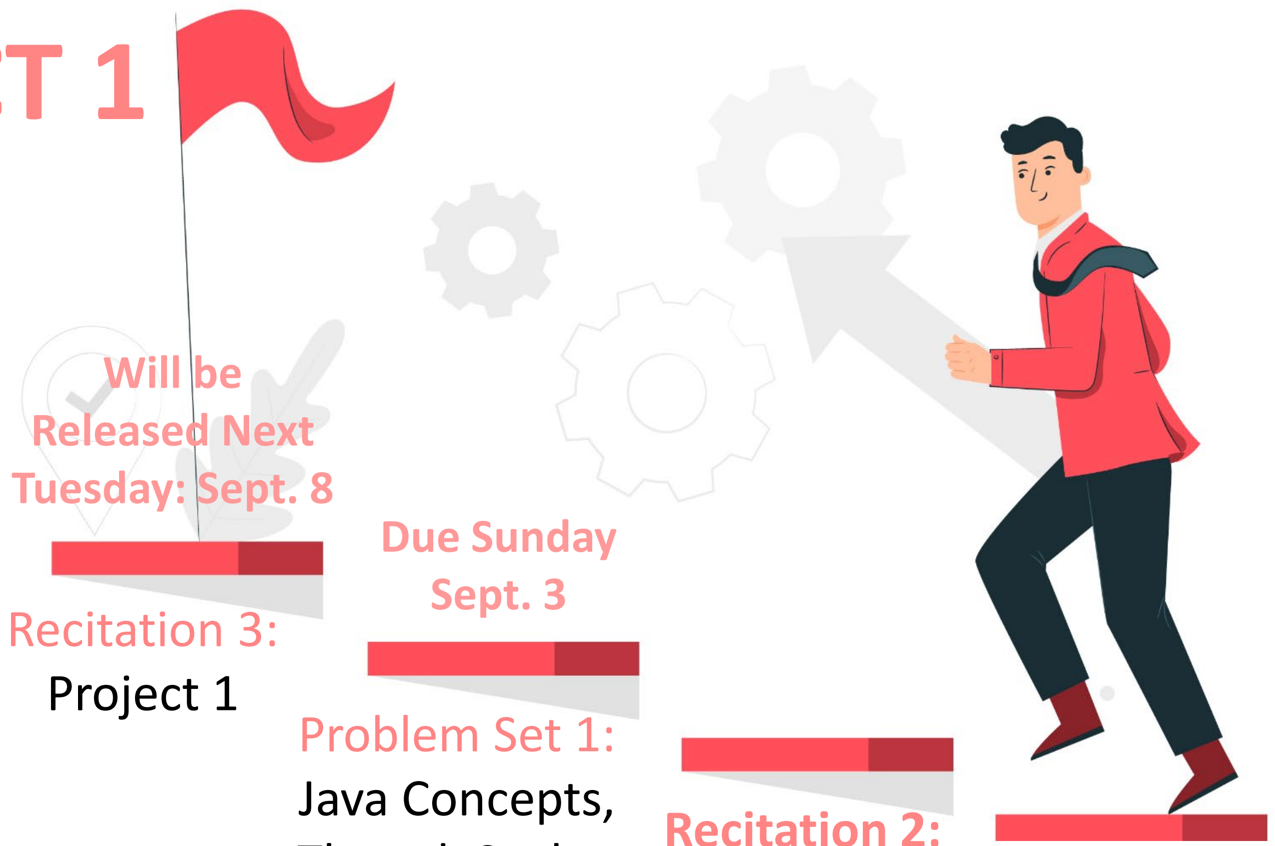
15-440 Distributed Systems Recitation 2

Slides By: Hend Gedawy

& Tamim Jabban

Big Picture

PROJECT 1



Will be Released Next Tuesday: Sept. 8

Recitation 3:
Project 1

Due Sunday Sept. 3

Problem Set 1:
Java Concepts, Thread, Socket Programming

Recitation 2:
Java Threads and Socket Programming

Recitation 1:
Java Concepts

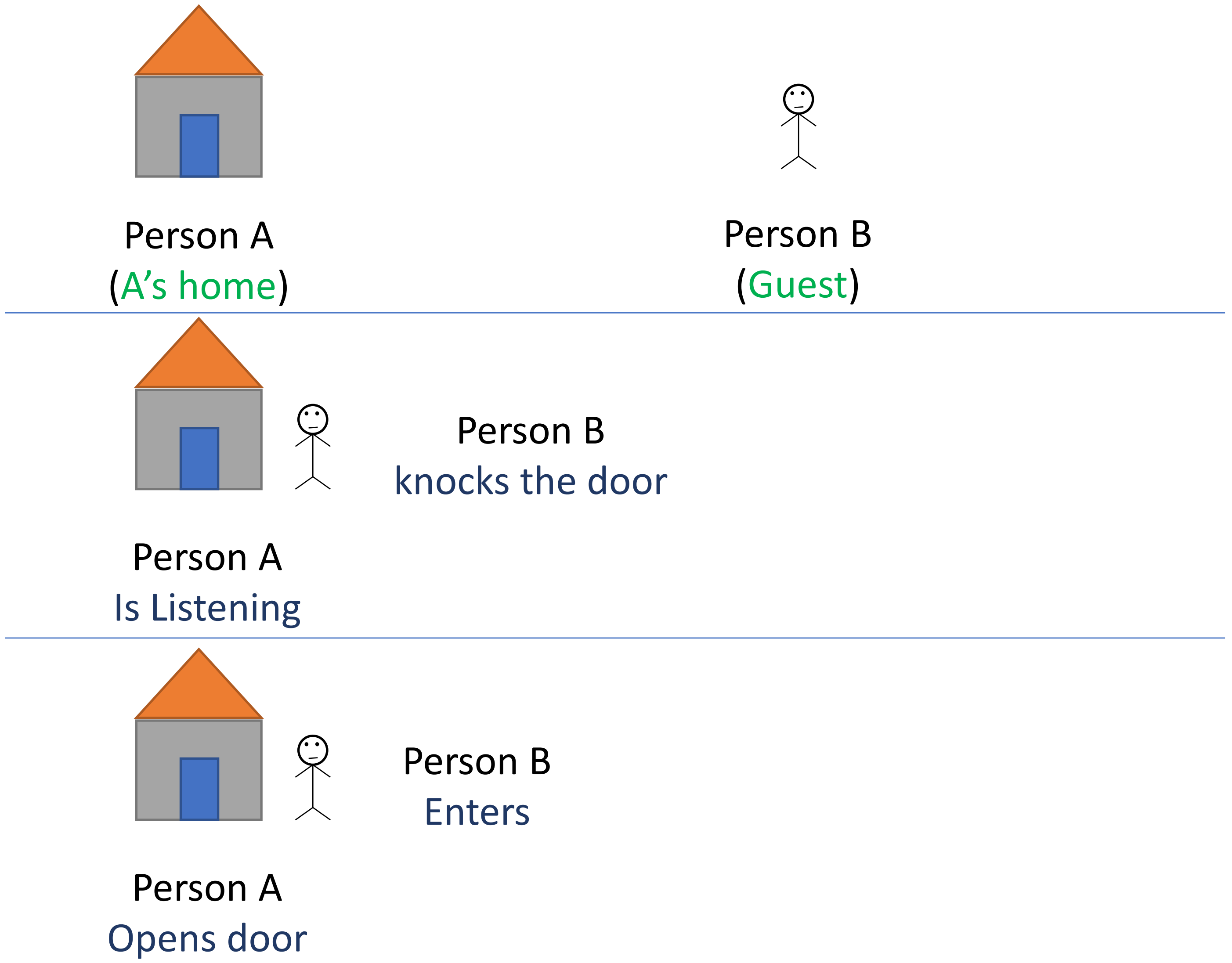
Outline

- **Communication via Sockets in Java**
- Multi-threading in Java
- Coding a full Client-Server Example
  On Eclipse, we’ll code an “echo” TCP Server-Client Example

