CSCI-142

Computer Science II

Networking

Activity: Getting Started

1. Begin by accepting the GitHub Classroom invitation for today's homework.
   a. *The project **may** already contain some code!*
2. Create a `session` package. This is where you will write your solutions to today's activities.
3. Create a `homework` package. This is where you will implement your solution to the homework.

When you submit your homework, you will include your activities. You may earn up to a 10% bonus on the homework if you have completed all of the activities.

Do not submit code that ***does not compile***. Comment it out if necessary.

This Week

WEEK 15	SUN	MON	TUES	WEDS	THURS	FRI	SAT
QUIZ			Quiz #21		Quiz #22		
LECTURE			Networking		Final Exam Review		
HOMEWORK	Hwk 18 Due (**11:30PM**)		*Hwk 22 Assigned*			Hwk 22 Due (**11:30pm**)	

The Final Exam

- 12/17 - 10:45am - 1:15pm
- *75 Minute Written Exam*
  - Comprehensive
- *75 Minute Practical Exam*
  - Exceptions & IO
  - Threads
  - Networking

Exceptions, IO, and Threads

- **Exceptions** come in two basic types:
  - **Unchecked** exceptions that do not need to be handled.
  - **Checked** exceptions that must be rethrown or handled with a try/catch.
- **Data** comes in two basic types:
  - **characters** - ASCII characters.
  - **bytes** - positive 8-bit integers.
- **IO** is handled by two kinds of objects:
  - **Readers/Writers** work with character data.
  - **Input/Output Streams** work with bytes.
- **Threads** allow a program to do more than one thing at the same time.

Believe it or not, there is a method to what can feel like the madness of CS2.

We teach exceptions when we do because exception handling is an important part of IO and Threads.

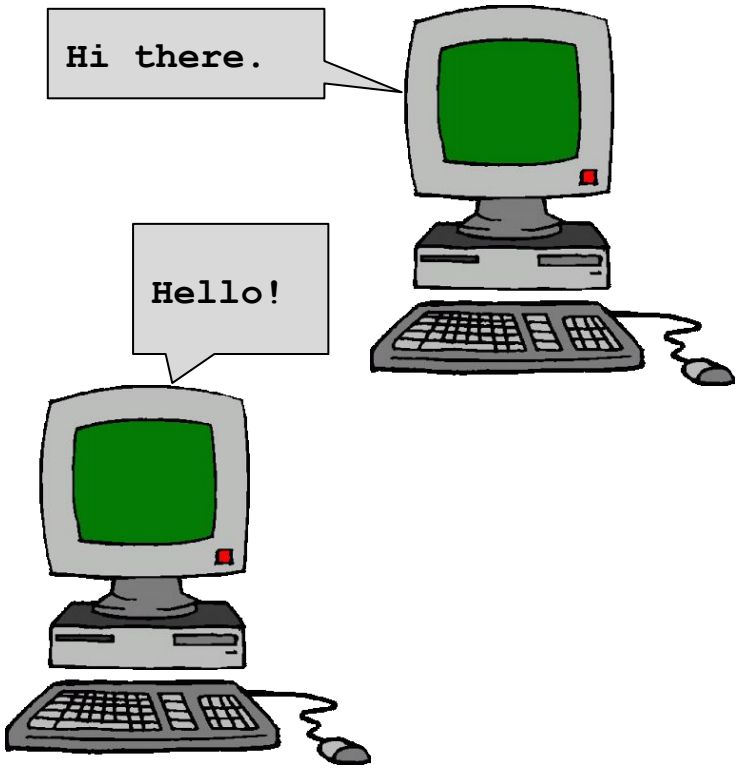
We teach IO when we do because reading and writing data is an important part of programming.

We teach Threads when we do because concurrent programming is also a necessary skill.

And once you have all of those tools in your toolbox, you are finally ready for **networking**!

