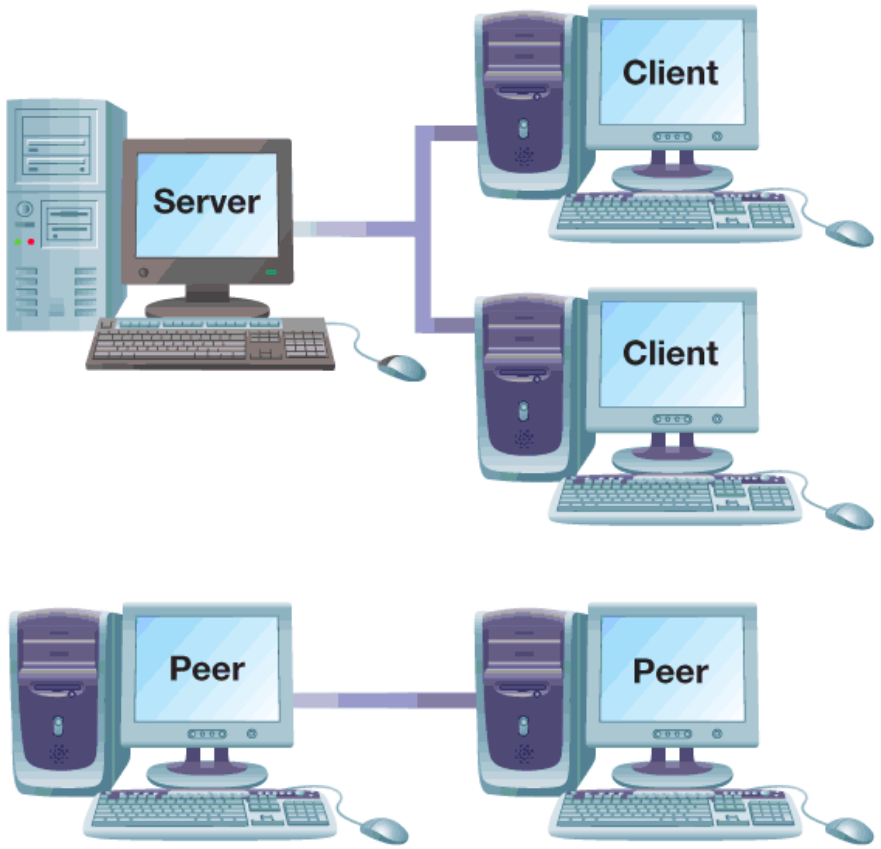
Agenda

- Networking Basics
- Transport Control Protocol/Internet Protocol (TCP/IP)
- IP Address
  - InetAddress class
- URL – Parsing, Reading, Connecting
  - URLConnection class
- Reading from a URL Connection
- Programming Using Sockets
  - Socket
  - SocketServer
- Multithreading Servers
- User Datagram Protocol (UDP)

Networking Basics-Definitions

- Network
  - A network consists of three separate components: server, client and peers
- Server
  - A server is any computer on the network that makes access to files, printing, communications, and other services available to users of the network. Servers only provide services
- Client
  - A client is any computer such as a user's workstation or PC on the network, or any software application such as a word processing application, that uses the services provided by server. Client only request services
- Peer
  - A peer is any computer that may both request and provide services.

