SER 321 B Session

SI Session

Sunday, April 6th 2025

7:00 pm - 8:00 pm MST

Agenda

- 3-2 Protocol Tips
- Communication Models
- Sockets!
  - Review Socket Steps
  - Port Examination

SI Session Expectations

Thanks for coming to the **SER 321** SI session. We have a packed agenda and we are going to try to get through as many of our planned example problems as possible. This session will be recorded and shared with others.

- If after this you want to see additional examples, please visit the drop-in tutoring center.
- We will post the link in the chat now and at the end of the session.
  - tutoring.asu.edu
- Please keep in mind we are recording this session and it will be made available for you to review 24-48 hours after this session concludes.
- Finally, please be respectful to each other during the session.

Interact with us:

Zoom Features

Zoom Chat

- Use the chat feature to interact with the presenter and respond to presenter's questions.
- Annotations are encouraged

SER 321
Protocol Tips

3-2 → Custom Protocol!

Stay Organized!

Try to emulate the structure in 3-1

Format your Markdown!

Assign 3-1 Starter Code

SER 321

Protocol Tips

Protocol Structure

```
## Protocol: ##

### Echo: ###

Request:

    {
        "type" : "echo", -- type of request
        "data" : <String>  -- String to be echoed
    }

Error response:

    {
        "type" : "echo",
        "ok" : false,
        "message" : <String> -- what went wrong
    }
```

```
General response:

    {
        "type" : "echo", -- echoes the initial response
        "ok" : <bool>, -- true or false depending on request
        "echo" : <String>,  -- echoed String if ok true
        "message" : <String>,  -- error message if ok false
    }
```

```
Success response:

    {
        "type" : "echo",
        "ok" : true,
        "echo" : <String> -- the echoed string
    }
```

Assign 3-1 Starter Code

SER 321

Protocol Tips

Table of Contents

```
<!-- TOC -->
  * [Protocol:](#protocol-)
    * [Echo:](#echo-)
    * [Add:](#add-)
    * [AddMany:](#addmany-)
    * [Roller:](#roller-)
    * [Inventory:](#inventory-)
    * [General error responses:]
<!-- TOC -->
```

README.md • SockServer.java

Generate

Test Data

Text

Number

Date/Time

Internet

UUID

Custom

Create Link Ctrl+Shift+U

Insert Table

Insert Image Ctrl+U

Generate Table Of Contents

```
m SE, ASU Polytechnic, CIDSE, SE

mple client server implementation. The server
ddmany. Basic error handling on the server side
not have error handling and only has hard coded

and `gradle Client` together.

## Protocol: ##
```

Assign 3-1 Starter Code

SER 321

Protocol Tips

Formatting

```
<!-- TOC -->
  * [Protocol:](#protocol-)
    * [Echo:](#echo-)
    * [Add:](#add-)
    * [AddMany:](#addmany-)
    * [Roller:](#roller-)
    * [Inventory:](#inventory-)
    * [General error responses:]
<!-- TOC -->
```

```
## Protocol: ##

### Echo: ###

Request:

    {
        "type" : "echo", -- type of request
        "data" : <String>  -- String to be echoed
    }

####General response:####
```

Protocol:

Echo:

Request:

```
{
    "type" : "echo", -- ty
    "data" : <String>  --
}
```

####General response:####

Spaces are important!

Assign 3-1 Starter Code

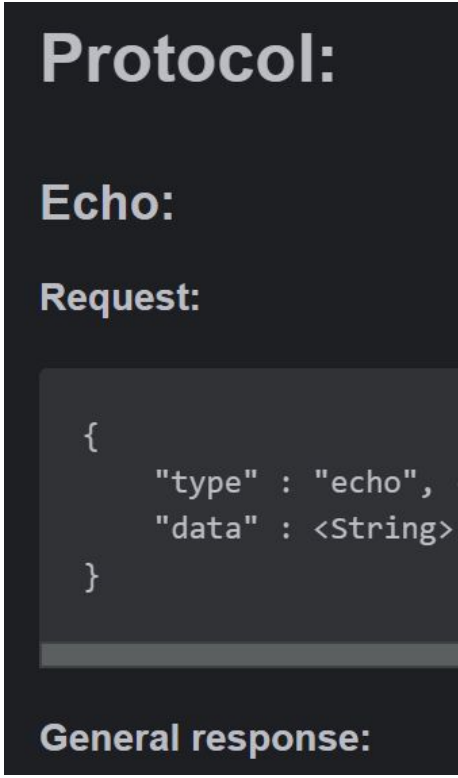
SER 321
Protocol Tips

Formatting

```
<!-- TOC -->
  * [Protocol:](#protocol-)
    * [Echo:](#echo-)
      * [Request:](#request-)
      * [General response:](#gen
    * [Add:](#add-)
    * [AddMany:](#addmany-)
    * [Roller:](#roller-)
    * [Inventory:](#inventory-)
    * [General error responses:]
<!-- TOC -->
```

```
## Protocol: ##

### Echo: ###

#### Request: ####

    {
        "type" : "echo", -- type of request
        "data" : <String>  -- String to be echoed
    }

#### General response: ####
```

Number of #s indicate the header level



org.json Docs

JSON Guide

SER 321

JSON Structure

Data is stored in…

Name:Value pairs → AKA → Members

"Katie"

"student" : "Katie"

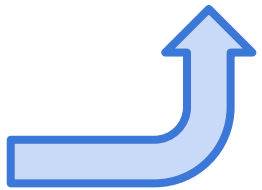
What uses curly braces?

Objects

{ }

What do Objects contain?

Members



org.json Docs

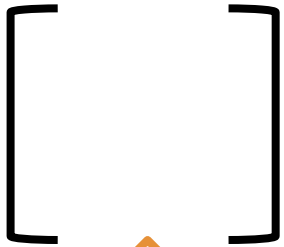
JSON Guide

SER 321

JSON Structure

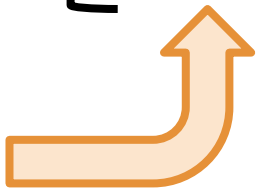
What uses brackets?

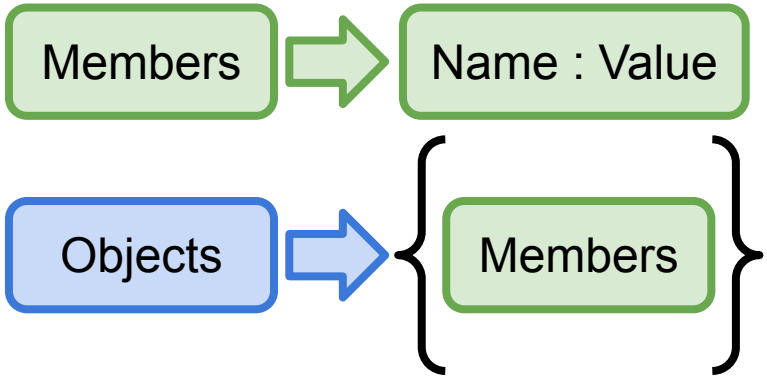
Arrays



What do Arrays contain?