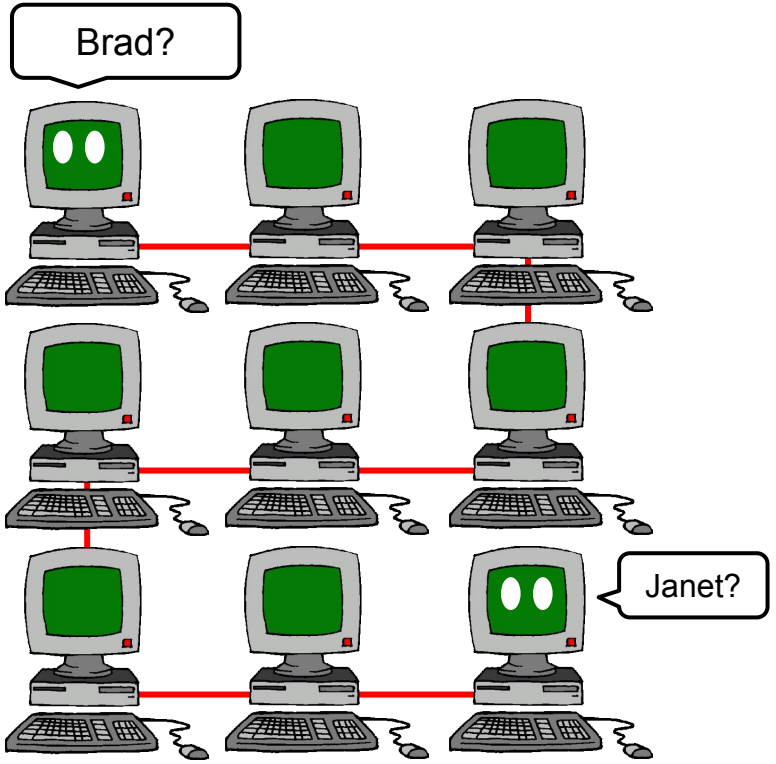
Networking

- A network comprises the hardware and software that provides two or more computers with the ability to communicate with each other.
- Provided that one computer is listening, another computer can establish a connection.
- Both computers can then send and receive data over the connection.
    - The data may be characters or binary data (bytes).



TCP/IP

- TCP/IP is the most widely used communications protocol on the internet.
- It is actually two protocols working together.
  - The Internet Protocol (IP) runs in the network layer. It handles routing and relaying of packets of information.
  - The Transmission Control Protocol (TCP) establishes connections between two computers and helps to insure that packets are delivered in order, reliably, and without corruption.
- Most of what we will be talking about today will be how to send and receive data between two processes (on the same or different computers) using TCP/IP.



Before one computer can communicate with another, it needs to find the other computer on the network using its **address**.

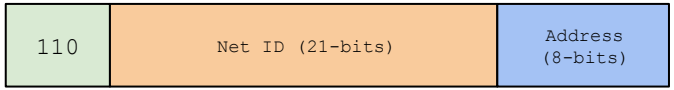
Internet Protocol Addresses

- Every Internet-connected system must have a unique IP address.
- IPv4 uses a 32-bit (4-byte) binary number, typically represented as a "dotted quad," e.g. `129.21.22.196`
  - Each integer ranges from 0-255.
- The relatively small size of an IPv4 address limits the number of computers that can be connected to the Internet.
- Because of this, many computers on private networks are connected to the Internet through a single router or proxy.
  - Only the router gets an Internet IP address.
  - The other computers use private addresses, e.g. `192.168.0.100`.

There are several different *classes* that determine how the bits in the address are used...