Networking Basics-Client/Server

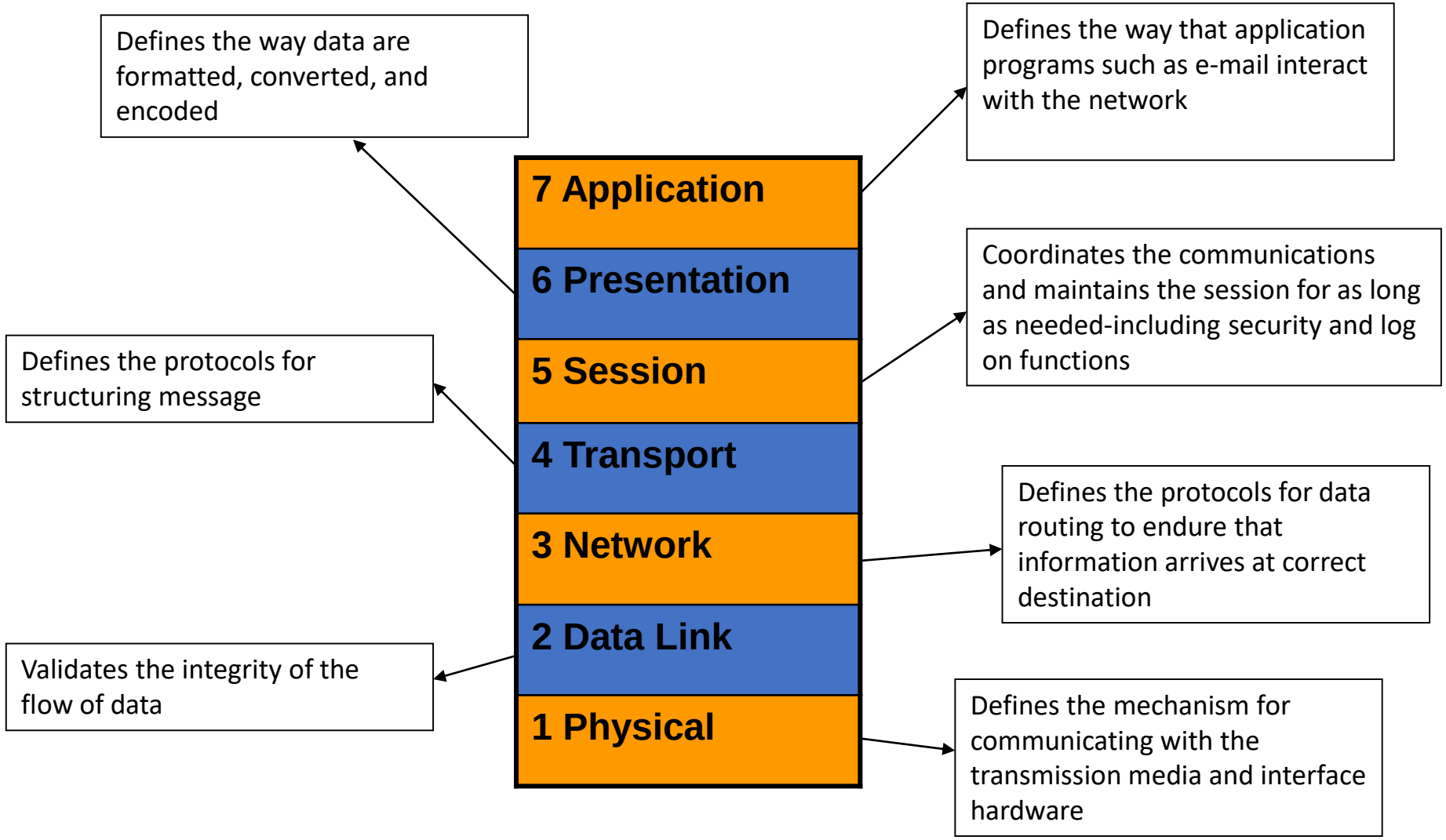


[Figure C.1 ➥ A server is a computer on the network that enables multiple computers (or "clients") to access data. A peer is a computer that may both request and provide services.]

Networking Basics - Layers

- Typical networks today have many layers of protocols, which generally start at the lowest, most basic level of the physical media (Ethernet, Token-Ring, dial-up, etc) and gradually add upper layers for functionality. Each layer communicates with the layer below it, gradually adding functionality.
- The International Organization for Standardization (ISO) defined a networking model called the Open Systems Interconnection (OSI) that divides computer-to-computer communications into seven layers.
- The OSI model is a protocol that represents a group of specific data tasks, represented with seven layers, which enables computers to communicate data.