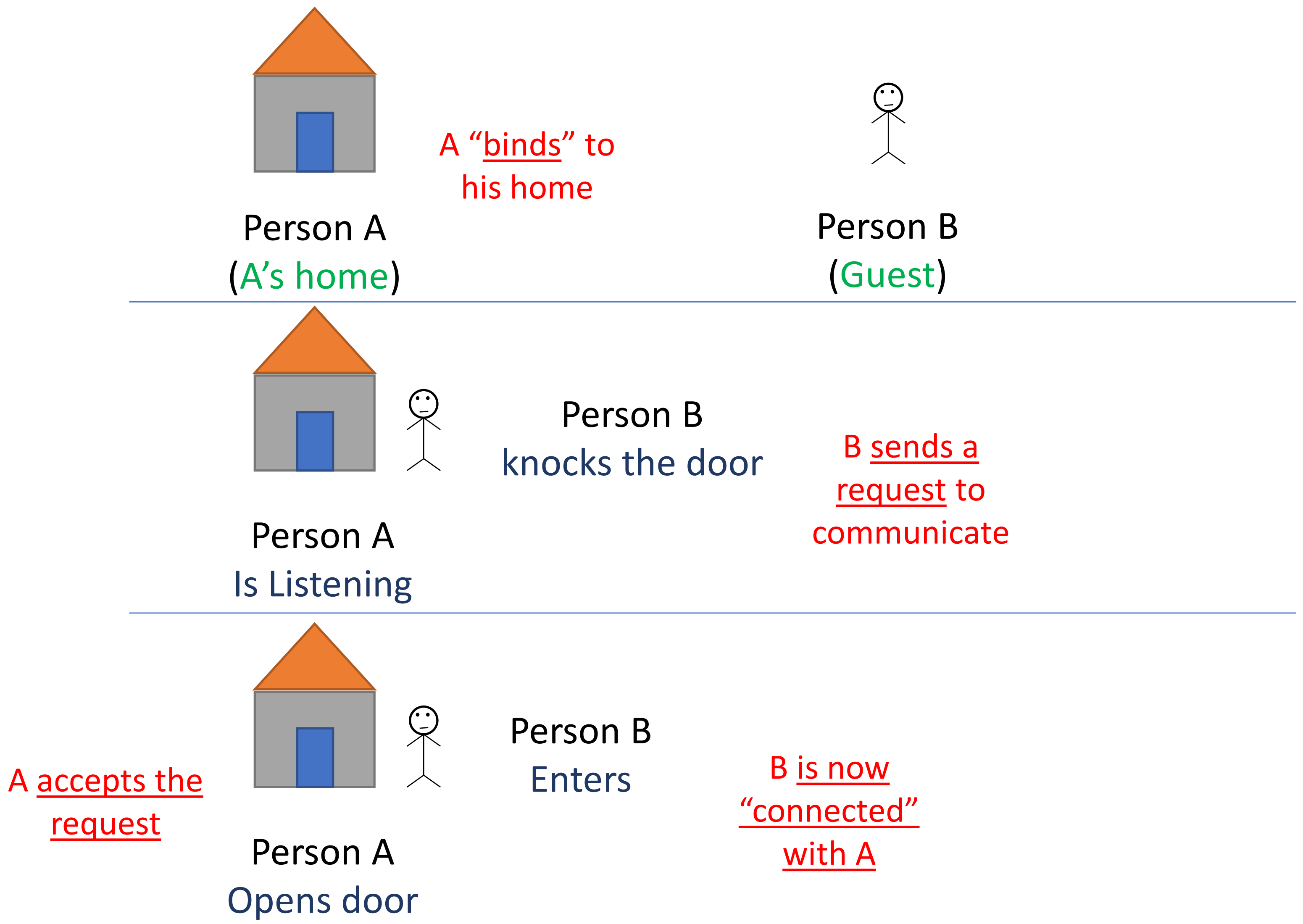
Communication via Sockets

- Sockets provide a communication mechanism between networked computers.
- A **Socket** is an end-point of communication that is identified by an IP address and port number.
- A client sends requests to a server using a client socket.
- A server receives clients' requests via a listening socket

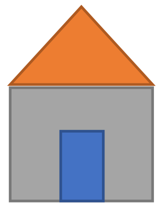