Class A
(16 million unique addresses)

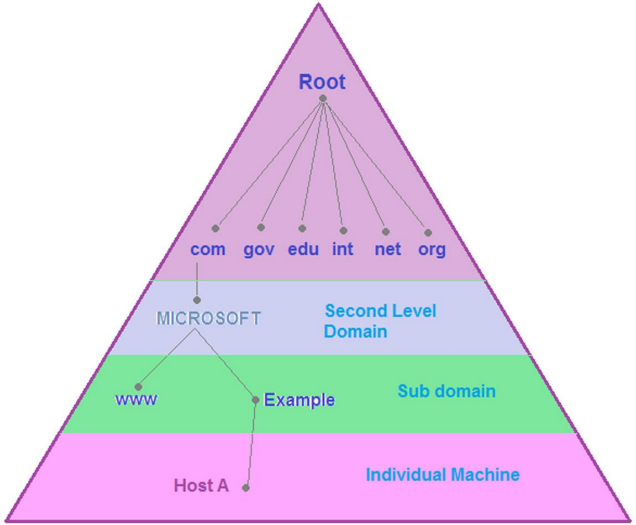
Class B
(65 thousand unique addresses)



Class C
(254 unique addresses)

Hostnames & DNS

- The IP address is ultimately used to send and receive data to and from a computer.
- But if is often the case that a computer is identified by its human readable hostname rather than its machine address.
  - This is particularly the case when the network uses the dynamic host control protocol (DHCP), and a computer's address may change at any time.
- A domain name server (DNS) on the network provides a service that maps a hostname to real address for the computer.
  - A hostname is used to look up the address before trying to communicate.



DNSs on the internet are arranged into a hierarchy, with the *root* servers at the top and individual machines at the bottom.

InetAddress

- The `java.net.InetAddress` class is a class that represents internet addresses. It provides `static` methods that can be used to obtain the `InetAddress` for some hostname or IP address.
  - `getByName(String host)` - gets the InetAddress for the specific hostname or IP address.
  - `getLocalHost()` - gets the InetAddress for the local computer.
- Once created, an `InetAddress` provides a few useful methods:
  - `getHostAddress()` - returns the IP address as a String.
  - `getHostname()` - returns the hostname.

There is a special address that always refers to the local computer called the **loopback address**: `127.0.0.1`.

There is also a special hostname that refers to the local computer: ***localhost***.

Using either of these can be a shortcut to refer to the local computer (rather than its real address or hostname).

Activity: InetAddress

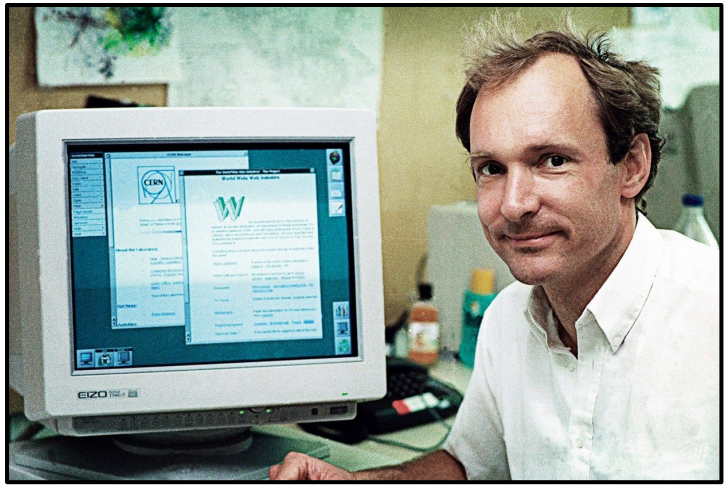
Create a new class, `PrintHostInfo`, with a `main` method that does the following:

- Create an `InetAddress` for your local computer.
- Print the ***host name***.
- Print the ***IP address***.
- What exception(s) do you need to handle?

```
$> java PrintHostInfo
glados.cs.rit.edu
129.21.22.196
```

The World Wide Web

- In 1989, Tim Berners Lee proposed the World Wide Web.
- It was designed to allow people to work together by combining their knowledge in a *web* of *hypertext documents*.
  - A **hypertext document** is a document that contains links to other documents.
- The Hypertext Transfer Protocol (HTTP) defines several different methods that a client can call on a server.
  - These include operations like GET, POST, and DELETE.
  - The client *always* initiates communication by sending the server a request.
  - The server *always* responds.