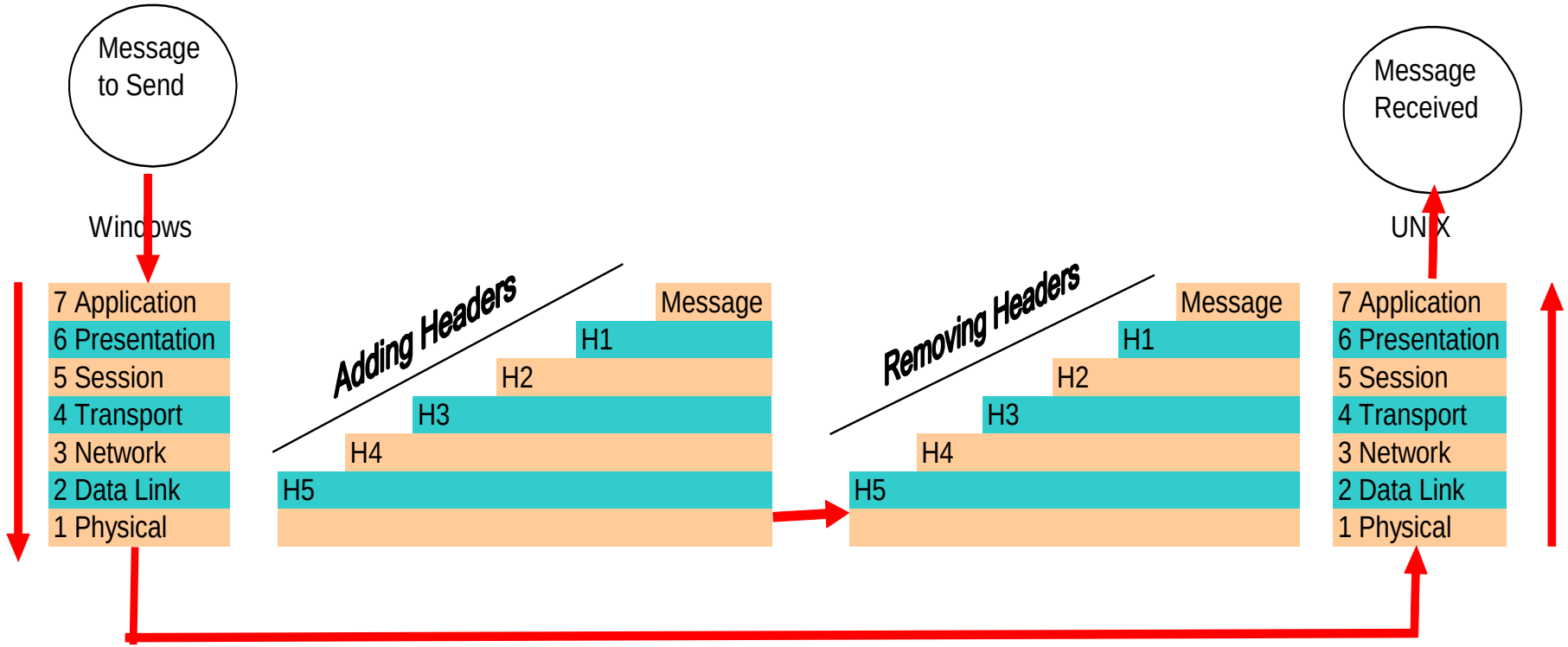
Networking Basics-Layers



Networking Basics-Message Passing

- When a message is transmitted from one computer to another one, assuming they use different operating systems, it is passed down from layer to layer in the operating system protocol environment of the first computer.
- At each layer, special bookkeeping information specific to the layer, called a header, is added to the data.
- Eventually, the data and headers are transferred from the first computers Layer 1 to second computers Layer 1 over some physical medium.
- Upon receipt, the message is passed up through the layers in the OS of the second computer.
- At each layer, the corresponding header information is stripped away, the requested task is performed, and the remaining data package is passed on until the message is arrives as it is sent.

Networking Basics Message Passing



Message passing between two different computers on a typical network

Networking Basics - IP

- Datagram
  - Data Layer is also called Datagram layer. It sits on top of physical layer. It receives data from the physical layer and passes it up to higher layers.
  - It simply packets the data for transmission and unpacks it for reception.

- IP layer
  - Transport layer is also called IP layer. It sits above datagram support. The IP layer adds addressing and routing information onto the data being sent.
  - This allows the packet to be routed and received by the intended computer
  - Two major protocols then sit on top of IP.
    - Transmission Control Protocol (TCP)
    - User Datagram Protocol (UDP)

Networking Basics

Application (HTTP, ftp, telnet, etc)
Transport (TCP, UDP, etc)
Network (IP, ethernet, etc)
Data Link (device drivers)