Any ***Valid*** Value





org.json Docs

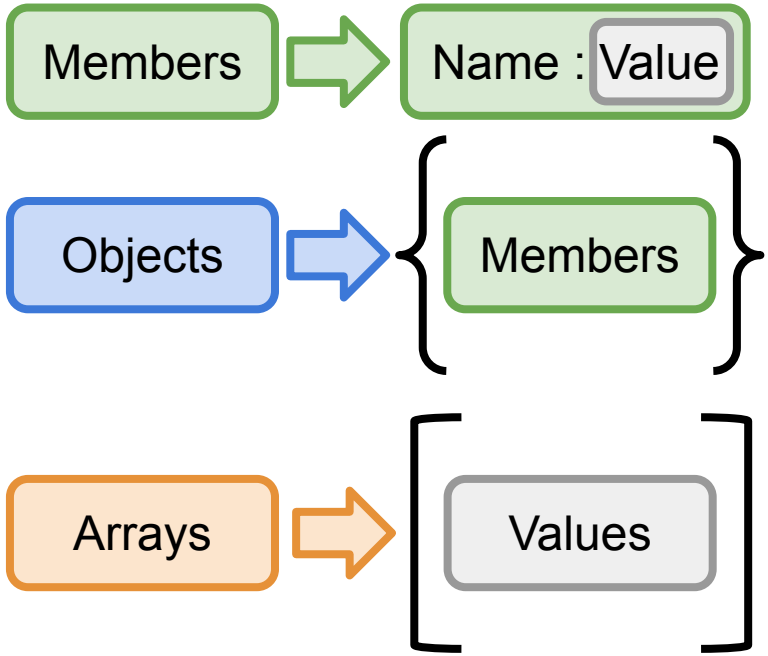
JSON Guide

SER 321

JSON Structure

What is a valid value?

- Strings
- Booleans
- Numbers
- NULL
- Objects
- Arrays



SER 321

Socket Protocol Types

Two Main Conversation Models

1. Client Request

2. Server Response

Pull/Polling Model

Client *pulls* info from the server

Client *polls* the server for info

SER 321

Socket Protocol Types

Two Main Conversation Models

1. Server sends update

2. Client acknowledges

Push Model

Server *pushes* info to client

Push notifications

SER 321

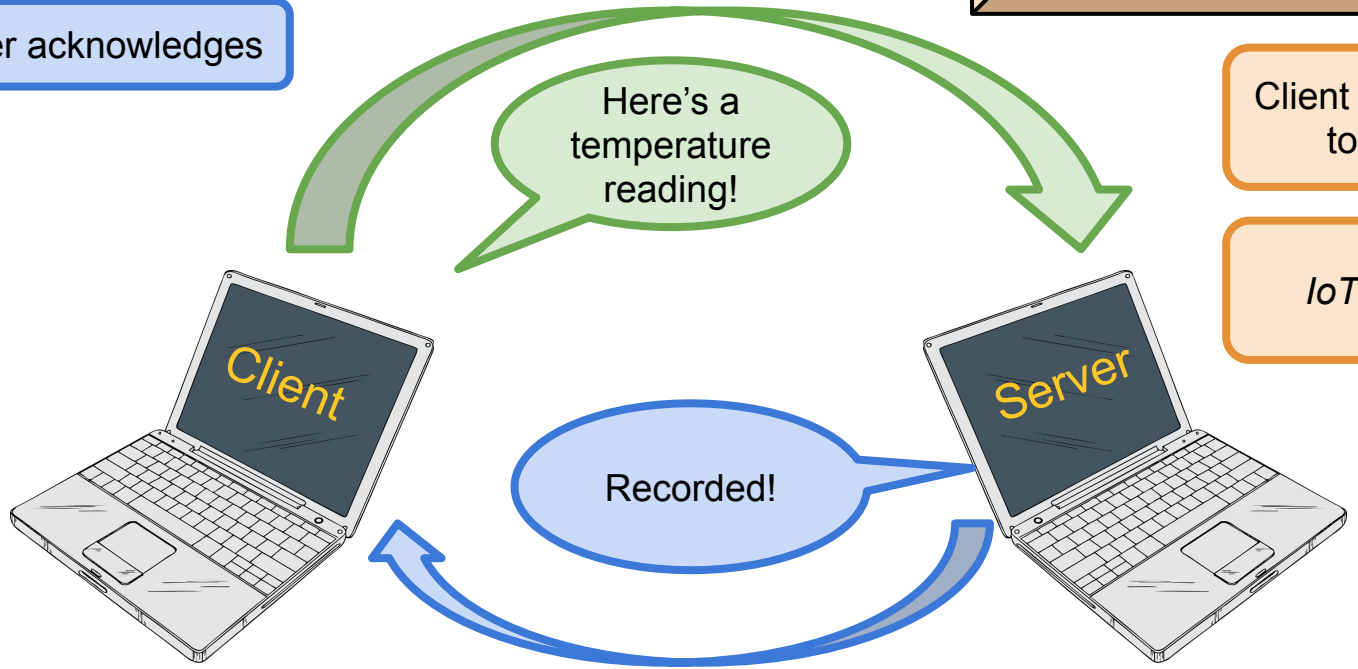
Socket Protocol Types

Two Main Conversation Models

Client Push Model

1. Client sends update

2. Server acknowledges



Client *pushes* info to Server

IoT sensors

SER 321
Client Socket

Put the Steps for the **Client Socket** in the correct order:

1.
2.
3.
4.
5.
6.
7.
8.

A. Send Message
B. Close Socket
C. Define Params
D. Create Param Struct
E. Receive Message
F. Create Socket
G. Repeat
H. Establish Connection

SER 321
Client Socket

Put the Steps for the **Client Socket** in the correct order:

C	1. Define Params
F	2. Create Socket
D	3. **C ONLY** Create a struct for the address
H	4. Establish Connection
A	5. Send Message
E	6. Receive Message
G	7. Repeat #5 and #6 as needed
B	8. Close Socket

A. Send Message
B. Close Socket
C. Define Params
D. Create Param Struct
E. Receive Message
F. Create Socket
G. Repeat
H. Establish Connection

SER 321

Server Socket

Put the Steps for the **Server Socket** in the correct order:

1.

2.

3.

4.

5.

6.

7.

8.

9.

A. Mark Socket to Listen

B. Close Socket

C. Define Params

D. Create Param Struct

E. Continue Listening

F. Handle Client

G. Wait for Connection

H. Bind Socket to Address

I. Create Socket

SER 321

Server Socket

Put the Steps for the **Server Socket** in the correct order:

F	1. Define Params
I	2. Create Socket
E	3. **C ONLY** Create a struct for the address
H	4. Bind Socket to Local Address
A	5. Mark Socket to Listen for Connections
C	6. Wait for Connection
B	7. Handle Client Connection
G	8. Close Client Connection
D	9. Continue Listening for Connections

A. Mark Socket to Listen
B. Handle Client
C. Wait for Connection
D. Continue Listening
E. Create Param Struct
F. Define Params
G. Close Socket
H. Bind Socket to Address
I. Create Socket

JavaSimpleSock

SER 321

Single Threaded Server

What will happen if there are two clients?