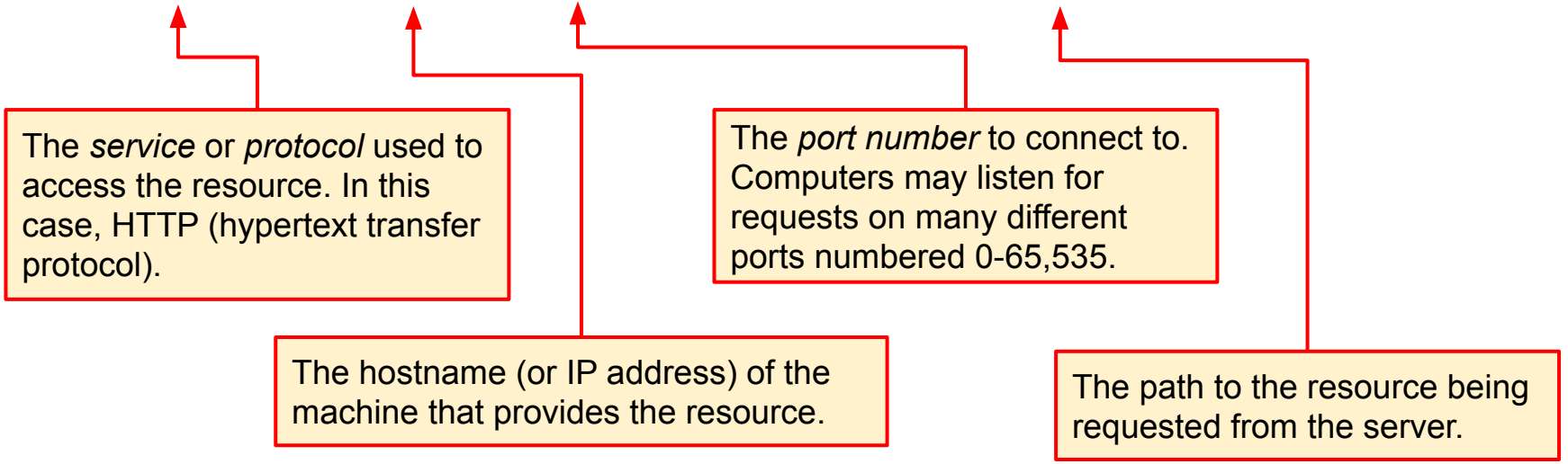
HTTP and HTML are both text-based, which means that, while they are meant for machines to interpret, they are human readable.

That makes HTTP an easy example protocol to learn because you can see and understand the messages.

URLs

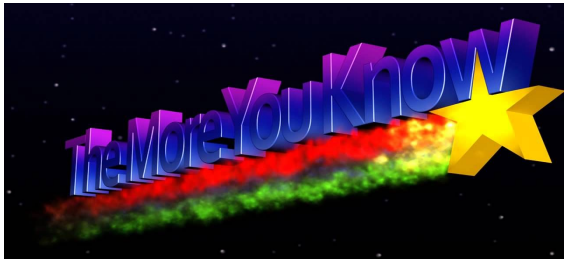
- A Uniform Resource Locator (URL) defines the unique location of a resource on the web in the following way:
  - `service://host:port/file and resource details`
- For example:

`http://cs.rit.edu:80/~csci142/Lectures/08/Threads-java.pdf`



URL & URLConnection

- The `java.net.URL` class represents a URL in Java. A `URL` can be created with a string:
  - `URL insta = new URL("http://www.instagram.com");`
- The first part of the URL string identifies the protocol that should be used to communicate with the resource.
  - There are several, e.g. HTTP, HTTPS, and FTP.
  - If Java does not understand the protocol, a checked `MalformedURLException` will be thrown.
- The `URL::openConnection()` method returns a `URLConnection`, which provides many methods, e.g.:
  - `connect()` - establishes a connection to the resource.
  - `getInputStream()` - returns an InputStream that can be used to read data from the resource (e.g. HTML).
  - `getOutputStream()` - returns an OutputStream that can be used to write data to the resource.



