

15-440  
Distributed Systems  
Recitation 2

Tamim Jabban

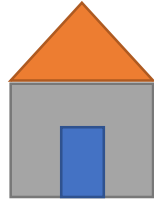
# Agenda

- Communication via Sockets in Java  
*(this enables you to complete PS1 and start P1 (goes out today!))*
- 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

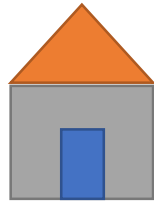
# Communication via Sockets



Person A  
(A's home)



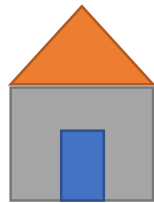
Person B  
(Guest)



Person A  
Is Listening



Person B  
knocks the door

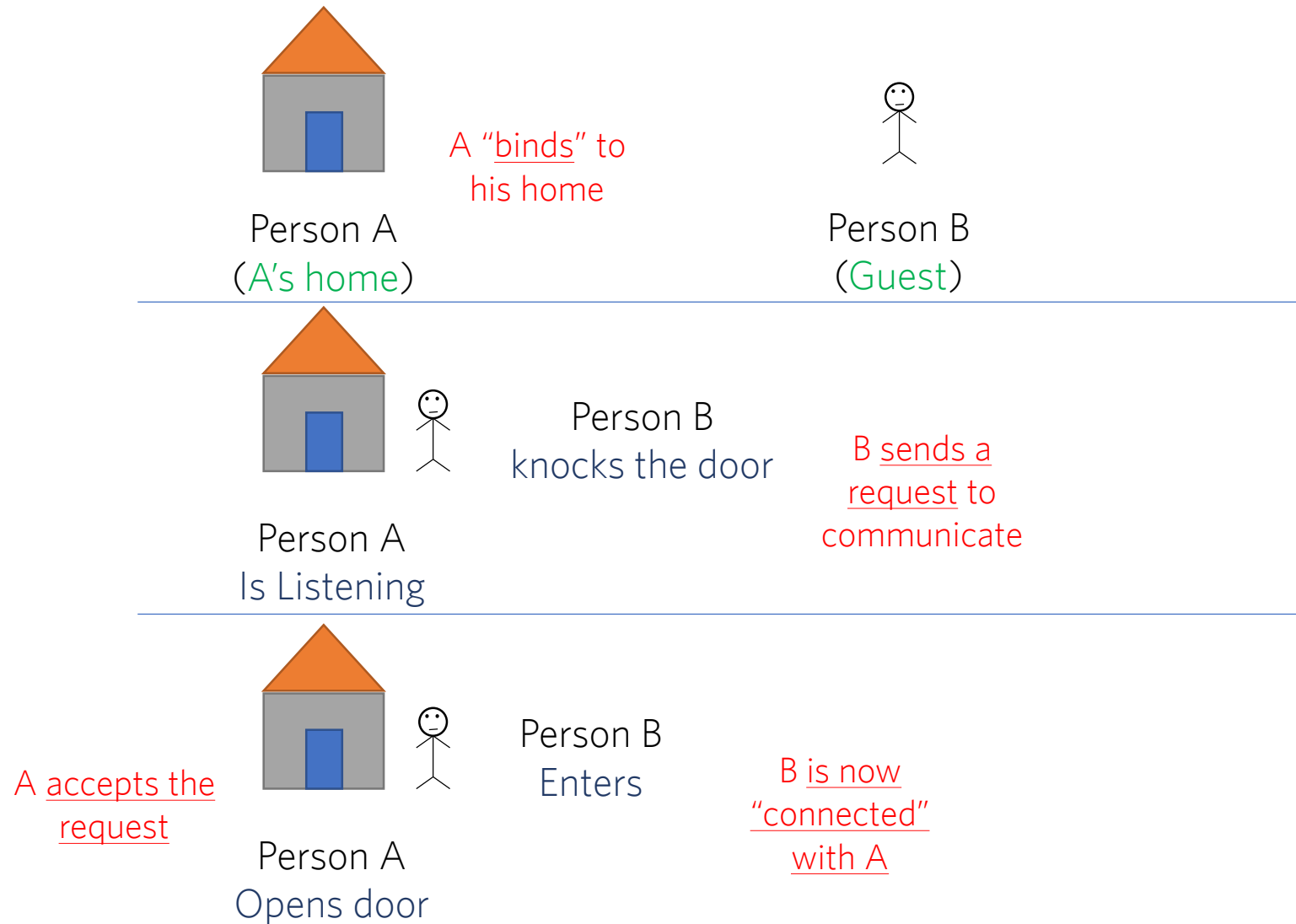


Person A  
Opens door

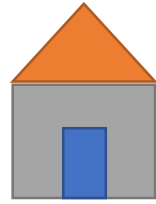


Person B  
Enters

# Communication via Sockets



# Communication via Sockets

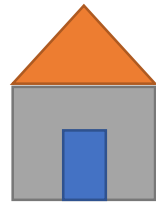


Server A

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



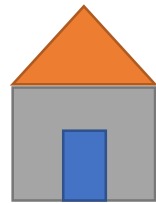
Client B



Server A is Listening  
to Requests



Client B sends a  
request to  
communicate  
with the server

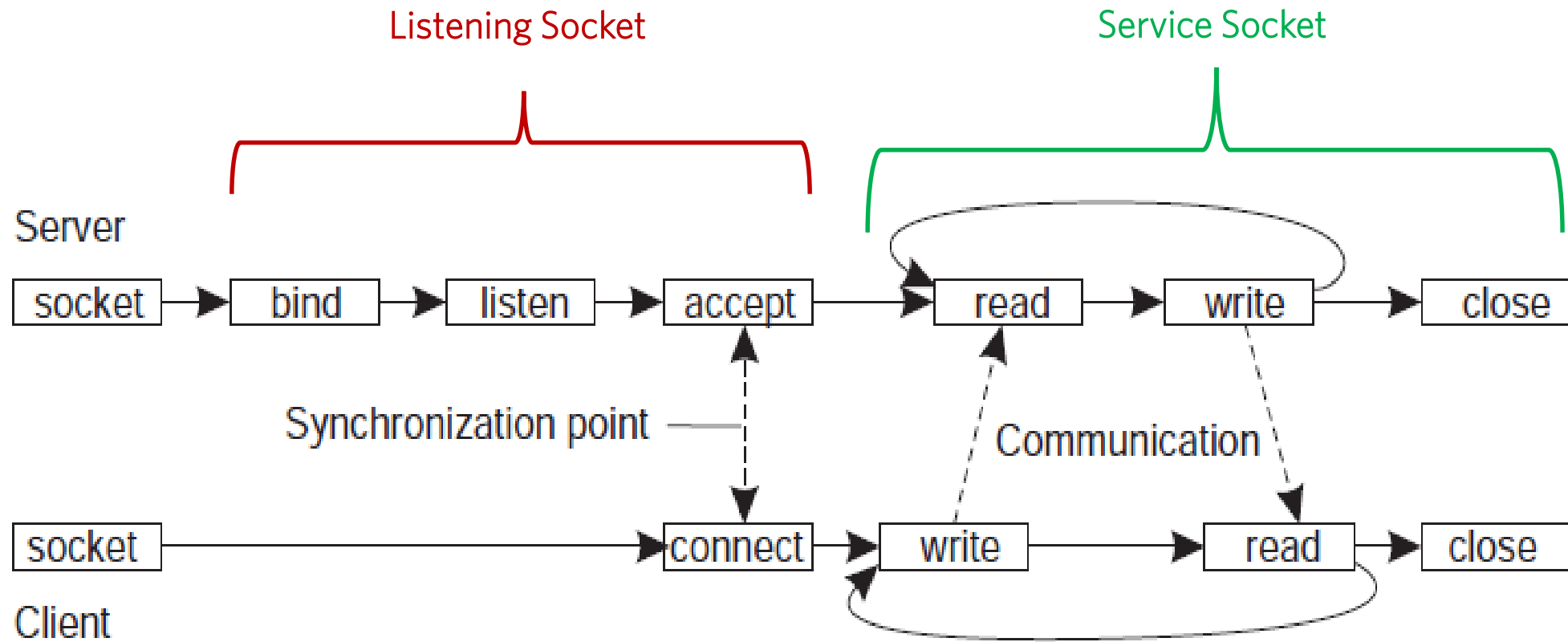


Server A accepts  
request



Client B is now  
connected with  
Server A

# Communication via Sockets



# Socket Communication Recipe

1. Server instantiates a `ServerSocket` object (usually passing a port number). This socket is referred to as the `listening socket`.
2. Server invokes the `accept()` method that awaits incoming client connections.
3. Client instantiates `Socket` object (passing an address to connect to). This socket is referred to as a `client socket`.
4. On the server side, `accept()` returns a new socket referred to as a `service socket` on which the client reads/writes.



# ServerSocket Methods

| SN | Methods with Description  |
|----|---|
| 1  | <b><u>public ServerSocket(int port)</u></b><br>Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application.   |
| 2  | <b><u>public ServerSocket()</u></b><br>Creates an unbound server socket. When using this constructor, use the bind() method when you are ready to bind the server socket.   |