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets
\JavaSimpleSock> gradle socketServer

> Task :SocketServer
Server ready for a connection
Server waiting for a connection
<==========----> 75% EXECUTING [20s]
> :SocketServer
```

Server

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sock
ets\JavaSimpleSock> gradle socketClient

> Task :SocketClient
Please enter a String to send to the Server (enter
 "exit" to quit"):
<==========----> 75% EXECUTING [14s]
> :SocketClient
```

Client 1

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sock
ets\JavaSimpleSock> gradle socketClient
```

Client 2

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets
\JavaSimpleSock> gradle socketServer

> Task :SocketServer
Server ready for a connection
Server waiting for a connection
<==========----> 75% EXECUTING [53s]
> :SocketServer
```

Server

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sock
ets\JavaSimpleSock> gradle socketClient

> Task :SocketClient
Please enter a String to send to the Server (enter
 "exit" to quit"):
<<===<<==========----> 75% EXECUTING [47s]
> :SocketClient
Hello!
```

Client 1

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sock
ets\JavaSimpleSock> gradle socketClient
Starting a Gradle Daemon, 2 busy and 4 stopped Dae
mons could not be reused, use --status for details
<==========----> 75% EXECUTING [15s]
> :SocketClient
```

Client 2

JavaSimpleSock

SER 321

Single Threaded Server

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets
\JavaSimpleSock> gradle socketServer

> Task :SocketServer
Server ready for a connection
Server waiting for a connection
Received the String Hello!
Received the Integer 9
<==========----> 75% EXECUTING [1m 27s]
> :SocketServer
```

Server

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sock
ets\JavaSimpleSock> gradle socketClient

> Task :SocketClient
Please enter a String to send to the Server (enter
 "exit" to quit"):
<<===<<==========----> 75% EXECUTING [59s]        P
lease enter a Number to send to the Server (enter
0 to quit"):
<<==========----> 75% EXECUTING [1m 18s]          9
 and Hello! ... Got it!
Please enter a String to send to the Server (enter
 "exit" to quit"):
<==========----> 75% EXECUTING [1m 21s]
> :SocketClient
```

Client 1

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sock
ets\JavaSimpleSock> gradle socketClient
Starting a Gradle Daemon, 2 busy and 4 stopped Dae
mons could not be reused, use --status for details
<==========----> 75% EXECUTING [49s]
> :SocketClient
```

Client 2

JavaSimpleSock

SER 321

Single Threaded Server

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets
\JavaSimpleSock> gradle socketServer

> Task :SocketServer
Server ready for a connection
Server waiting for a connection
Received the String Hello!
Received the Integer 9
<==========----> 75% EXECUTING [1m 55s]
> :SocketServer
```

Server

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sock
ets\JavaSimpleSock> gradle socketClient

> Task :SocketClient
Please enter a String to send to the Server (enter
 "exit" to quit"):
<<===<<==========----> 75% EXECUTING [59s]        P
lease enter a Number to send to the Server (enter
0 to quit"):
<<==========----> 75% EXECUTING [1m 18s]          9
 and Hello! ... Got it!
Please enter a String to send to the Server (enter
 "exit" to quit"):
<===<==========----> 75% EXECUTING [1m 49s]
> :SocketClient
exit
```

Client 1

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sock
ets\JavaSimpleSock> gradle socketClient
Starting a Gradle Daemon, 2 busy and 4 stopped Dae
mons could not be reused, use --status for details
<==========----> 75% EXECUTING [1m 18s]
> :SocketClient
```

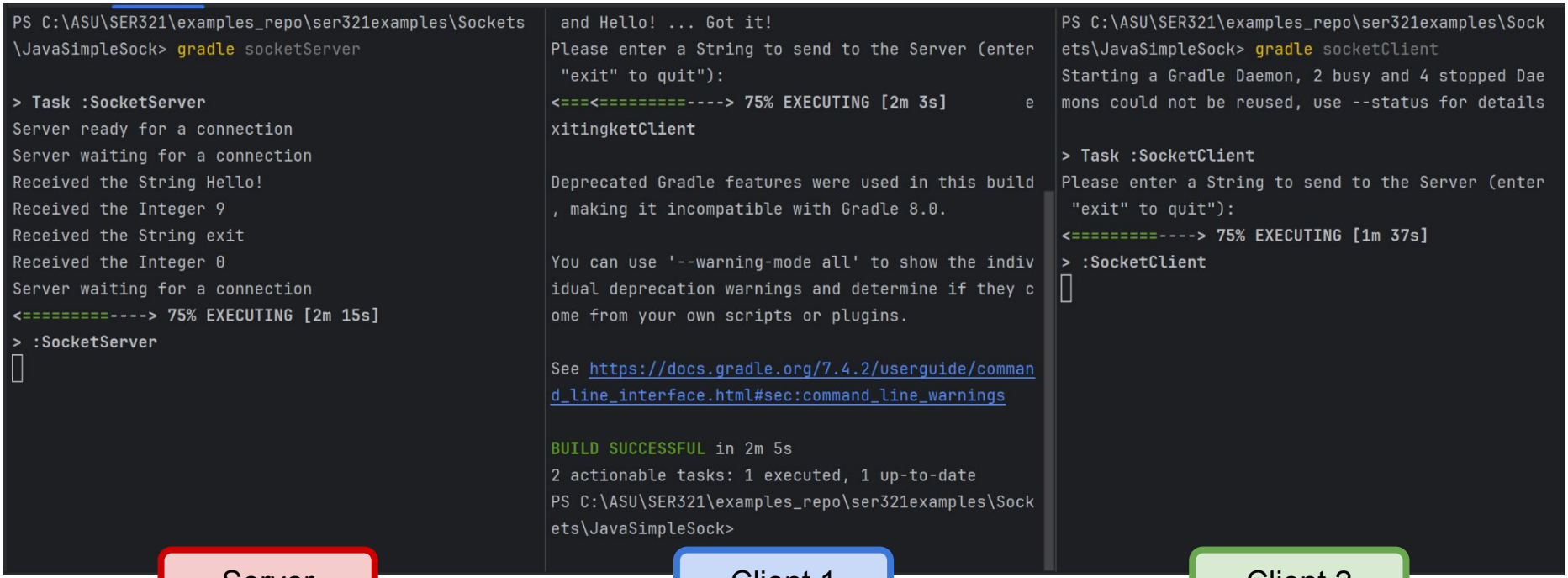
Client 2

What do we think will happen?