Communication via Sockets



Communication via Sockets

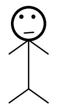


Communication via Sockets

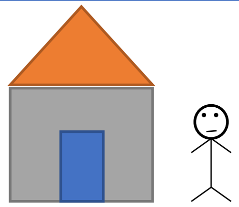


Server A

A binds to socket address:
(1) IP address
(2) Port number

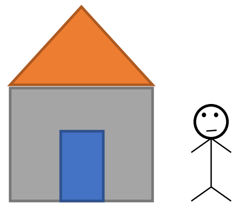


Client B



Server A is Listening to Requests

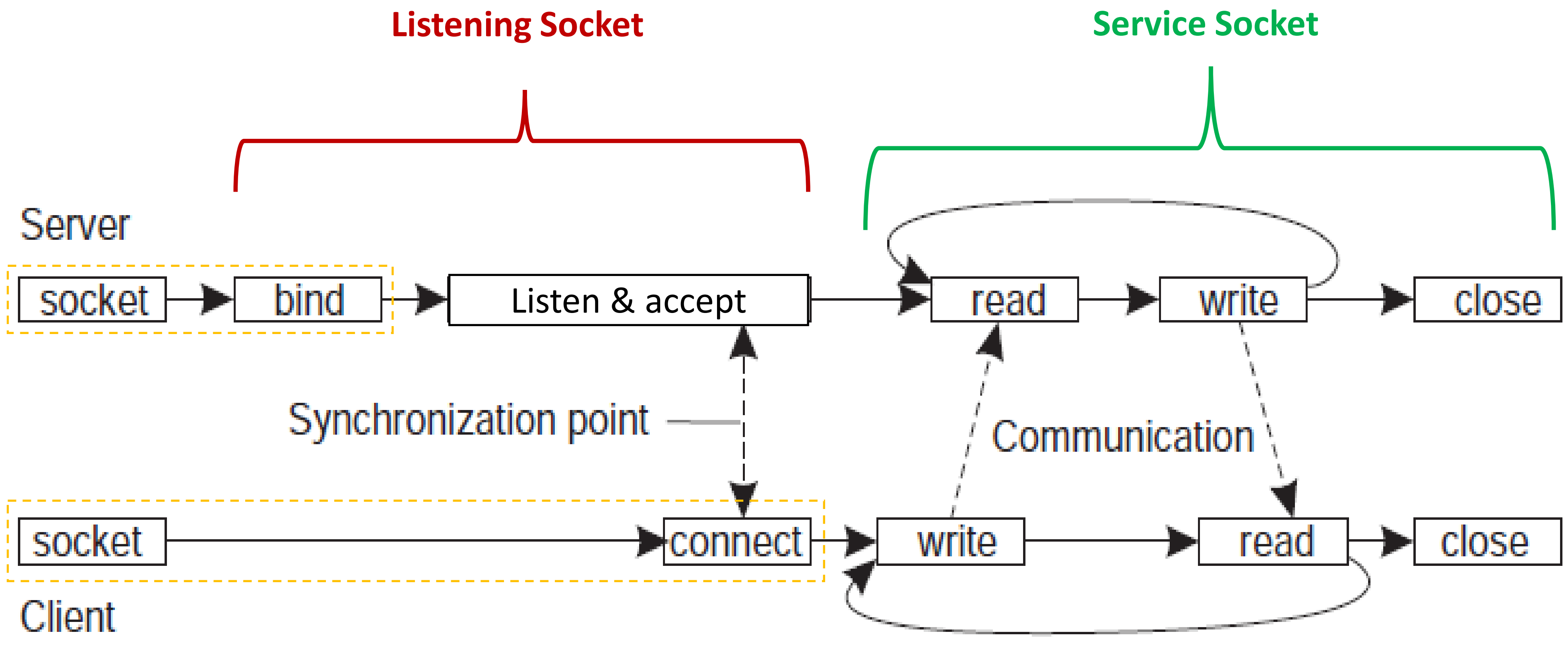
Client B sends a request to communicate with the server



