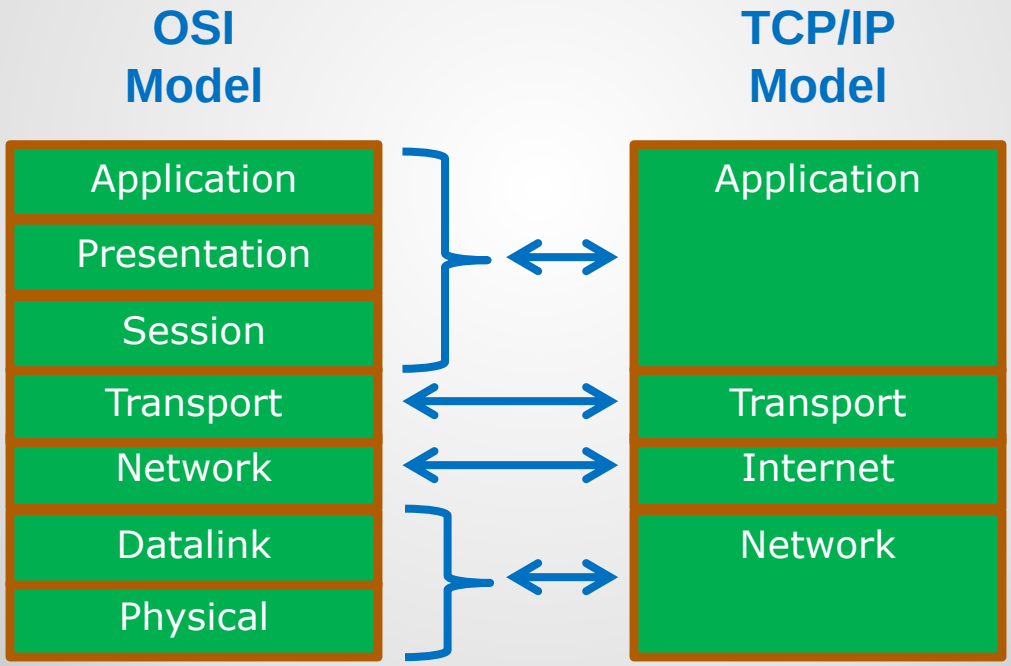
Java Programming

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

Today's Lecture

- Networking Programming

- A networking model conceptually describes how computer systems can communicate with each other.
- Open System Interconnect (OSI) and TCP/IP are two types of models.
- TCP/IP Model is simpler than OSI. It was actually developed after the TCP/IP protocol suite (protocol gives more specifics than a model).



OSI Model vs TCP/IP Model

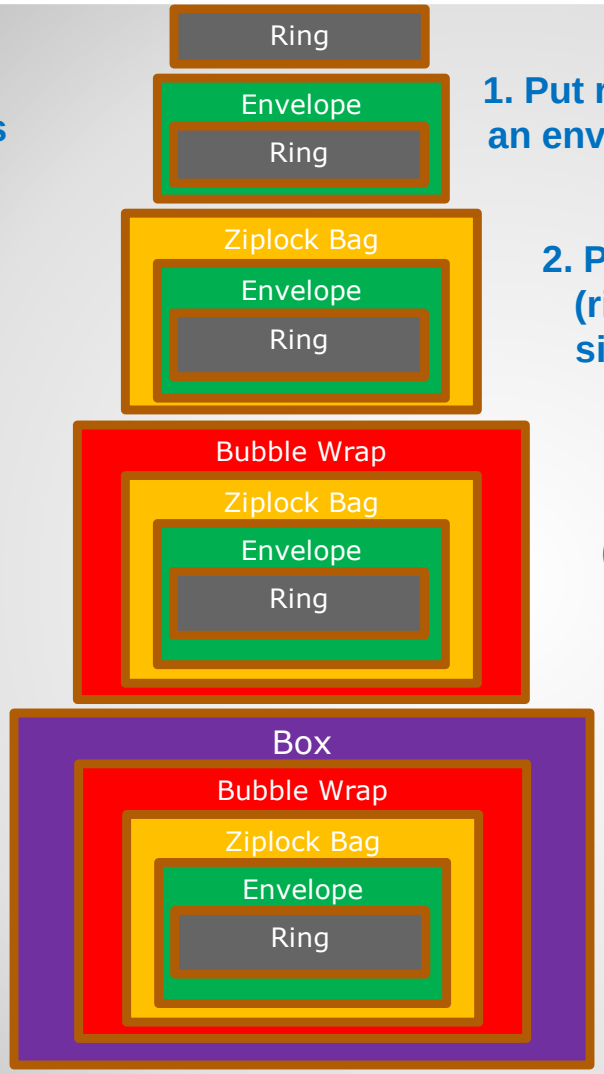
- Some companies use a certain set of packaging when they mail something to a customer.
- For example, you might want to mail a ring using a certain setup of packaging.