JavaSimpleSock

SER 321

Single Threaded Server



Server

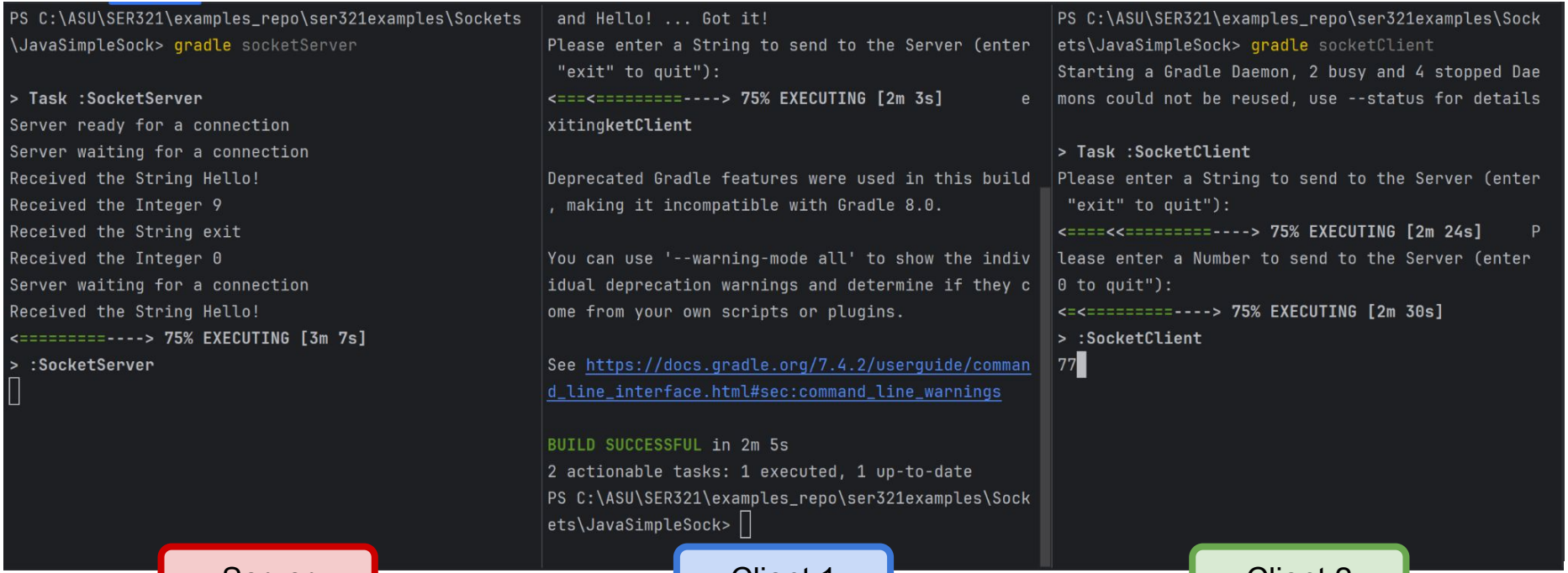
Client 1

Client 2

JavaSimpleSock

SER 321

Single Threaded Server



JavaSimpleSock

SER 321

Single Threaded Server

Why?


```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets
\JavaSimpleSock> gradle socketServer

> Task :SocketServer
Server ready for a connection
Server waiting for a connection
Received the String Hello!
Received the Integer 9
Received the String exit
Received the Integer 0
Server waiting for a connection
Received the String Hello!
<==========----> 75% EXECUTING [3m 7s]
> :SocketServer
```

1. Define Params
2. Create Socket
3-5. Mark Socket to Listen
6. Wait for Connection
7. Handle Client Connection
8. Close Client Connection
9.

```
ets\JavaSimpleSock>
```

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sock
ets\JavaSimpleSock> gradle socketClient
Starting a Gradle Daemon, 2 busy and 4 stopped Dae
mons could not be reused, use --status for details