Each protocol defines its own message format for sending and receiving data.

Because of this, `URLConnection` is an **abstract** class. Its subclasses provide implementations for many of its methods.

e.g. `HTTPURLConnection` implements the HTTP protocol, and provides a few more HTTP-specific methods.

Activity: URLConnection

Create a new class, `GooglePrinter`, with a main method that does the following:

- Create a URL to http://www.google.com
- Establish a `URLConnection`.
- Get the `InputStream` from the connection and use it to read (and print) the contents of the Google homepage.
  - Hint: Use a `BufferedReader` to read one line at a time.

TCP/IP Connections With Sockets

- `URL` and `URLConnection` are fine for communicating using established internet protocols like HTTP.
- But what if you want to establish your own TCP/IP connection to send and receive data using some other protocol (e.g. one of your own design)?
- The `java.net.Socket` class is used to establish a TCP/IP connection to a specific hostname (or IP address) and port.
  - `Socket sock = new Socket("localhost", 1234);`
- Once established, the socket can be used to send or receive data to/from the other computer.
  - We'll discuss exactly how to send and receive data presently.

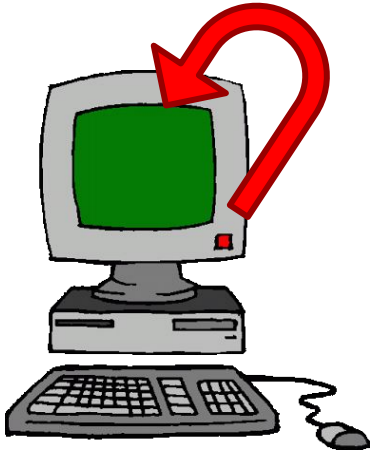
A socket connection can only be established to a **port** on which the other computer is actively listening for connections.

Trying to connect to a **port** that is not being used to listen for connections will cause an error.

In fact, pretty much everything you do with a socket might throw a checked exception.

The good news is that all of the exceptions extend `IOException`, and so that is the only one that you really need to handle.

Activity: Open a Socket



Create a new class, `EchoClient`, and add a `main` method that:

- Uses `InetAddress` to get the host name for the local computer.
- Uses the host name to open a socket connection on port `33075`.

What happens when you run the code?

Listening for Connections

- Before a client can connect to a server, the server must begin listening for incoming connections on an available port.
  - The port is a positive number between 1 and 65,535.
  - Many ports are in use by the operating system or reserved.
- The `java.net.ServerSocket` class can be used to listen for TCP/IP connections on a specified port.
  - `ServerSocket s = new ServerSocket(12345);`
- The `accept()` method on your `ServerSocket` will block until a client establishes a connection.
  - It returns a `Socket` that can be used to communicate with the client, i.e. by using the Socket's input and output streams.

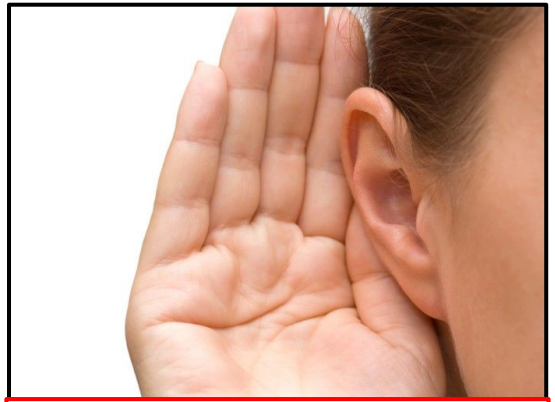
In networking, the **server** is the host listening for incoming connections and the **client** is the host that establishes the connection.