Networking Basics-TCP/UDP

- TCP vs. UDP
  - TCP guarantees the delivery of the packets but it takes extra network traffic and extra CPU cycles to keep TCP running
  - If necessary it retransmits the data if there were errors or if the delivery was not successful
  - UDP (User Datagram Protocol), on the other hand, is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival.
  - UDP is low-level and quite “raw”. It is basically a thin programmatic wrapper around its lower IP level. It doesn’t add much more functionality over IP nor consequently, overhead.
  - Using power of Java classes, we can use the upper TCP and UDP

Networking Basics – TCP/IP

- TCP/IP
  - Because so many different networks are interconnected nowadays, they must have a common language, or protocol, to communicate. The protocol of the internet is called TCP/IP. (Transmission Control Protocol/Internet Protocol)
  - The first part, TCP, breaks information into small chunks called data packets and manages the transfer of those packets from computer to computer
  - The IP defines how a data packet must be formed and where a router must forward each packet.
  - Together TCP and IP provide a reliable and efficient way to send data across different networks as well as the Internet.

Networking Basics

- TCP/IP Protocols
  - The Internet supports many protocols that sits on top of TCP/IP
  - By instantiating Java classes, we are able to use these protocols without understanding too much about the underlying TCP/IP packets or for that mater, the protocols themselves.
  - The following protocols uses TCP/IP to send and receive messages
  - Example: `SendMail`

Acronym	Protocol Name	Description
HTTP	HyperText Transport Protocol	World Wide Web
NNTP	Network News Transfer Protocol	News Groups
SMTP	Simple Mail Transfer Protocol	Sending electronic mail
POP3	Post Office Protocol 3	Reading electronic mail
FTP	File Transfer Protocol	Transferring files
TELNET	Terminal Emulation	Remotely controlling another computer

Networking Basics

- IP Addressing
  - IP stands for Internet Protocol and it is the protocol that moves packets of data between source and destination
  - Data transmitted over the internet is accompanied by addressing information that identifies the computer and the port for which it is destined
  - IP address is a unique four bytes (32 bit) number that identifies a computer in a network. Example: 142.204.1.22
  - Since these numbers are unique way to access a computer on a network the TCP/IP protocol takes care of the rest
  - Most IP addresses have a corresponding name known as domain name
    - Example: the domain name http://www.centennialcollege.ca/ has the IP address `199.212.60.207`
  - Although it seems plenty of available addresses, in reality, there is a shortage of addresses.
  - Example: `GetIP`

Networking Basics – Ports

- Ports
  - Ports are identified by a 16-bits (two bytes) number, which TCP and UDP use to deliver the data to the right application.
    - Example: 80 for HTTP
  - The port number is added to IP address and it is a unique place within the machine
  - The port is not a physical location in a machine, but a software abstraction
  - In fact, the port is a really just a number that both the server program and the client program agree to use as a channel between them
  - The reason why we don't see port numbers when we specify addressed on the web is because there are default port numbers assigned to various services
  - Example: `LookForPorts`

Port Number	Service	Port Number	Service
20	FTP Data	80	HTTP
21	FTP Control	110	POP3
23	TELNET	119	NNTP
53	DNS		

Networking Basics

- Universal Resource Locator (URL)
  - URLs point to resource files on the internet. The files may be graphics, text, HTML, video, or any kind of data. The URL specifies the file name, location, as well as which protocol to use to retrieve that file.
  - The URL is actually made up of several parts
    - The protocol to use
    - The internet address
    - The port number (optional)
    - The directory path to the resource
  - Example: http://db2.centennialcollege.ca/ce/coursedetail.php?CourseCode=COMP-424
  - Example: URLConnectionTest
- Domain Name Service (DNS)
  - Address numbers are hard to remember
  - A service known as DNS allows us to identify an address of a machine, using a name
  - The name service translates this name into a number, then the number is used to make the connection
  - Example: `DNSLookup`