> Task :SocketClient
Please enter a String to send to the Server (enter
 "exit" to quit"):
<====<<==========----> 75% EXECUTING [2m 24s]     P
lease enter a Number to send to the Server (enter
0 to quit"):
<=<==========----> 75% EXECUTING [2m 30s]
> :SocketClient
77
```

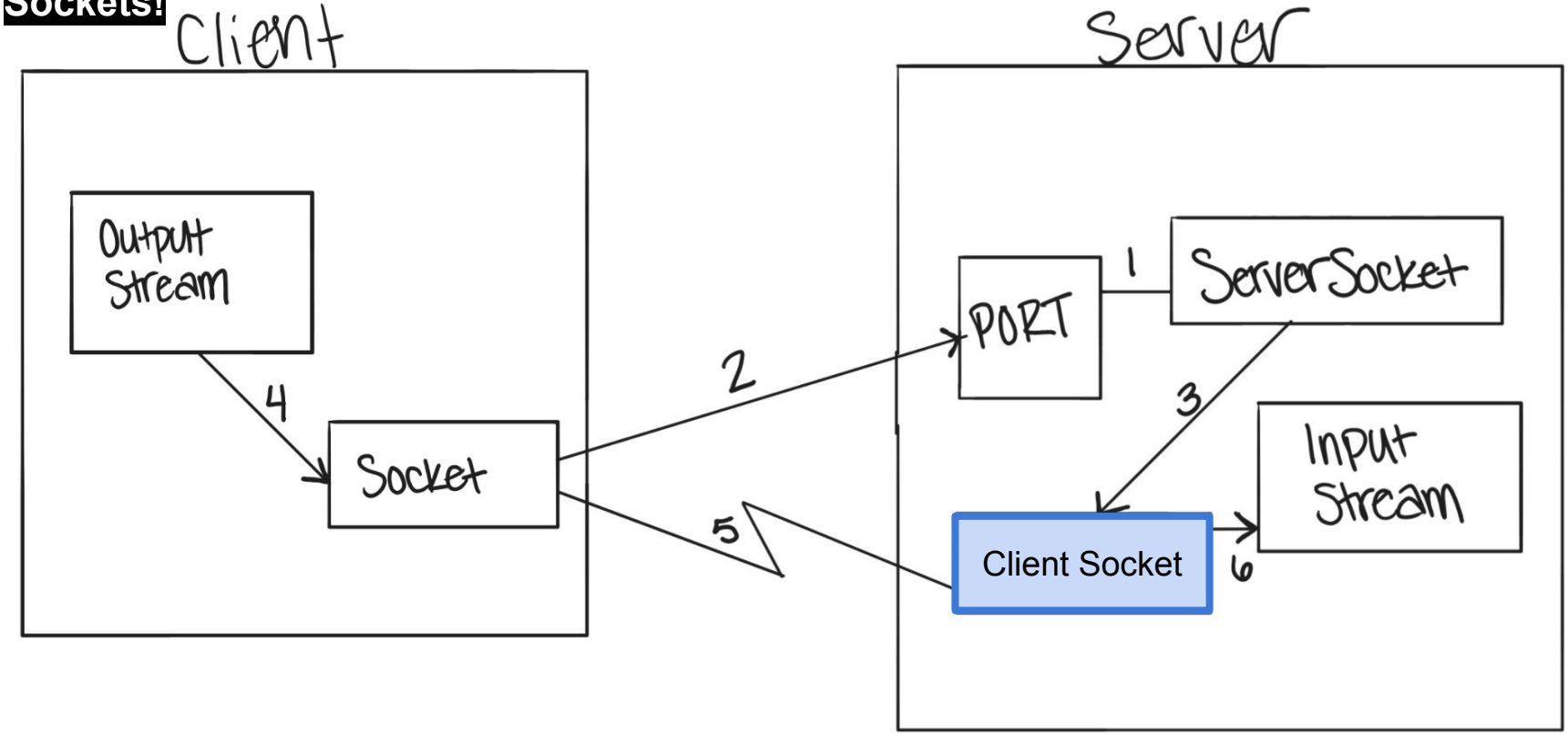
Server

Client 1

Client 2

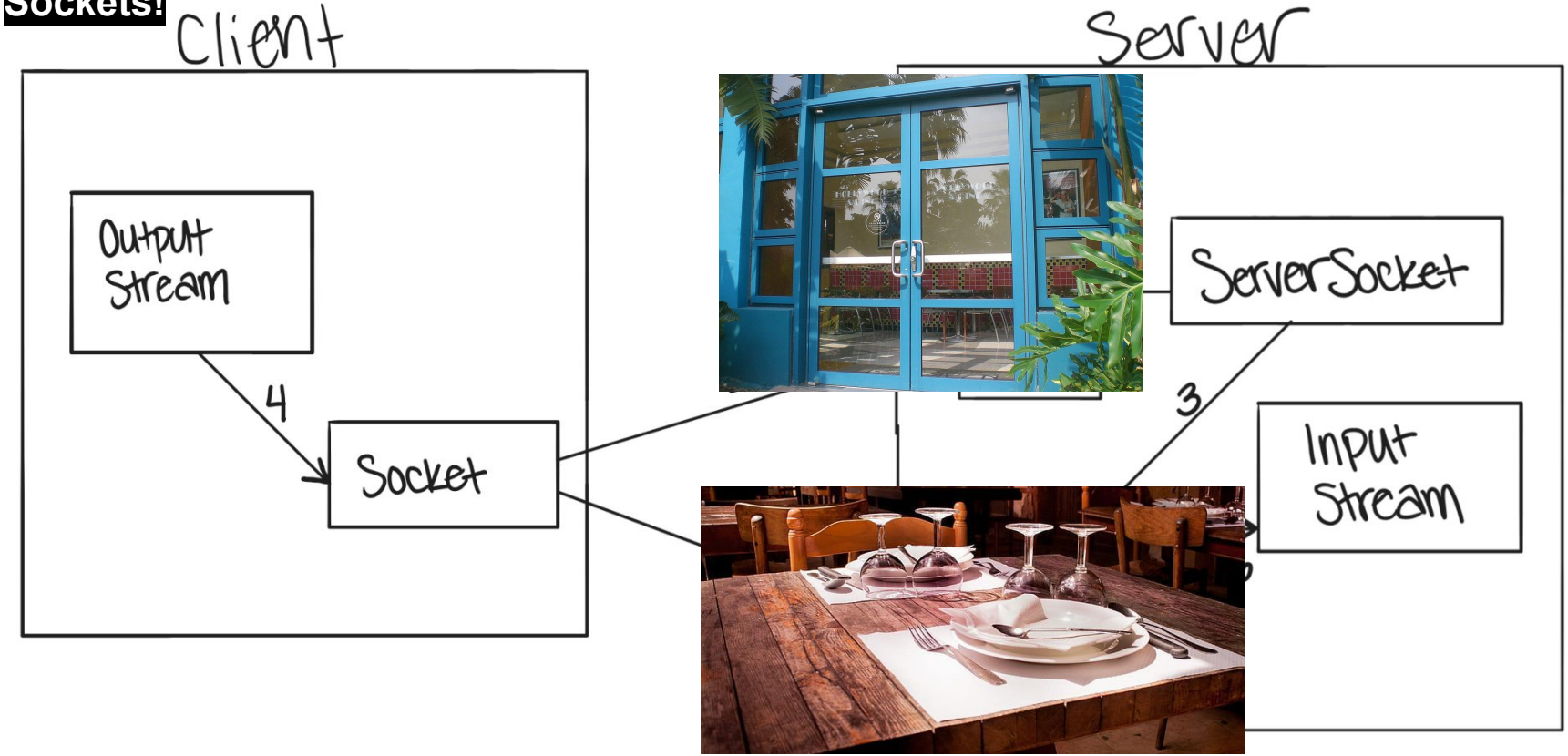
SER 321

Sockets!



SER 321

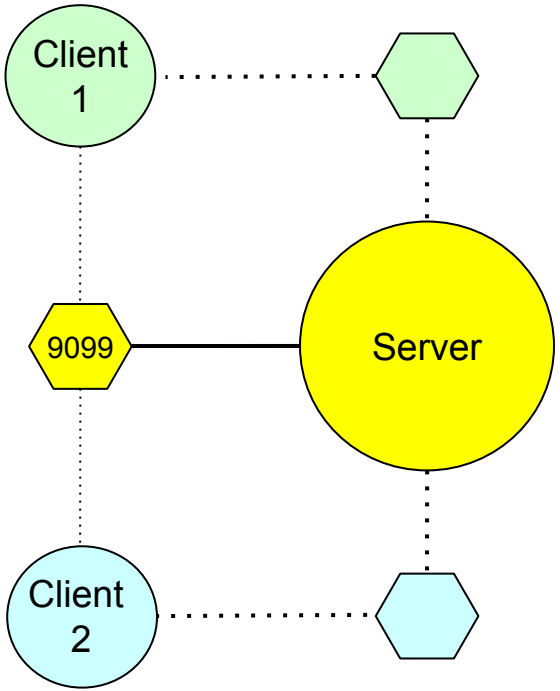
Sockets!

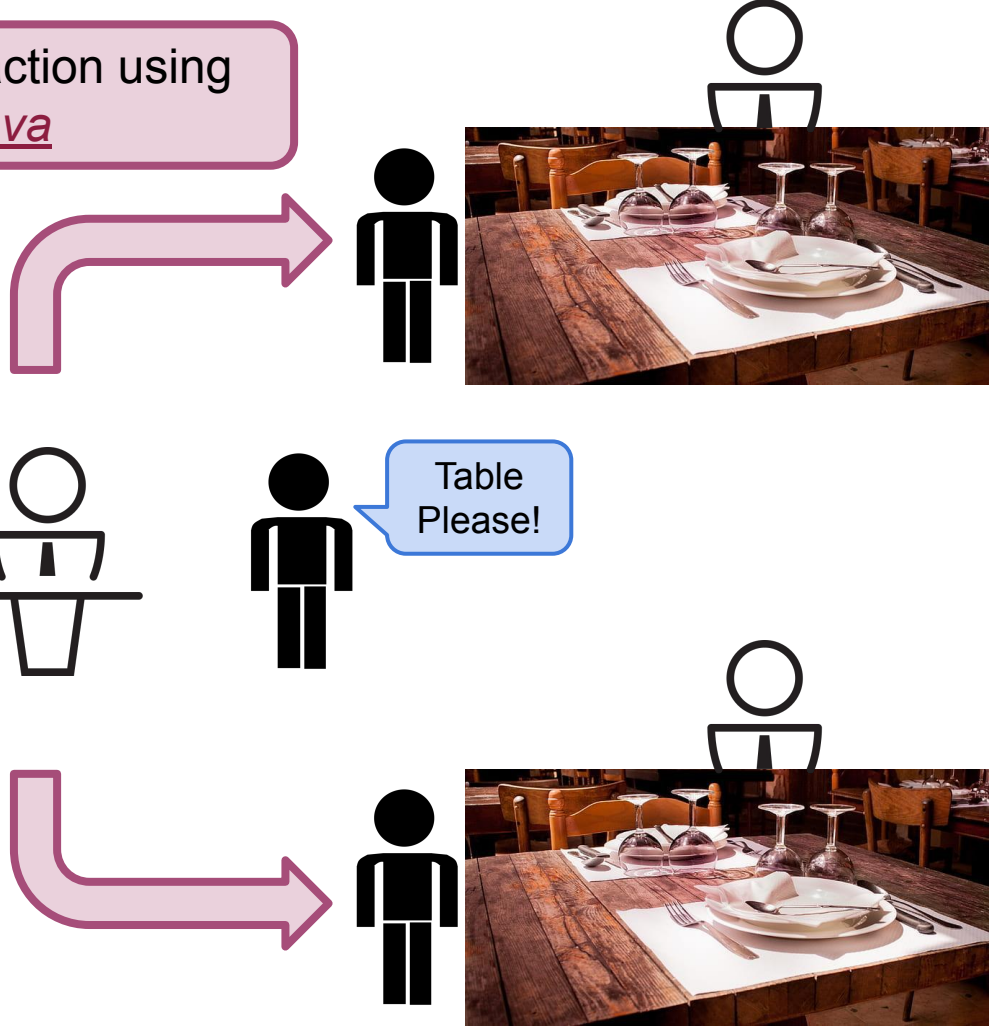


SER 321

Port Examination

Let's see this in action using *Echo_Java*





SER 321

Sockets!

Original

Sockets/Echo_Java