It is usually the case that the **server** is providing some useful service that the **client** would like to use.

In Java, once a connection is established, both ends of the connection are represented as Java `Sockets`.

This means that both the **client** and **server** send and receive data in the same way.

Activity: Listening for Connections



What will happen if the specified port is already in use?

Create a new class, `EchoServer`, and add a `main` method that:

- Creates a `ServerSocket` that listens for incoming connections on port `33075`.
- Uses the `accept()` method to create a connection with a client. When the connection is established, print a message.
- Run your `Server`.
- Run your `Client`.

Sending Data

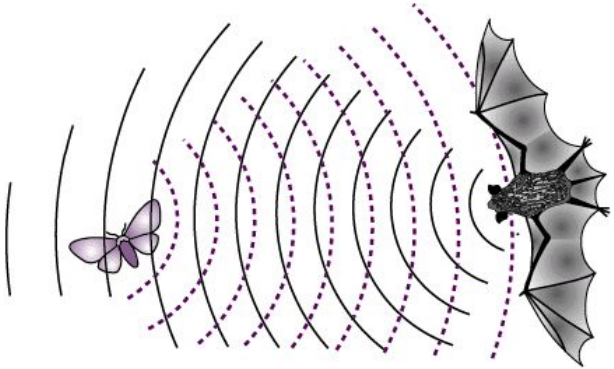
- Once a connection is established, both sides of the connection are represented as `Sockets`.
- The `getOutputStream()` method on the `Socket` will return an `OutputStream` that can be used to send data to the other end of the connection.
- Recall that Java streams are used to send binary data. If text data is preferred, the `OutputStream` may be wrapped in an instance of the `java.io.PrintWriter` class.
  - `PrintWriter writer = new PrintWriter(stream);`
- A `PrintWriter` provides lots of useful methods:
  - `println(String)` - writes a string of text terminated with a newline.
  - `write(char[])` - writes a character buffer.
  - `flush()` - sends any buffered data to the other side.

It can be helpful to think of the two ends of the connection like a telephone.

Audio sent through the microphone (**output stream**) on one end...

...is heard through the speaker (**input stream**) on the other.

Activity: Sending Data



Modify the `main` method in your `EchoClient` class:

- Get the `OutputStream` from the `Socket`.
- Create a `PrintWriter` using the `OutputStream`.
- Send the string `"Hello, Server!"` using the PrintWriter.
  - Don't forget to close your `Socket` at the end!
- Run your `Server` and then your `Client`. What happens?!

Receiving Data

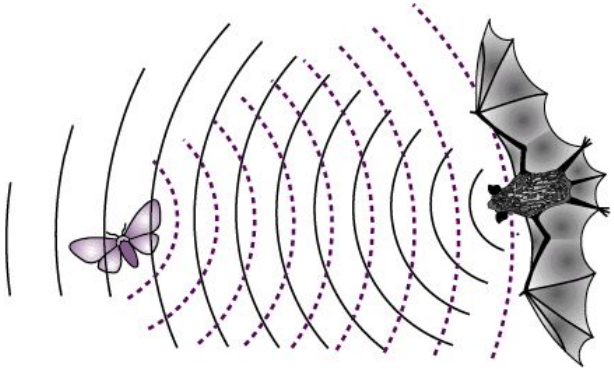
- Once a connection is established, both sides of the connection are represented as `Sockets`.
- The `getInputStream()` method on the `Socket` will return an `InputStream` that can be used to receive data from the other end of the connection.
- Recall that Java streams are used to read binary data. If text data is preferred, the `InputStream` may be wrapped in an instance of the `java.util.Scanner` class.
  - `Scanner sc = new Scanner(stream);`
- Remember, a `BufferedReader` provides the `readLine()` method that will read the next line of text.
  - You will need to create an `InputStreamReader` first.