Server A accepts request

Client B is now connected with Server A

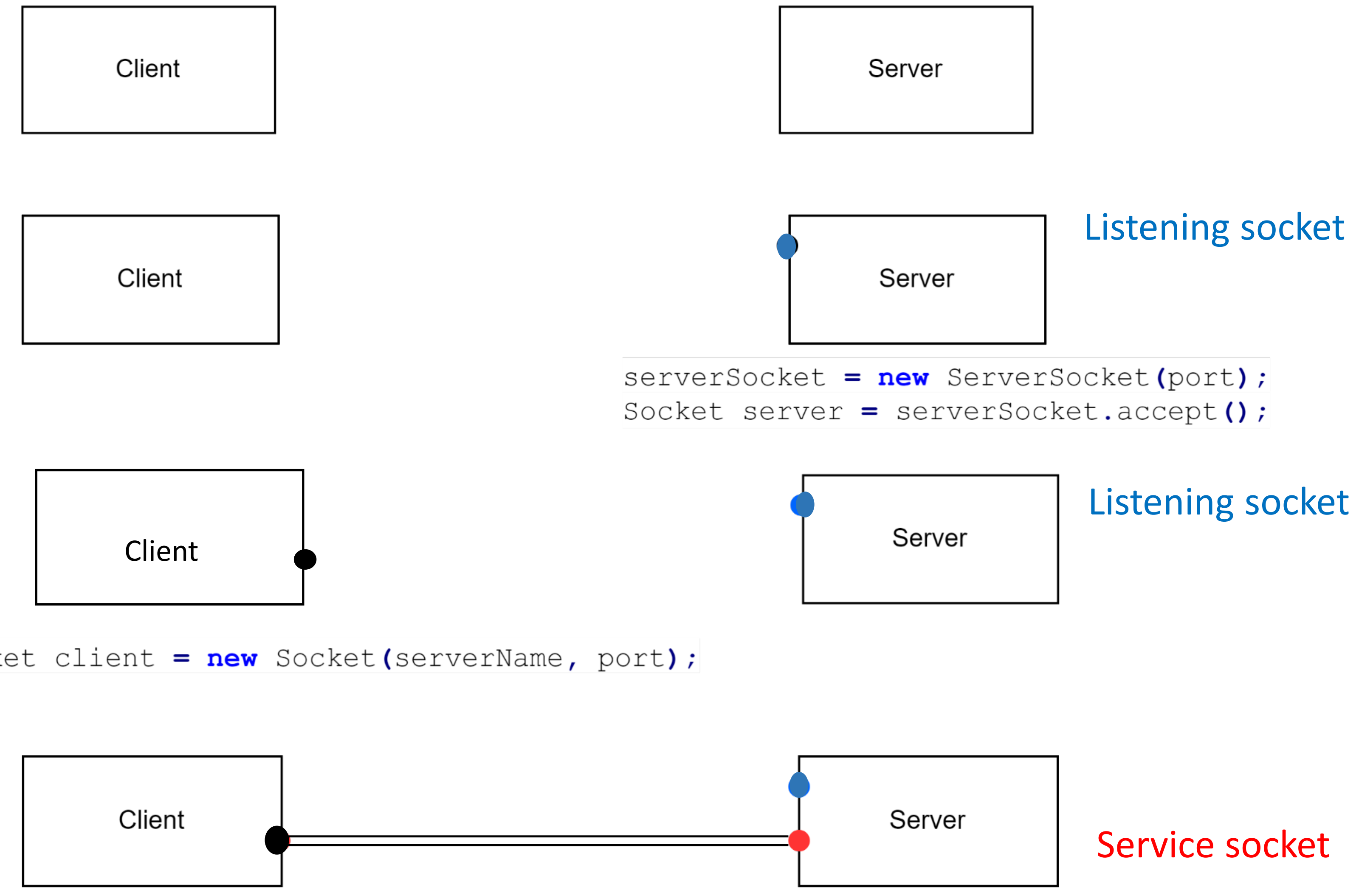
Communication via Sockets



When writing the code, these steps could be merged

More on that later!

Socket Communication Recipe



```
serverSocket = new ServerSocket(port);
Socket server = serverSocket.accept();
```

```
Socket client = new Socket(serverName, port);
```

ServerSocket Methods

✓ Methods we will use in the demo

SN	Methods with Description
1	**public ServerSocket(int port)** Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.
✓ 2	**public ServerSocket()** Creates an unbound server socket. When using this constructor, use the bind() method when you are ready to bind the server socket.
✓ 3	**public void bind(SocketAddress host)** Binds the socket to the specified server and port in the SocketAddress object. Use this method if you instantiated the ServerSocket using the no-argument constructor.
✓ 4	**public Socket accept()** Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely.
5	**public SocketAddress getLocalSocketAddress()** Returns the address of the endpoint this socket is bound to, or null if it not bound yet.
✓ 6	**public void close()** Closes the socket

There are two ways to create and bind ServerSocket:

1) **ServerSocket(int port):** which will create the socket and bind it with the given port
2) **InetSocketAddress(port)** + **ServerSocket()**+ **bind(address)**

