# Computer Programming – CS 6011
# Lecture 4: Network Programming

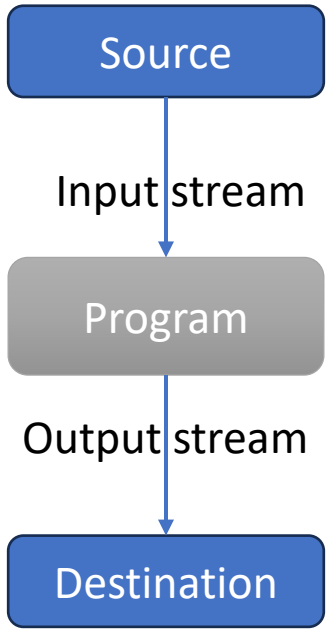
Fall 2023

# Topics

- Network Programming (HTTP Web Server)
  - Streams
  - Sockets
  - Implementing Server Side
  - Implementing Client Side
  - HTTP headers

# Streams

- Java uses *streams* and *adapters* for I/O.
  - Input / Output
- For example:
  - Input Streams
    - System.in (Stream connected to the console)
    - FileInputStream (Stream connected to a file)
    - ByteArrayInputStream (Stream connected to a byte[])
  - Output Streams
    - System.out
    - FileOutputStream
    - ByteArrayOutputStream



# Adapters

- `I/O Streams` read or write data
  - So put the stream into a wrapper object (an adapter) that does more for us.

- `Adapters`: Higher level objects that use streams to provide higher level operations
  - Input: `Scanner` adapter object reads multiple bytes and parses / returns numeric values, strings, etc.
    - Wraps an input stream
      - `Scanner sc = new Scanner( fileInputStream );`
  - Output: `PrintWriter` adapter object takes in variables, and sends their values into the associated output stream.
    - Wraps an output stream
      - `PrintWriter pw = new PrintWriter( fileOutputStream );`

# More Adapters

- ObjectInputStream (reads in an “Object”)
- ObjectOutputStream (write objects – so you can send them to a file)
- DataInputStream
- DataOutputStream

# How the scanner works?

- What do I mean by “Take an `InputStream` and shove it into a `Scanner`”…

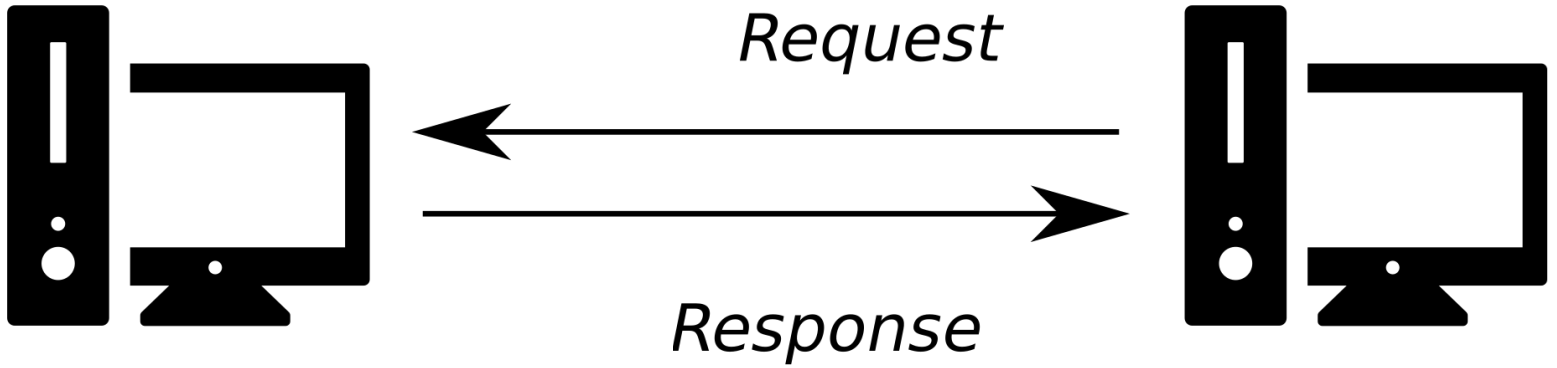
```
FileInputStream inputStream = new FileInputStream("test.txt");
Scanner myScanner = new Scanner( inputStream );
myScanner.useDelimiter("A");
String result = myScanner.next();
```

# Client/Server Implementation

**Idea of stream connected to the network**

- Socket I/O stream instead of I/O stream

# Client/Server communication




Server

- Server Socket
- Listen
- A client connects and there would be a socket
- Communication via **I/O streams**

Client (Web browser)

- Socket
- Connect
- Communication via **I/O streams**
- Close

# What do we want on the **Server** side?

- Server waits for client connections – on a specific port Number
- Create a **Server Socket** and wait for the Client requests – the constructor takes a port Number
- Use sockets I/O streams to perform communication with Client
- Close sockets

# What do we want on the **Client** side?

- Client connects to a server – that needs an IP and port
- Create a **Socket** Object – the constructor takes an IP address and a port Number
- Use sockets I/O streams to perform communication with the server
- Close the socket when done

# Java Network Programming

- Important Classes
  - `Socket` – represents an open connection
  - `ServerSocket` – a server socket that listens for connections on a given port.
  - The `ServerSocket` doesn't actually contain a socket, but it creates/returns one when a client connects to it.