```java
try {
        if (args.length != 1) {
                System.out.println("Usage: gradle runServer -Pport=9099");
                System.exit( status: 0);
        }
        int port = -1;
        try {
                port = Integer.parseInt(args[0]);
        } catch (NumberFormatException nfe) {
                System.out.println("[Port] must be an integer");
                System.exit( status: 2);
        }
        Socket clientSock;
        ServerSocket sock = new ServerSocket(port);
        System.out.println("Server ready for connections");

        int bufLen = 1024;
        byte clientInput[] = new byte[bufLen]; // up to 1024 bytes in a message.
        while(true) {
                System.out.println("Server waiting for a connection");
                clientSock = sock.accept(); // blocking wait
                PrintWriter out = new PrintWriter(clientSock.getOutputStream(), autoFlush: true);
                InputStream input = clientSock.getInputStream();
                System.out.println("Server connected to client");
                int numr = input.read(clientInput, off: 0, bufLen);
                while (numr != -1) {
                  String received = new String(clientInput, offset: 0, numr);
                  System.out.println("read from client: " + received);
                  out.println(received);
                  numr = input.read(clientInput, off: 0, bufLen);
                }
```

Client

```java
String host = args[0];
Socket server = new Socket(host, port);
System.out.println("Connected to server at " + host + ":" + port);
InputStream input = server.getInputStream();
OutputStream output = server.getOutputStream();
BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
```

SER 321
Sockets!

Modification

Sockets/Echo_Java

```java
try {
        if (args.length != 1) {...}
        int port = -1;
        try {
                port = Integer.parseInt(args[0]);
        } catch (NumberFormatException nfe) {
                System.out.println("[Port] must be an integer");
                System.exit( status: 2);
        }
        Socket clientSock;
        ServerSocket sock = new ServerSocket(port);
        System.out.println("Server ready for connections");
        System.out.println("Server is listening on port: " + port);
        System.out.println("-----");
        System.out.println("Values of the ServerSocket Object:");
        System.out.println("Inet Address: " + sock.getInetAddress());
        System.out.println("Local Port: " + sock.getLocalPort());

        int bufLen = 1024;
        byte clientInput[] = new byte[bufLen]; // up to 1024 bytes in a message.
        while(true) {
                System.out.println("Server waiting for a connection");
                clientSock = sock.accept();             // blocking wait

                PrintWriter out = new PrintWriter(clientSock.getOutputStream(), autoFlush: true);
                InputStream input = clientSock.getInputStream();
                System.out.println("Server connected to client");
                System.out.println("-----");
                System.out.println("Values of the Client Socket Object after Connection:");
                System.out.println("\tInet Address: " + clientSock.getInetAddress());
                System.out.println("\tLocal Address: " + clientSock.getLocalAddress());
                System.out.println("\tLocal Port: " + clientSock.getLocalPort());
                System.out.println("\tAllocated Client Socket (Port): " + clientSock.getPort());

                int numr = input.read(clientInput, off: 0, bufLen);
```

Client

```java
String host = args[0];
Socket server = new Socket(host, port);
System.out.println("Connected to server at " + host + ":" + port);
System.out.println("Values of the Socket Object for the Server:");
System.out.println("\tHost: " + server.getLocalAddress());
System.out.println("\tPort: " + server.getPort());
System.out.println("\tLocal Port: " + server.getLocalPort());
InputStream input = server.getInputStream();
OutputStream output = server.getOutputStream();
BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
```

SER 321

Sockets!

Sockets/Echo_Java

Server

```
> Task :runServer
Server ready for connections
Server is listening on port: 9099
-----
Values of the ServerSocket Object:
Inet Address: 0.0.0.0/0.0.0.0
Local Port: 9099
Server waiting for a connection
<==========----> 75% EXECUTING [10s]
> :runServer
```

```
try {
        if (args.length != 1) {...}
        int port = -1;
        try {
                port = Integer.parseInt(args[0]);
        } catch (NumberFormatException nfe) {
                System.out.println("[Port] must be an integer");
                System.exit( status: 2);
        }
        Socket clientSock;
        ServerSocket sock = new ServerSocket(port);
        System.out.println("Server ready for connections");
        System.out.println("Server is listening on port: " + port);
        System.out.println("-----");
        System.out.println("Values of the ServerSocket Object:");
        System.out.println("Inet Address: " + sock.getInetAddress());
        System.out.println("Local Port: " + sock.getLocalPort());

        int bufLen = 1024;
        byte clientInput[] = new byte[bufLen]; // up to 1024 bytes in a message.
        while(true) {
                System.out.println("Server waiting for a connection");
                clientSock = sock.accept();             // blocking wait

                PrintWriter out = new PrintWriter(clientSock.getOutputStream(), autoFlush: true);
                InputStream input = clientSock.getInputStream();
                System.out.println("Server connected to client");
                System.out.println("-----");
                System.out.println("Values of the Client Socket Object after Connection:");
                System.out.println("\tInet Address: " + clientSock.getInetAddress());
                System.out.println("\tLocal Address: " + clientSock.getLocalAddress());
                System.out.println("\tLocal Port: " + clientSock.getLocalPort());
                System.out.println("\tAllocated Client Socket (Port): " + clientSock.getPort());

                int numr = input.read(clientInput, off: 0, bufLen);
```

Client

```
Socket server = new Socket(host, port);
System.out.println("Connected to server at " + host + ":" + port);
System.out.println("Values of the Socket Object for the Server:");
System.out.println("\tHost: " + server.getLocalAddress());
System.out.println("\tPort: " + server.getPort());
System.out.println("\tLocal Port: " + server.getLocalPort());
InputStream input = server.getInputStream();
OutputStream output = server.getOutputStream();
BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
```

SER 321

Sockets!

Sockets/Echo_Java

Server

```
> Task :runServer
Server ready for connections
Server is listening on port: 9099
-----
Values of the ServerSocket Object:
Inet Address: 0.0.0.0/0.0.0.0
Local Port: 9099
Server waiting for a connection
Server connected to client
-----
Values of the Client Socket Object after Connection:
        Inet Address: /127.0.0.1
        Local Address: /127.0.0.1
        Local Port: 9099
        Allocated Client Socket (Port): 60296
<==========----> 75% EXECUTING [1m 13s]
> :runServer
```

Client

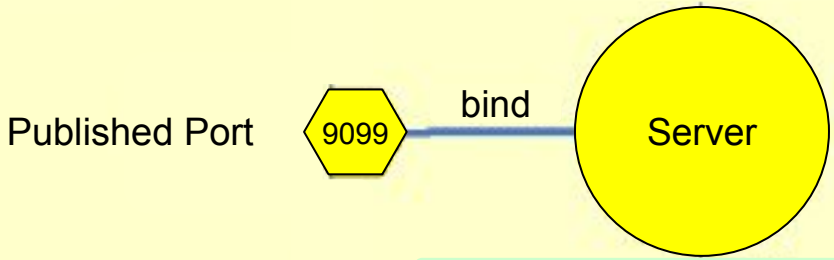
```
> Task :runClient
Connected to server at localhost:9099
Values of the Socket Object for the Server:
        Host: /127.0.0.1
        Port: 9099
        Local Port: 60296
String to send>
<==========----> 75% EXECUTING [31s]
> :runClient
```

```
try {
        if (args.length != 1) {...}
        int port = -1;
        try {

        } catch

        }
        Socket
        ServerS
        System.
        System.
        System.
        System.
        System.
        System.

        int buf
        byte cl
         hile(t
                                                    It
                                                    ));
```

```
System.out.println("Server waiting for a connection");
clientSock = sock.accept();                // blocking wait