Socket Methods

✓ Methods we will use in the demo

	SN	Methods with Description
	1	**public Socket(String host, int port)** This method attempts to connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server.
✓	2	**public Socket()** Creates an unconnected socket. Use the connect() method to connect this socket to a server.
✓	3	**public void connect(SocketAddress host)** This method connects the socket to the specified host. This method is needed only when you instantiated the Socket using the no-argument constructor.
✓	4	**public InputStream getInputStream()** Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.
✓	5	**public OutputStream getOutputStream()** Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket
	6	**public SocketAddress getLocalSocketAddress()** Returns the address of the endpoint this socket is bound to, or null if it is not bound yet.
✓	7	**public void close()** Closes the socket, which makes this Socket object no longer capable of connecting again to any server

There are two ways to create and connect a client socket:

1) **Socket(String host, int port)**
   - You can use “127.0.0.1” for local host
2) **InetSocketAddress(String host, int port) + Socket() + connect(SocketAddress host)**

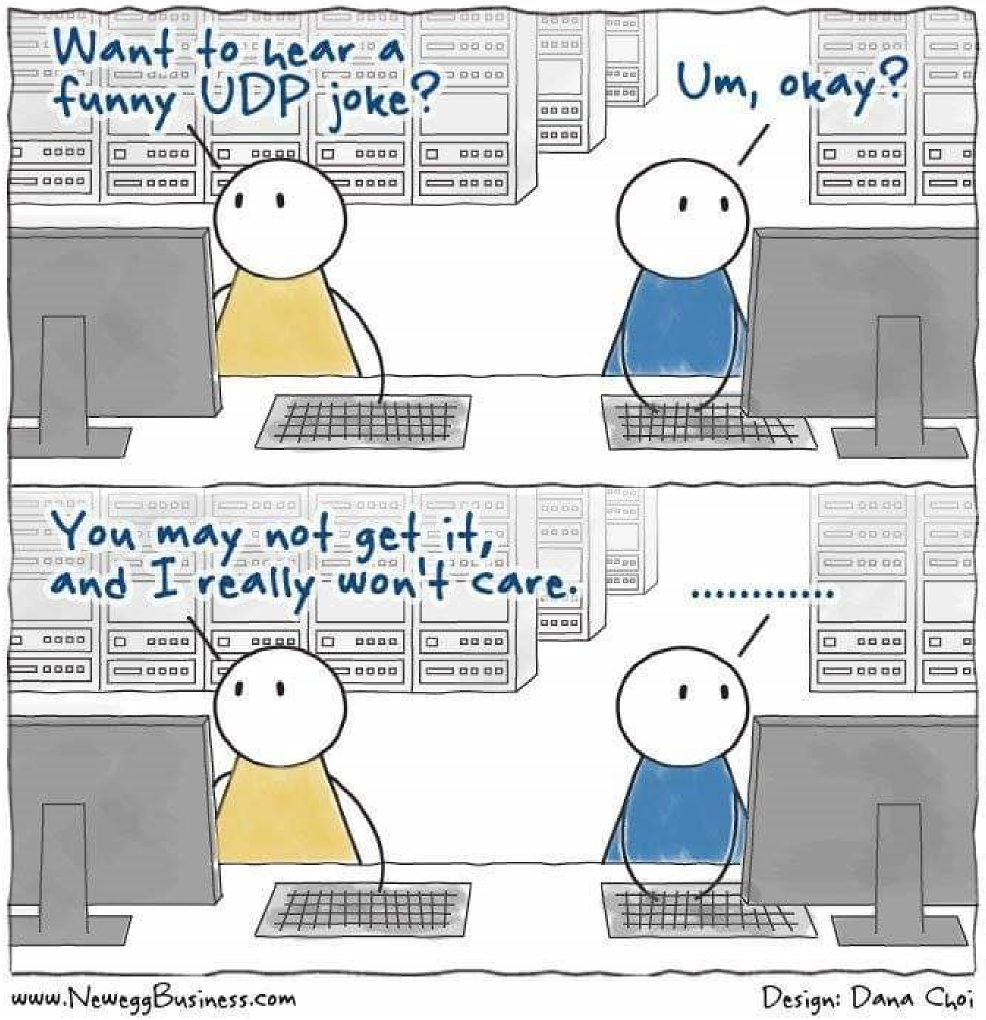
Transport Protocols

- **Socket:** endpoint to read and write data
- Each Socket has a **network protocol**
- Two types of **protocols** used for communicating data/*packets* over the internet:
  - TCP:
    - ***Transmission Control Protocol***
    - Connection Oriented (*handshake*)
  - UDP:
    - ***User Datagram Protocol***
    - “Connectionless”