# Implementing a Server

- Create a Server Socket:

```
ServerSocket server = new ServerSocket(Port#);
```

- Wait for the Client Request:

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

- Create I/O streams for communicating to the client

```
Scanner textFromSocket = new Scanner( client.getInputStream() );
PrintWriter textToSocket = new PrintWriter( client.getOutputStream() );
```

- Perform communication with client

  Receive from client: `String str = textFromSocket.next();`

  Send to client: `textToSocket.print(”Sending text to client\n");`

- Close sockets:

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

# Implementing a client

- Create a Socket Object:

```
Socket client = new Socket( serverIP, PORT# );
```

- Create I/O streams to communicate with the server.

```
Scanner textFromSocket = new Scanner(client.getInputStream() );
PrintWriter textToSocket = new PrintWriter( client.getOutputStream() );
```

- Perform communication with server

  Receive from server: `String str = textFromSocket.next();`

  Send to the server: `textToSocket.print(”Sending text to the server\n");`

- Close sockets:

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

# Lifecycle of a `ServerSocket`

- A `ServerSocket` waits around until a client tries to connect.
- To tell the `ServerSocket` to wait for a connection, we call its *accept()* method.
- *`.accept()`* **blocks** the server code until a client connects.
  - When a client connects, `accept()` returns a Socket that is connected to that client.
  - At this point we can read/write to that socket to communicate with the client.
- When we are done using a socket, we must *`close()`* it to disconnect.
- A single ServerSocket can `accept()` many clients simultaneously (which we'll do later).
- When we are done listening for clients, we should `close()` the `ServerSocket` and end the program.
- Remember, while getting / using sockets looks very similar on both the client and the server, the code must be implemented separately for each.

# Socket I/O

- Once we have an open (connected) socket, we can read/write to it by calling the `getInputStream()` or `getOutputStream()` method.
- We can, for example, connect the `InputStream` to a `Scanner`.
- And connect the `OutputStream` to a `PrintWriter`.
- We use the `OutputStream's` **`flush()`** method to send any output data immediately.
  - Output data is usually buffered until the OS decides there is enough data to send.

# What is a Web/HTTP Server?

# What is a Web/HTTP Server?

- Request/Response model
- Different components involved:
  - **HTTP(s)** (HyperText Transfer Protocol) that defines the format between the browser and the server
  - **Web Server** waiting for client requests
  - **Web content / File(s)** that will be sent to the client
  - **Additional information (HTTP header)** on the top of the files sent

- An http server copies files to socket output streams

# HTTP Header Example

- curl msd.Utah.edu example
- Server responses:
  - `HTTP/1.x 200 OK`
  - `Date: Sep 25, 2023`
  - `Content-Type: text/html`
  - `Content-Length: 1024`
  - Blank Line
- Note:
  - End of each header line (including the blank line) should be `\r\n`
    - Not just `\n`

# HTTP Headers

- What data is the server receiving? What information is it sending to the client? How does the server know / specify this?
- Some references:

  https://www.tutorialspoint.com/http/http_responses.htm

  https://code.tutsplus.com/tutorials/http-headers-for-dummies--net-8039
- What headers do you see?
  - Messages from client to server.
  - Messages from server to client.
- Client request header:
  - `GET / HTTP/1.1`
  - `Host: localhost`
  - How does the server know when the client's request header is done?
    - Blank line
- `curl` example

# Basic HTTP Server

- Today's assignment is writing a basic HTTP Server.
- You will NOT write code for a client, just the server.
- What will we use for a client?
  - Safari (or any other web browser)
  - `curl` (sends less "stuff" to the server – so a good way to start). `curl -v` for verbose…
  - Connect to: `127.0.0.1` (this is always your machine.)
    - Or `localhost` Or `10.132.25.16` (Whatever your IP address is – might change)
- What port to use?
  - Normal HTTP web servers use port 80 – you may use it
  - Better to use 8080 so the URL to connect to is: `localhost:8080`

# Dealing with error for the web server

- When something breaks…
  - File Not Found
  - Network connection dies
  - etc…
- For today, just use IntelliJ to add the Exception to the function signature
  - Looks like: `throws IOException`

```
public static void main( String[] args ) throws IOException
```

# Server Pseudocode

- Sketching the outline of a server…

```
   ServerSocket server;
// Until the server is killed, or some other method is used to tell it to stop
while( true ) {
        socketToClient <- server.accept()
        PrintWriter out <- socketToClient.getOutputStream()
        Scanner in <- socketToClient.getInputStream()
        // Read all the information that the client has sent us to determine what it wants
        // Parse the HTTP request simple text header
        // Send data back to the client
        // Create a HTTP response header text
        // For example, open the requested file (index.html) and send it to the client.
        // Start over listening for the next connection… (accept() above)
}
```

# Thursday Assignment(s)

- Code Review – Rainfall
- Assignment – Basic HTTP Server