- The ring goes in an envelope.
- The envelope goes in a ziplock bag.
- The ziplock bag goes in bubble wrap.
- The bubble wrap goes in a box.

- For example…

Using a Protocol to Mail Something

Keep adding layers of packaging



1. Put ring in an envelope

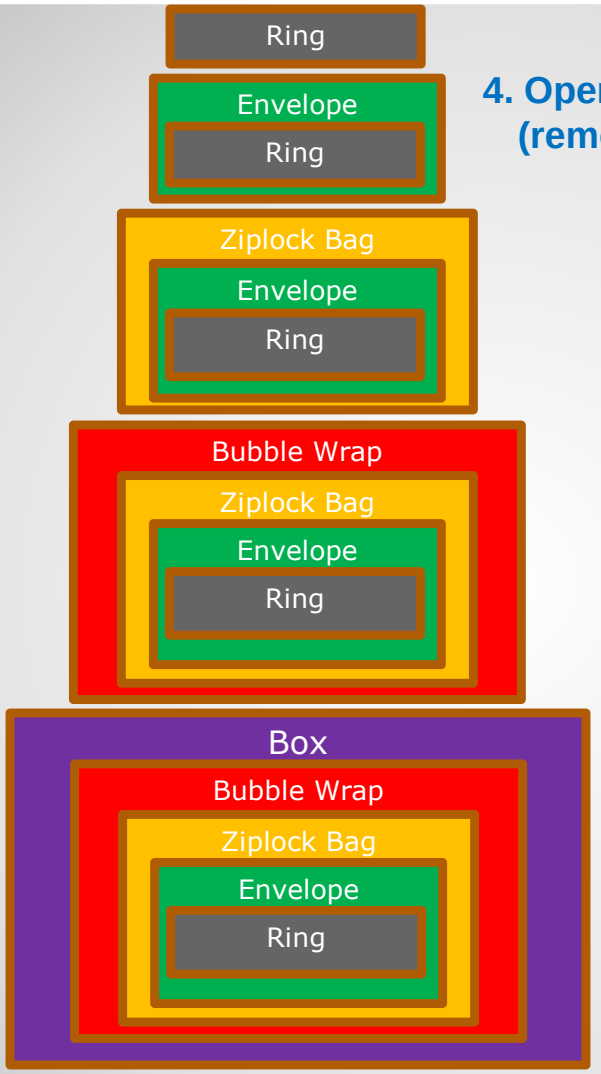
2. Put envelope in ziplock bag (ring is also in ziplock bag since it is in the envelope)