It can be helpful to think of the two ends of the connection like a telephone.

Audio sent through the microphone (**output stream**) on one end...

...is heard through the speaker (**input stream**) on the other.

Activity: Receiving Data



Modify the `main` method in your `Server` class:

- Get the `InputStream` from the client `Socket`.
- Create a `BufferedReader` using the `InputStream`.
- Read the first string sent by the client and print it out.
- Run your `Server` and then your `Client`. What happens *now*?!

A Typical Client & Server

```
ServerSocket server = new ServerSocket(42975);
Socket client = server.accept();

InputStream fromClient =
    client.getInputStream();

OutputStream toClient =
    client.getOutputStream();
```

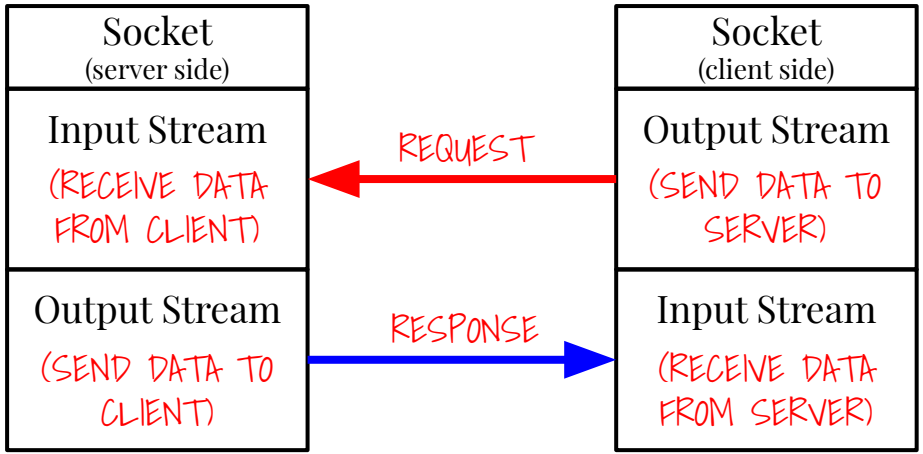
```
Socket sock =
    new Socket("ahost.rit.edu", 42975);

OutputStream toServer =
    sock.getOutputStream();

InputStream fromServer =
    sock.getInputStream();
```

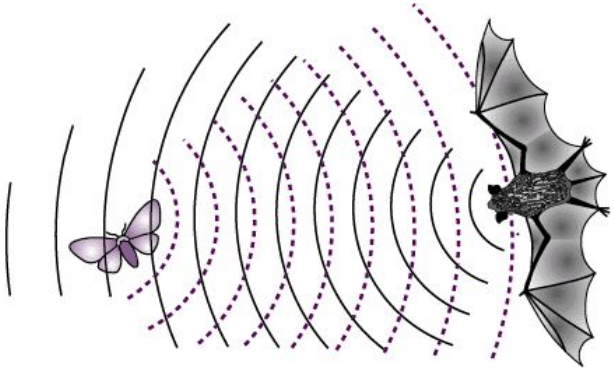
The **server** must begin listening for connections on a specific port.

Once the connection is established, the **server** uses streams to send/receive data to/from the client.



Next, the **client** must establish the connection to the server on the correct port.

Once the connection is established, the **client** uses streams to send/receive data to/from the server.



Activity: Finishing the Echo Client & Server

Complete rout echo client/server so that the server sends any messages received back to the client.

- Finish the `EchoServer` so that it sends any message that it receives from the client back to the client.
- Finish the `EchoClient` to print the response out.

Common Networking Errors & Causes

Error	When It Happens
BindException	Port is already in use by another process.
ConnectionRefused	Remote computer is not listening on the port.
ConnectionClosed	Tried to read/write using a closed socket.
UnknownHost	The hostname or IP can't be found on the network.
IOException	A general IO problem. All of the above are IOExceptions.