PrintWriter out = new PrintWriter(clientSock.getOutputStream(), autoFlush: true);
InputStream input = clientSock.getInputStream();
System.out.println("Server connected to client");
System.out.println("-----");
System.out.println("Values of the Client Socket Object after Connection:");
System.out.println("\tInet Address: " + clientSock.getInetAddress());
System.out.println("\tLocal Address: " + clientSock.getLocalAddress());
System.out.println("\tLocal Port: " + clientSock.getLocalPort());
System.out.println("\tAllocated Client Socket (Port): " + clientSock.getPort());

int numr = input.read(clientInput, off: 0, bufLen);
```

SER 321
Sockets!

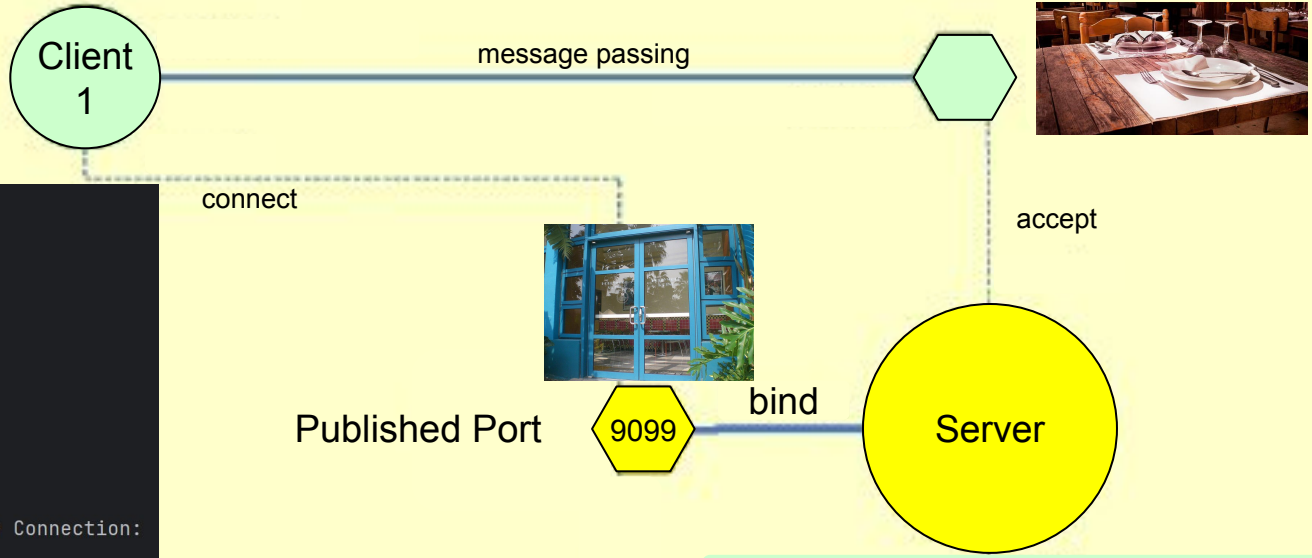
```
> Task :runServer
Server ready for connections
Server is listening on port: 9099
-----
Values of the ServerSocket Object:
Inet Address: 0.0.0.0/0.0.0.0
Local Port: 9099
Server waiting for a connection
Server connected to client
-----
Values of the Client Socket Object after Connection:
        Inet Address: /127.0.0.1
        Local Address: /127.0.0.1
        Local Port: 9099
        Allocated Client Socket (Port): 60296
<==========----> 75% EXECUTING [2m 36s]
> :runServer
```




```
> Task :runClient
Connected to server at localhost:9099
Values of the Socket Object for the Server:
        Host: /127.0.0.1
        Port: 9099
        Local Port: 60296
String to send>
<==========----> 75% EXECUTING [2m 18s]s]
> :runClient
```

Design of an RFID Vehicle Authentication System: A Case Study for Al-Nahrain University Campus - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Client-and-Server-Socket-Ports_fig4_282671198

SER 321
Sockets!




```
> Task :runServer
Server ready for connections
Server is listening on port: 9099
-----
Values of the ServerSocket Object:
Inet Address: 0.0.0.0/0.0.0.0
Local Port: 9099
Server waiting for a connection
Server connected to client
-----
Values of the Client Socket Object after Connection:
        Inet Address: /127.0.0.1
        Local Address: /127.0.0.1
        Local Port: 9099
        Allocated Client Socket (Port): 60296
<==========----> 75% EXECUTING [2m 36s]
> :runServer
```

```
> Task :runClient
Connected to server at localhost:9099
Values of the Socket Object for the Server:
        Host: /127.0.0.1
        Port: 9099
        Local Port: 60296
String to send>
<==========----> 75% EXECUTING [2m 18s]s]
> :runClient
```

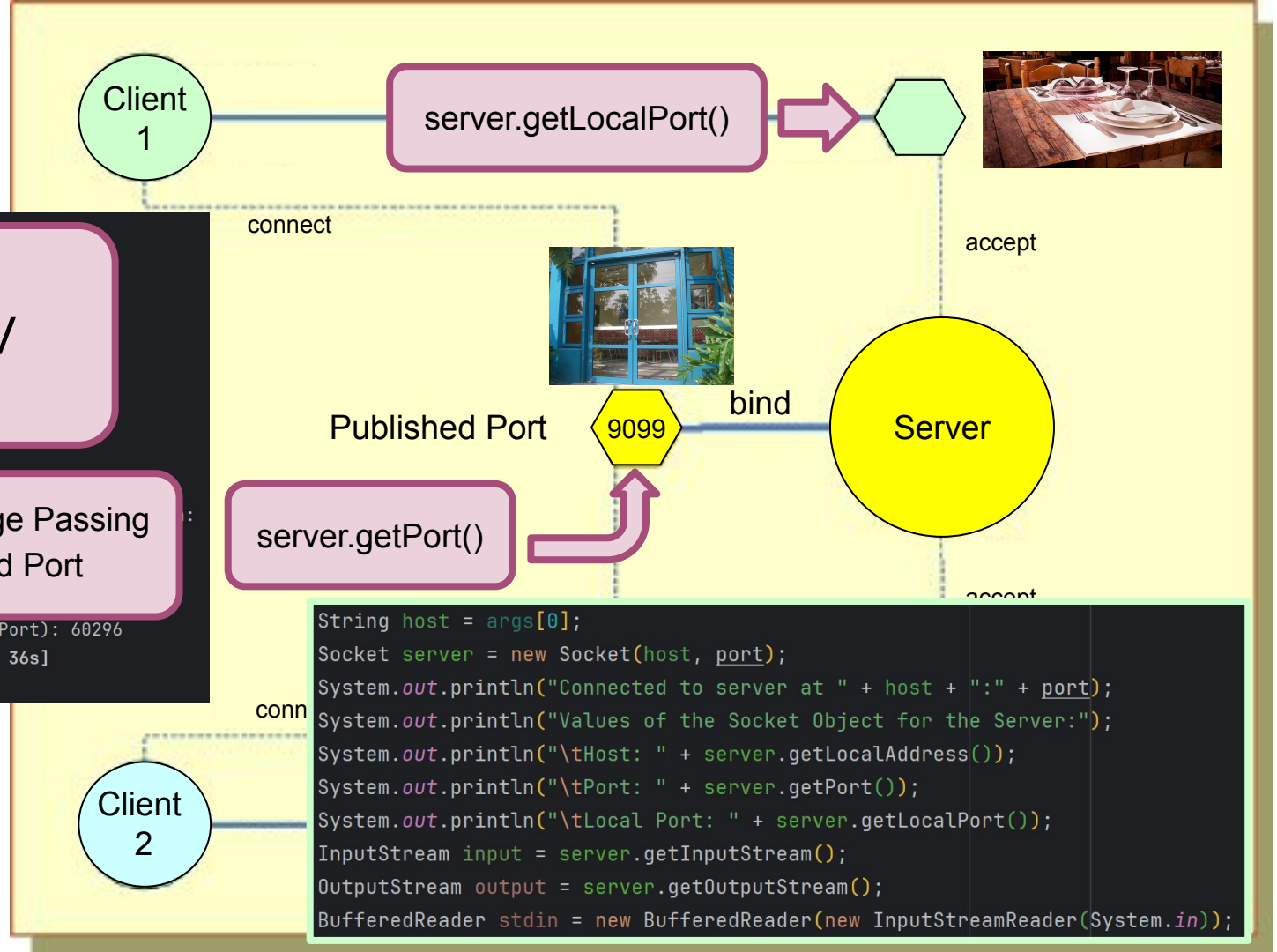
Design of an RFID Vehicle Authentication System: A Case Study for Al-Nahrain University Campus - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Client-and-Server-Socket-Ports_fig4_282671198

SER 321
Sockets!

Client POV

Local Port → Message Passing
Port → Published Port