3. Put ziplock bag in bubble wrap (this means the envelope and ring are also in the bubble wrap)

4. Put bubble wrap in box (everything in the bubble wrap is in the box too)

The box gets sent

Mailing a Ring Using a Protocol



4. Open envelope (remove ring)

3. Open ziplock bag (remove envelope)

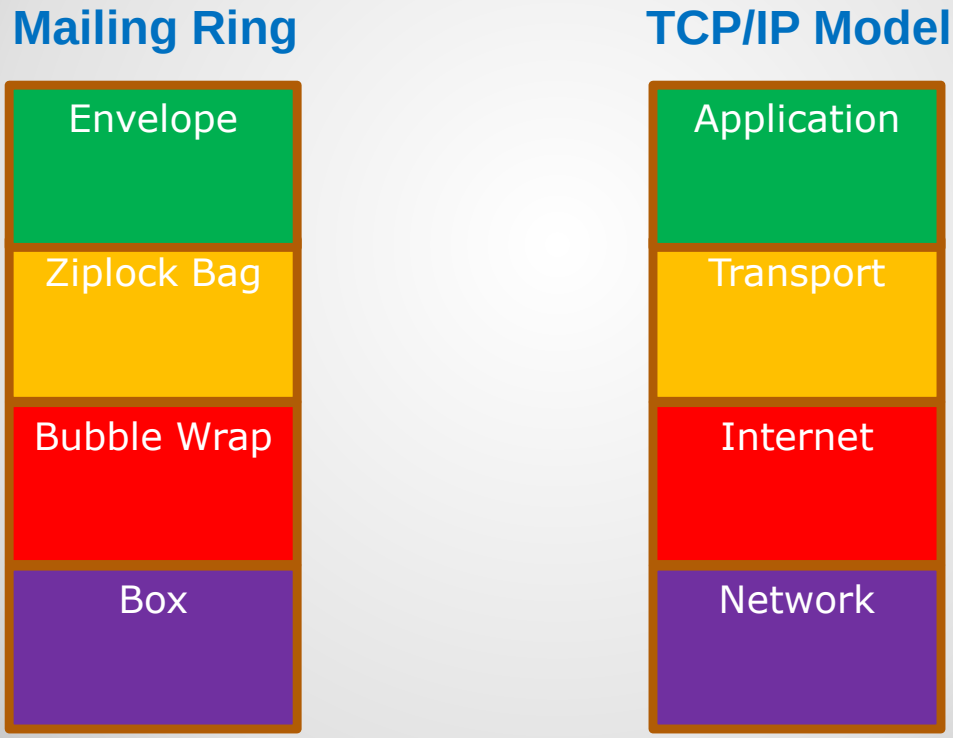
2. Open bubble wrap (remove ziplock bag)