| 3  | <b><u>public Socket accept()</u></b><br>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. |
| 4  | <b><u>public void bind(SocketAddress host)</u></b><br>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.  |
| 5  | <b><u>public SocketAddress getLocalSocketAddress()</u></b><br>Returns the address of the endpoint this socket is bound to, or null if it not bound yet.   |

# Socket Methods

| SN | Methods with Description  |
|----|---|
| 1  | <b><u>public Socket(String host, int port)</u></b><br>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  | <b><u>public Socket()</u></b><br>Creates an unconnected socket. Use the connect() method to connect this socket to a server.  |
| 3  | <b><u>public void connect(SocketAddress host)</u></b><br>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  | <b><u>public InputStream getInputStream()</u></b><br>Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.   |
| 5  | <b><u>public OutputStream getOutputStream()</u></b><br>Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket   |
| 6  | <b><u>public SocketAddress getLocalSocketAddress()</u></b><br>Returns the address of the endpoint this socket is bound to, or null if it is not bound yet.  |
| 7  | <b><u>public void close()</u></b> Closes the socket, which makes this Socket object no longer capable of connecting again to any server   |

# Multi-Threading

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

```
public class Service implements Runnable
```

- STEP 1: Implement the method *run()*

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

- STEP 2: Instantiate a Thread object

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

- STEP 5: Invoke *start()* on the new thread

```
t.start() // invokes the run() method
```

# 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"