Java Networking Classes – java.net

DatagramPacket	This class represents a datagram packet.
DatagramSocket	This class represents a socket for sending and receiving datagram packets.
DatagramSocketImpl	Abstract datagram and multicast socket implementation base class.
HttpURLConnection	A URLConnection with support for HTTP-specific features.
InetAddress	This class represents an Internet Protocol (IP) address.
InetSocketAddress	This class implements an IP Socket Address (IP address + port number) It can also be a pair (hostname + port number), in which case an attempt will be made to resolve the hostname.
MulticastSocket	The multicast datagram socket class is useful for sending and receiving IP multicast packets.
PasswordAuthentication	The class PasswordAuthentication is a data holder that is used by Authenticator.
ServerSocket	This class implements server sockets.
Socket	This class implements client sockets (also called just "sockets").
SocketAddress	This class represents a Socket Address with no protocol attachment.
SocketImpl	The abstract class SocketImpl is a common superclass of all classes that actually implement sockets.
URI	Represents a Uniform Resource Identifier (URI) reference.
URL	Class URL represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web.
URLClassLoader	This class loader is used to load classes and resources from a search path of URLs referring to both JAR files and directories.
URLConnection	The abstract class URLConnection is the superclass of all classes that represent a communications link between the application and a URL.
URLDecoder	Utility class for HTML form decoding.
URLEncoder	Utility class for HTML form encoding.

InetAddress class

- This class represents an internet IP address
- getLocalHost determines the IP address of the local host and returns a matching InetAddress object
  - public static InetAddress getLocalHost()
- getByName determines the IP address of the host name specified and returns a matching InetAddress object
  - public static InetAddress getByName(String host)
- getHostName returns the host name for this IP address
  - String getHostName()
- getAddress() returns the raw IP address of InetAddress object.
  - byte[] getAddress()
- getHostAddress() works like getAddress, except getHostAddress returns a string representing the numeric IP address of the InetAddress object
  - String getAddress()
- Example: `GetIP`

URL Class

- This class allows us to create a URL in one of several ways
- The first version of the constructor allows us to specify all of the pieces of the URL separately
  - URL(String protocol, String host, int port, String file)
  - Note: if port equals -1, then the default port is used
- The second version of the constructor is the same as the first version with a -1 for the port number implied
  - URL(String protocol, String host, String file)
- The third constructor accepts the standard string representation of a URL
  - URL(String spec)
- Parsing a URL with get Methods
  - getPort(), getProtocol(), getHost(), getFile()
- Example: `ParseURL`

URLConnection class

- This class allows us to make a connection to the URL specified
- openConnection() method returns a URLConnection object
  - URLConnection openConnection() throws IOException
- After connecting to the URL, we can read data from the URL using getInputStream() method
- Example: URLConnectionReader

Reading from a URLConnection

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

public class URLConnectionReader {
  public static void main(String[] args) throws Exception {
        URL seneca = new URL("http://www.senecac.on.ca/");
        URLConnection sc = seneca.openConnection();
        BufferedReader in = new BufferedReader(
                                new InputStreamReader(
                                sc.getInputStream()));
        String inputLine;
        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
        in.close();
  }
}
```

Debugging Servers

- Telnet
  - One easy way to debug your servers that send ASCII data is to test them with telnet program
  - Telnet provides a way to send data to your server
  - Format
    - telnet [IP or DNS] [port]
  - Example
    - telnet localhost 1234
- Anything you type in the telnet console gets sent to the server. Anything the server sends is displayed on the telnet screen

Programming Using Sockets

- Server and Clients
  - The whole point of a network is to allow two machines to connect and talk to each other
  - Once the two machines have found each other they can have nice, two way conversation
  - The machine that “stays in one place” is called a server, and the one seeks is called a client.
- Server’s and Client’s Job
  - The job of a server is to listen for a connection, that’s performed by the special server object that we create
  - The job of a client is to try to make a connection to a server, and this is performed by the special client object that we create
  - Once the connection is made, you’ll see that at both server and client ends, the connection turns into an IO stream object, and from then on we can treat the connection as if we were reading from and writing to a file

Programming Using Sockets

- Sockets
  - The socket is the software abstraction need to represent the "terminals" of a connection between two machines
  - When we connect to an IP ***address:port***, we automatically get a socket on the server side. There is also a socket on the client side
  - For a given connection, there is a socket on each machine, and you can imagine a "hypothetical cable" running between two machines with each end of the "cable" plugged into a socket
  - Of course, the physical hardware and cabling between machines is completely unknown
  - You can create multiple sockets between the client and server
  - To create a connection, both the client and server have to create a socket
    - ***Socket*** class is used to create a socket on the client side
    - ***ServerSocket*** class is used to create a socket on the server side

Programming Using Sockets

- Sockets in Java
  - In Java, you create a socket to make the connection to the other machine, then you get and InputStream (or with the appropriate converters, Reader and Writer) from the socket in order to be able to treat the connection as an IO stream object.
  - Once a client makes a socket connection, the ServerSocket returns (via ***accept()*** method) a corresponding server side Socket through which direct communications will take place. From then on, you have a true Socket to Socket connection and you treat both ends the same way because they are the same.
  - At this point, you use the methods getInputStream() and getOutputStream() objects from each Socket. These must be wrapped inside buffers and formatting classes just like any other stream object
  - When you create a ServerSocket, you only give a port number. No IP number is needed because it is already on the machine it represents
  - When you create Socket, however, you must give both IP and port where you are trying to connect

Steps for Implementing a Server

1. Create a ServerSocket object

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

2. Create a Socket object from ServerSocket