1. Open box (remove bubble wrap)

Keep unpacking layers

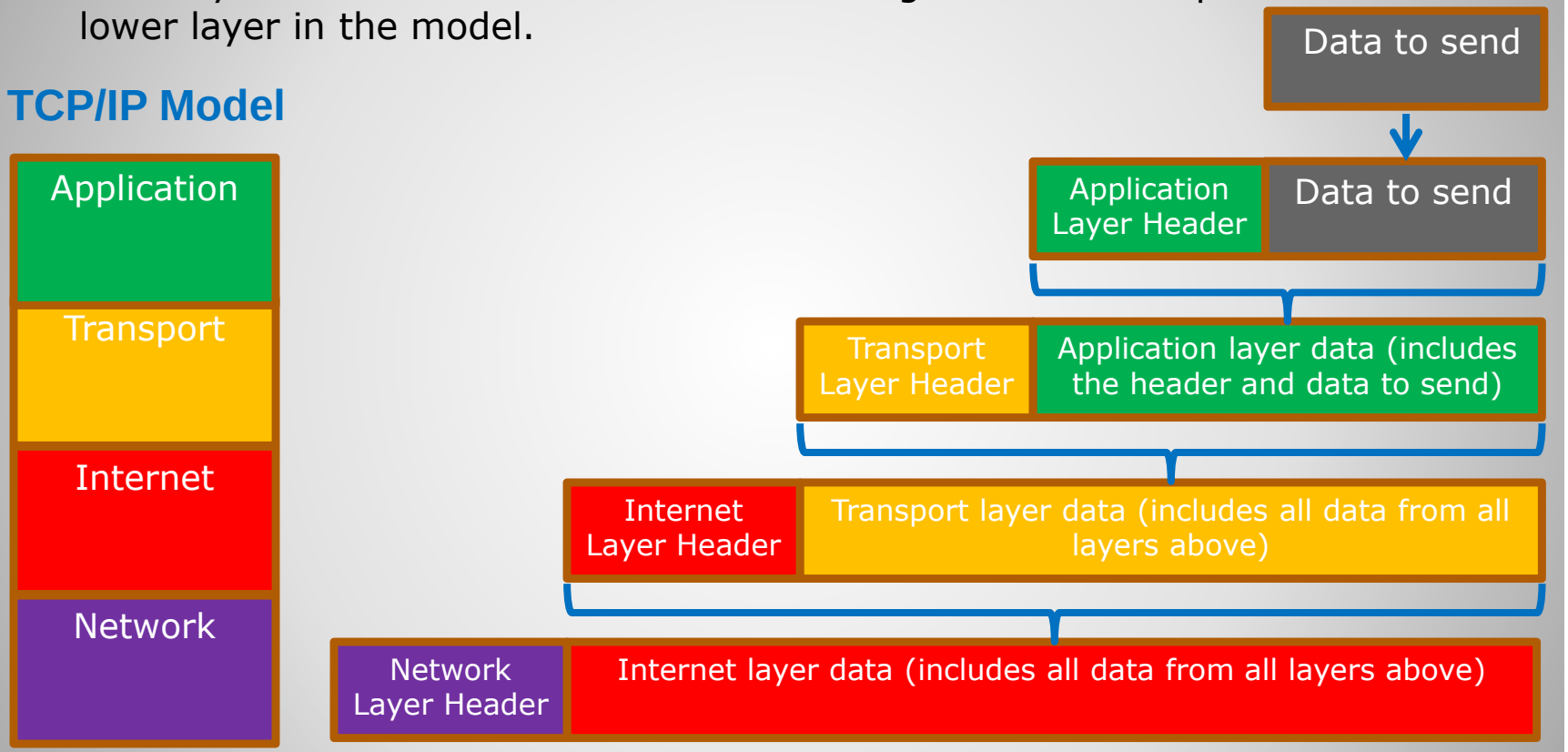
Receiving a Ring Using a Protocol

- The TCP/IP model works similarly to how the ring was sent and received.
- When sending, data at a higher level is "packaged" and given to the next lowest layer.
- When receiving, data at a lower level is "unpacked" and sent up one layer.