Transport Protocols

TCP

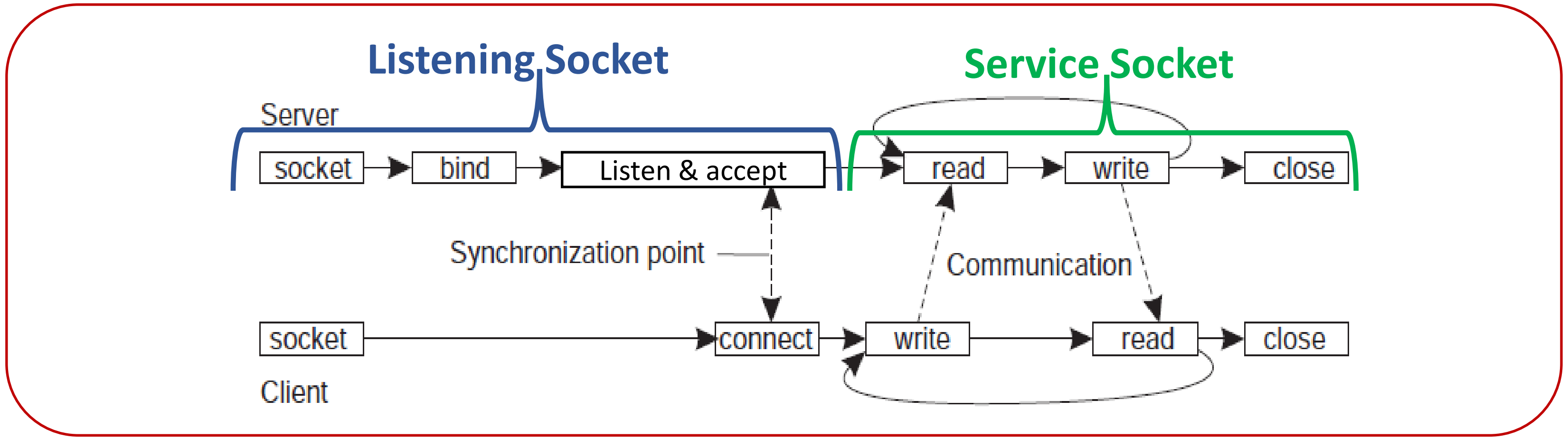
UDP


Want to hear a funny UDP joke?
Um, okay?
You may not get it, and I really won't care.
....
www.NeweggBusiness.com
Design: Dana Choi

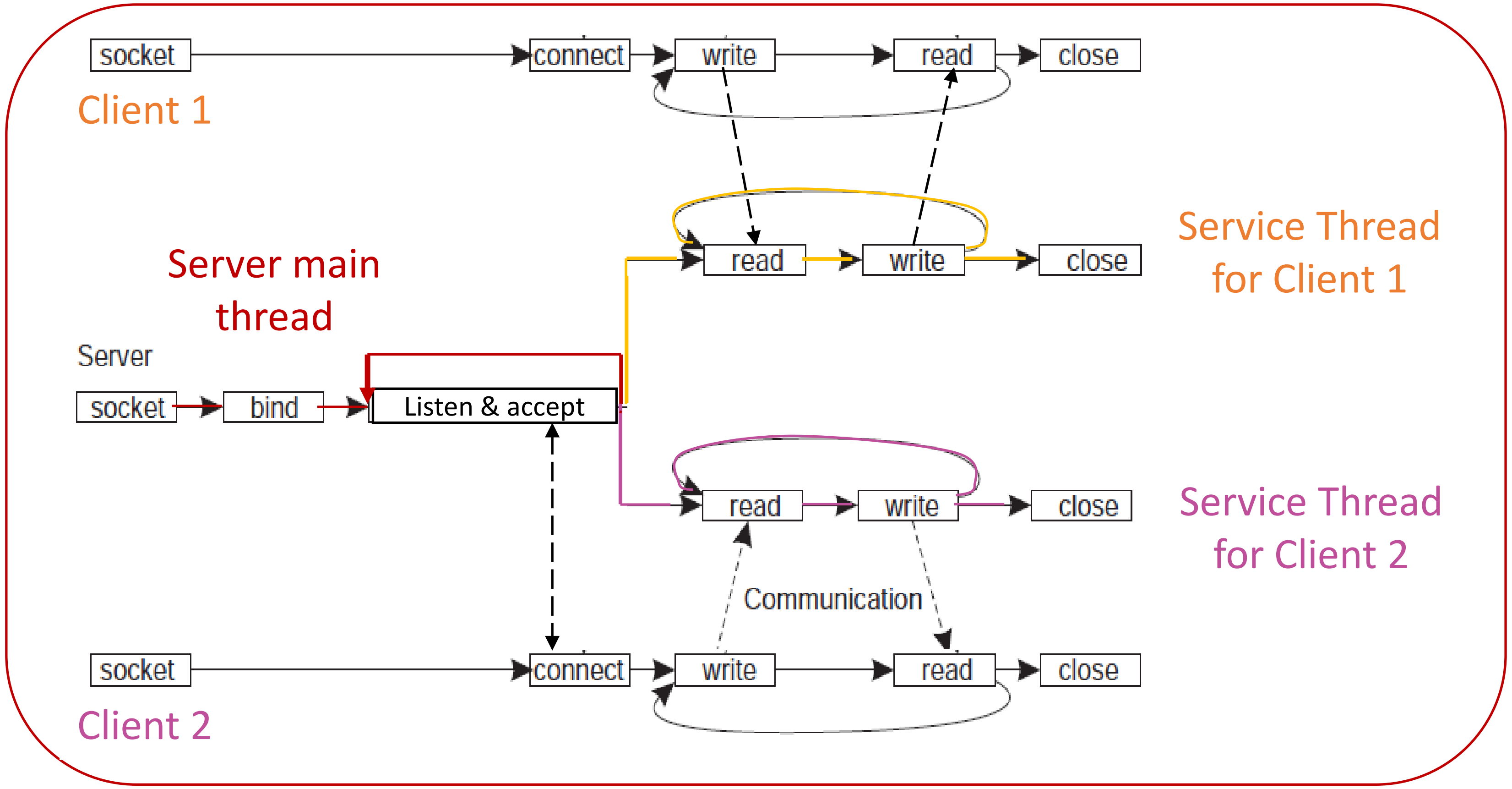
Outline

- Communication via Sockets in Java
- **Multi-threading in Java**
- Coding a full Client-Server Example
  On Eclipse, we’ll code an “echo” TCP Server-Client Example