```
while(someCondition) {
  Socket server = listenSocket.accept();
  doSomethingWith(server);
}
```

   – Note that it is quite common to have doSomethingWith spin off a separate thread

3. Create an input stream to read client input

```
BufferedReader in = new BufferedReader (new
  InputStreamReader(server.getInputStream()));
```

Steps for Implementing a Server

4. Create an output stream that can be used to send info back to the client.

```
// Last arg of true means autoflush stream
// when println is called
PrintWriter out = new
  PrintWriter(server.getOutputStream(), true)
```

5. Do I/O with input and output Streams
   - Most common input: readLine
   - Most common output: println
   - Again you can use ObjectInputStream and ObjectOutputStream for Java-to-Java communication
6. Close the socket when done

```
server.close();
```

   - This closes the associated input and output streams.

Example: `NetworkServerTest`

Steps for Implementing a Client

1. Create a Socket object

```
Socket client = new Socket("hostname",portNumber);
```

2. Create an output stream that can be used to send info to the Socket

```
// Last arg of true means autoflush – flush stream
// when println is called
PrintWriter out = new
  PrintWriter(client.getOutputStream(), true);
```

3. Create an input stream to read the response from the server

```
BufferedReader in = new BufferedReader(new
  InputStreamReader(client.getInputStream()));
```

Steps for Implementing a Client

4. Do I/O with the input and output Streams
   - For the output stream, PrintWriter, use print and println, similar to System.out.println
     - The main difference is that you can create PrintWriters for different Unicode characters sets, and you can't with PrintStream (the class of System.out).
   - For the input stream, BufferedReader, you can call read to get a single character or an array of characters, or call readLine to get a whole line
     - Note that readLine returns null if the connection was terminated (i.e. on EOF), but waits otherwise
   - You can use ObjectInputStream and ObjectOutputStream for Java-to-Java communication. Very powerful and simple.
5. Close the socket when done

```
client.close();
```

   - Also closes the associated input and output streams

Example: `NetworkClientTest`

Multithreading Servers

- The previous server only supports one client at a time. In reality, we would want server to support many clients. That's why we need Java's multi-threading capabilities.
- A separate thread is used for each client
- Benefits of multi-threaded servers
  - It makes the server scalable. The server can accept new requests independent of its speed in handling them
  - By handling each request in a new thread, clients do not have to wait for every request ahead of them to be served
  - The program source can be better organized, as the server processing is written in different class
- Example: `MyMultiServer`

User Datagram Protocol (UDP)

- In addition to TCP sockets, Java also supports a second transport mechanism known as UDP or User Datagram Protocol.
- UDP operates at a lower level of functionality than TCP. TCP takes care of things such as:
  - Verifying if the message received was intact
  - Verifying if a message received was indented for the machine that received it
  - Verifying if the transmitted data successfully reached an intended destination
  - Verifying that the order of reception matches the order of transmission
- UDP does not do any of these things
- UDP is connectionless
  - A client come and go, so can a server. There is no “lost connection”. Multiple packets sent from one machine to another might be routed differently, and might arrive in any order
- UDP is unidirectional
  - The data always travels from server to client

- Examples: `UDPSend and UDPReceive`

Additional Reading and Other Resources

- ***Networking Tutorial by Oracle***
  - ***https://docs.oracle.com/javase/tutorial/networking/***
- ***Sockets Programming in Java: A Tutorial by Quasay H. Mahmoud, JavaWorld.com***
  - ***http://www.javaworld.com/article/2077322/core-java/core-java-sockets-programming-in-java-a-tutorial.html***
- ***Just Java 2 by Peter van der Linden***
  - ***Chapter 17. Networking in Java***