TCP/IP Layering and Sending Data

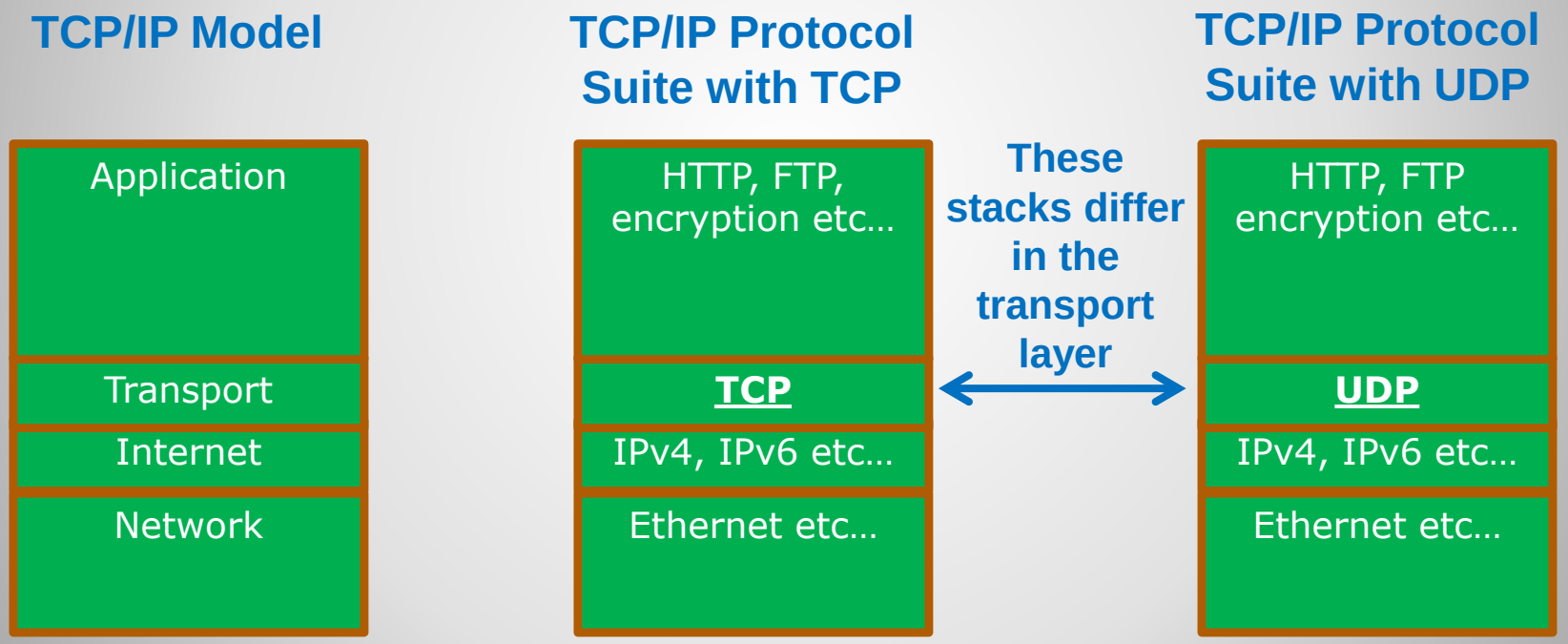
- Data to send is given to the application layer.
- The application layer adds a header and passes it to the transport layer.
- Each layer in the model adds its header and gives the whole piece to the next lower layer in the model.