TCP Single-Threading



TCP Multi-Threading



Multi-Threading in General

- STEP 1: A class intended *to execute as a thread* must implement the ***Runnable*** interface

```
public class Service implements Runnable
```

  - Implement the method ***run()***

```
public void run() {  //thread's logic goes here }
```

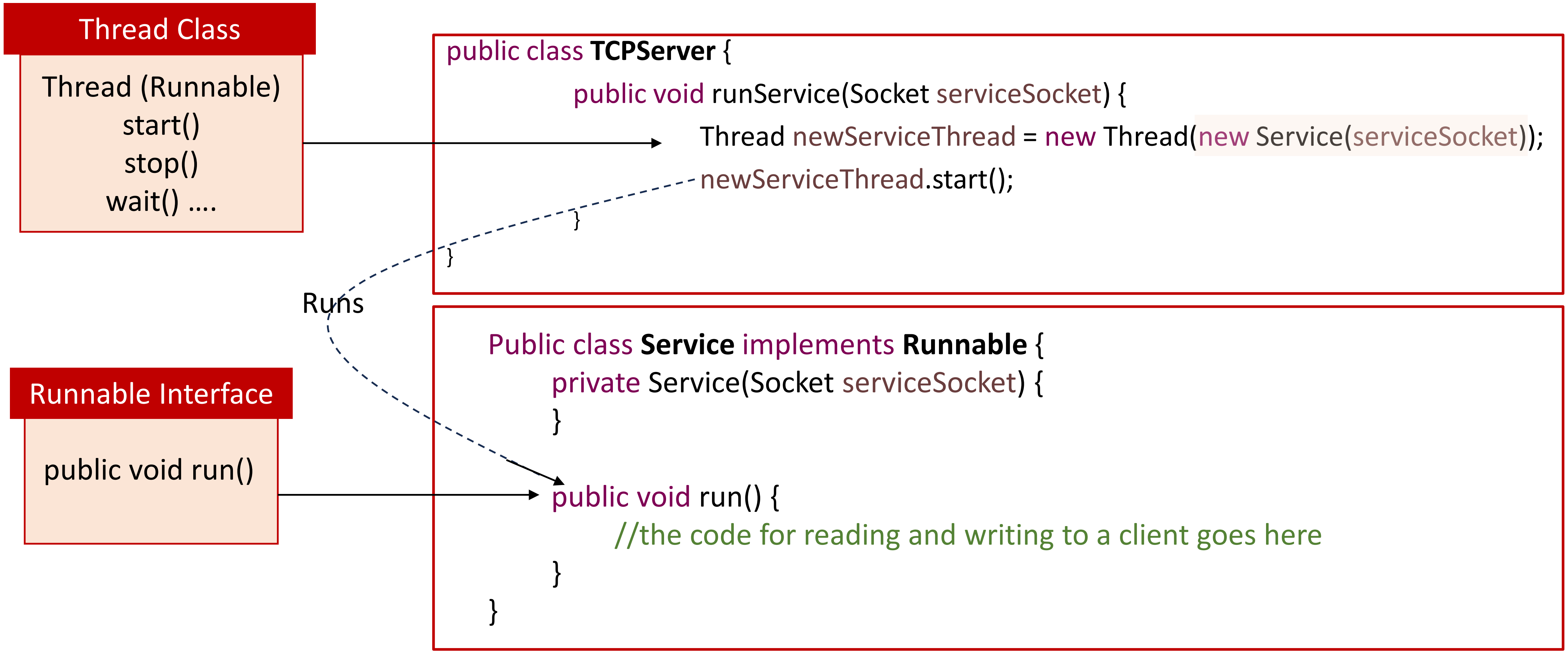
- STEP 2: Instantiate a Thread object *passing an instance of the intended class*

```
Thread t = new Thread(new Service())
```

- STEP 3: Invoke ***start()*** on the new thread

```
t.start()              // invokes the run() method implemented in
                             the Service class
```

TCP Multi-Threading Example



Thread Class

Thread (Runnable)
start()
stop()
wait() ….

Runnable Interface

public void run()

```
public class TCPServer {
            public void runService(Socket serviceSocket) {
                        Thread newServiceThread = new Thread(new Service(serviceSocket));
                        newServiceThread.start();
            }
}
```

```
Public class Service implements Runnable {
      private Service(Socket serviceSocket) {
      }

      public void run() {
            //the code for reading and writing to a client goes here
      }
}
```

Outline

- Communication via Sockets in Java
- Multi-threading in Java
- **Coding a full Client-Server Example**
  On Eclipse, we'll code an "echo" TCP Server-Client Example

Let’s start with Psuedocode

Server

```
serverAddres = new InetSocketAddress(port)
listenSocket= new **ServerSocket**()
listenSocket.**bind**(serverAddres)
While(true)
        serviceSocket= listenSocket.**accept**()
        Thread service= new thread(new
        Service(serviceSocket))
        service.start()
```

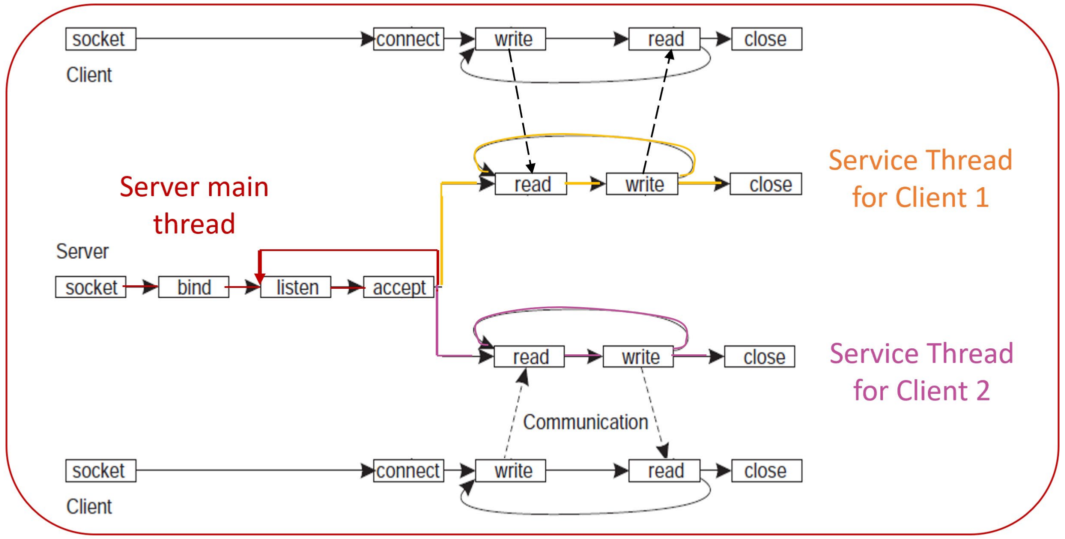
Service implements Runnable

```
While(true)
        Read client message from socket
        Write message back to client
serviceSocket.close()
```

Client

- serverAddres= new **InetSocketAddress**(port)
- clientSocket= new **Socket**()
- clientSocket.**connect**(serverAddres)
- While(true)
  - **Read** user’s **input** message
  - **Write** the message **to the socket**
  - **Read** the echoed message **from the socket**
- clientSocket.**close**()

How to do these?



Useful Java Methods/Classes: To Read User's input

Scanner class allows to read user input.

Scanner(InputStream source)
Constructs a new Scanner that produces values scanned from the specified input stream.

If you pass (System.in), you can read input from the keyboard

Methods to read different input types using the scanner object

Method	Description
`nextBoolean()`	Reads a `boolean` value from the user
`nextByte()`	Reads a `byte` value from the user
`nextDouble()`	Reads a `double` value from the user
`nextFloat()`	Reads a `float` value from the user
`nextInt()`	Reads a `int` value from the user
`nextLine()`	Reads a `String` value from the user
`nextLong()`	Reads a `long` value from the user
`nextShort()`	Reads a `short` value from the user

Useful Java Methods/Classes: To Read and Write to Socket

When you create a socket, you can retrieve the **socket's InputStream and OutputStream** which **allow** you to **write raw bytes to the socket**

public InputStream getInputStream()
public OutputStream getOutputStream()

Java has more classes that build on InputStream and OutputSream to allow writing data in different forms and ways

We will create **ObjectInputStream** and **ObjectOutputStream** objects **to be able to read and write objects instead of raw bytes.**

We will use the following constructors:

ObjectInputStream(InputStream in)

ObjectOutputStream(OutputStream out)

Then we can use the **readObject()**, **writeObject()** methods to read from and write to the socket

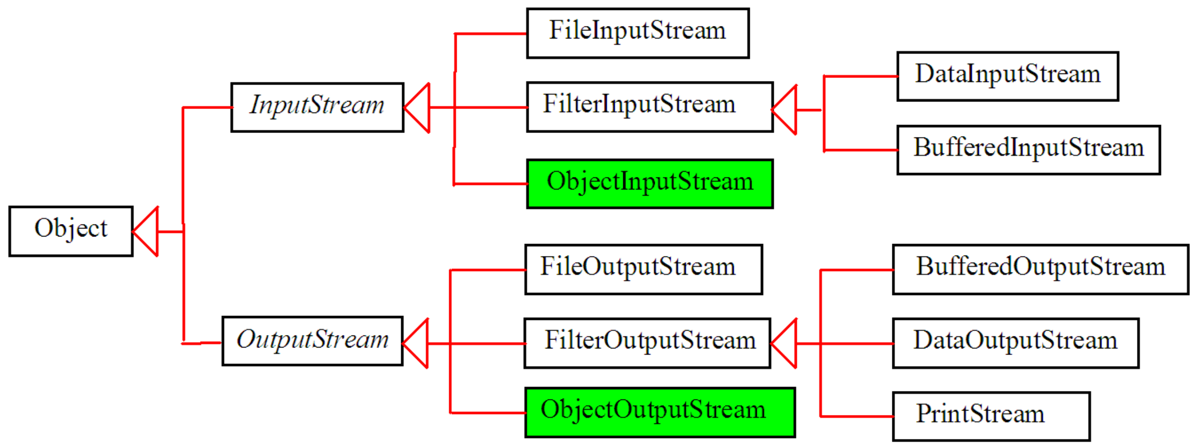


Photo credit: https://voyager.deanza.edu/~hso/cis35a/lecture/java18/intro/io.html

Demo Time ☺