TCP/IP Layering and Sending Data

TCP/IP Protocol Suites

- The TCP/IP model is a conceptual description.
- The TCP/IP protocol suites are more specific.



TCP/IP Protocol Suites

Transmissions Control Protocol (TCP)

- Connection-based
- Has connection overhead
  - Must open a connection
  - Must make sure messages were correctly received
  - Must terminate a connection
- Allows data to be sent in two directions over the connection.
- Has built-in error checking to make sure that data was sent correctly.
- Resends data if there was an error in transmission.

TCP

Universal Datagram Protocol (UDP)

- Connectionless
- Less overhead
  - Does not make connections
  - Does not make sure message was correctly received
  - Does not terminate connections
- Data is sent in only one direction.
- Has small amount of error checking.
- Does not resend data.
- Good for real time communications. For example, send out a message giving the current time (no need to resend this type of message because the data will be "stale" if resent). Also, good for live streaming video, real-time online gaming (losing a little data won't really hurt and UDP is faster than TCP).

UDP

- Now we will cover sockets…

Sockets

Socket

- Endpoint of communication.
- Makes network programming similar to file I/O.
- You can read/write to a socket in a similar way that you read/write to a file.
- Socket types
  - Stream
  - Datagram

Socket

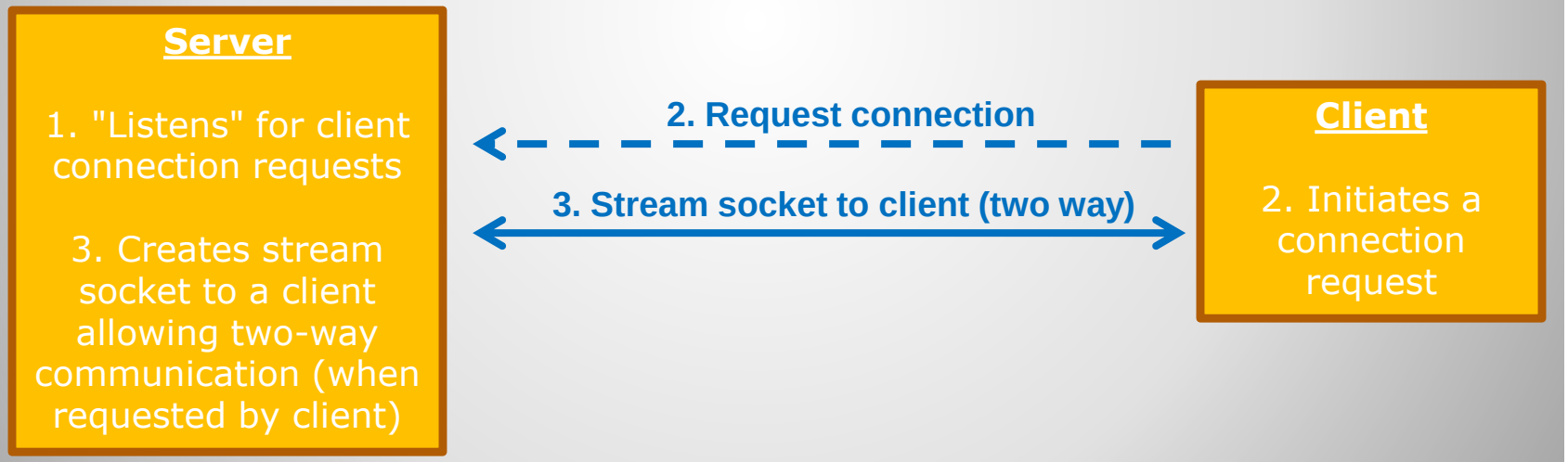
Stream vs Datagram Sockets

- **Stream sockets**
  - Based on TCP
  - Establishes a connection
  - Allows two-way communication
  - More reliable
  - Slower than datagram
- **Datagram sockets**
  - Based on UDP
  - Connectionless
  - Only one way communication allowed
  - Less reliable
  - Faster than stream

- The following slides detail a basic stream socket client/server setup…

Basic Stream Sockets

- Stream sockets are connection based.
- With stream sockets there is a client and server.
- The server "listens" for connection requests from clients (other computers).
- When a client requests a connection, the server generates a new socket connected to that client.



Stream Sockets - Connections

Server Socket Listening

- A server socket must be created to "listen" for client connections.
- The ServerSocket class is used to receive client connection requests.
- You must call the accept() method on the server socket to actually "listen" for incoming client requests.
- accept() blocks (does not move on to the next instruction) until a client connection request is made.
- accept() will create a Socket instance for the client when it receives a connection request.

Create a server socket on port 55555 (call to new requires a try/catch, left out to save space on slide)

```
ServerSocket servSocket = new ServerSocket(55555);

Socket clientSocket = servSocket.accept();
```

accept() makes server socket "listen" for client connections

accept() returns a socket instance connected to the client

Stream Sockets – Server Listening

Socket Client Connect

- A client application will request a connection from the server.
- The client application must know the server's IP address and the port that the server is listening on.

```
String hostIP = "127.0.0.1";
int hostPort = 55555;

Socket socket = new Socket(hostIP, hostPort);
```

127.0.0.1 is the local host address. A real application would change this to the address of the server machine.

The Socket instance returned by new is connected and ready to use (new will throw an exception if it fails, needs to be in a try/catch)

The call to new requests a connection to the server (the constructor will do the connecting)

Stream Sockets – Client Connect

Receive Data Using Socket

- Both the client and server use the same code to receive data.
- Get the input stream from the socket (only need to do once).
- Read data from the input stream.
- Important – Reading from the socket returns null if the other socket it is connected to closes.

Get input stream from the socket

```
InputStream input = socket.getInputStream();
BufferedReader reader = new BufferedReader(new InputStreamReader(input));

String data;
data = reader.readLine();
```

Wrap input stream in a BufferedReader to allow for reading lines of text

Reads data from the input stream (data will be a null reference if the other socket closed)

Stream Sockets – Receive Data

Send Data Using Socket

- Both the client and server use the same code to send data.
- First get the output stream from the client socket.
- Next, write data to the output stream.

Get output stream from the socket

```
OutputStream output = socket.getOutputStream();
PrintWriter writer = new PrintWriter(output, true);

writer.println("This is data");
```

Write data to the output stream (this is writing to the socket)

Wrap the output stream in a PrintWriter to allow for writing formatted output (not just byte data)

Stream Sockets – Send Data

Close Sockets and Streams

- When you are done with a socket close the following:
  - Socket input stream
  - Socket output stream
  - The socket itself (do this last)

Stream Sockets – Close

Server Connections

- The server thread as previously described could only handle one client.
- If we use multiple threads in the server, we can allow the server to handle multiple clients.
- The server should spawn a new thread for each client connection so it can handle interactions between each client separately.

Multithreading and Sockets – Server Connections

Receiving Data

- An application can create a thread that is dedicated to receiving messages from a socket.
- The main thread will no longer need to block when receiving messages on the socket.
- Use an executor for the threads so the threads can be shut down if the application closes.
- Message Thread Writing to GUI.
  - Worker Threads do not have access to GUI controls on the main thread.
  - If the thread receiving messages must update the GUI special code needs to be added.
  - For example, in JavaFX you will need to use Platform.runLater(…) to update controls in the GUI thread.

Multithreading and Sockets – Receiving Data

- End of slides

End of Slides