```
Server connected to client
Allocated Client Socket (Port): 60296
<==========----> 75% EXECUTING [2m 36s]
> :runServer
```




```
String host = args[0];
Socket server = new Socket(host, port);
System.out.println("Connected to server at " + host + ":" + port);
System.out.println("Values of the Socket Object for the Server:");
System.out.println("\tHost: " + server.getLocalAddress());
System.out.println("\tPort: " + server.getPort());
System.out.println("\tLocal Port: " + server.getLocalPort());
InputStream input = server.getInputStream();
OutputStream output = server.getOutputStream();
BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
```

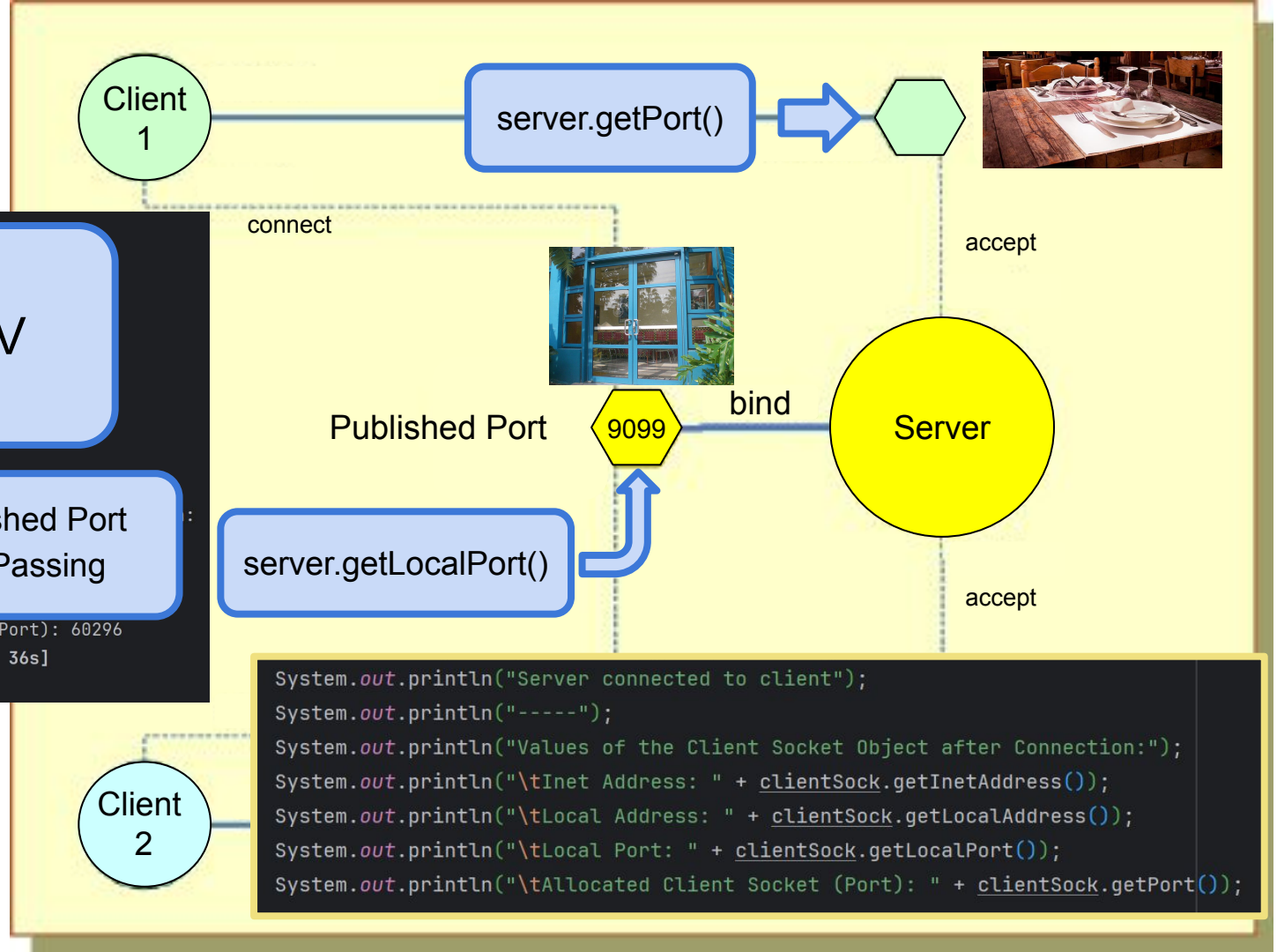
Design of an RFID Vehicle Authentication System: A Case Study for Al-Nahrain University Campus - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Client-and-Server-Socket-Ports_fig4_282671198

SER 321
Sockets!

Server POV

Local Port → Published Port
Port → Message Passing

```
Server connected to client
        Allocated Client Socket (Port): 60296
<==========----> 75% EXECUTING [2m 36s]
> :runServer
```

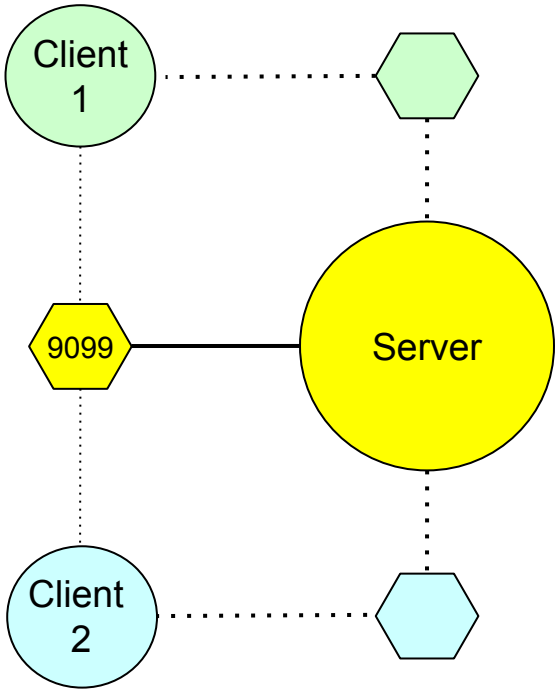


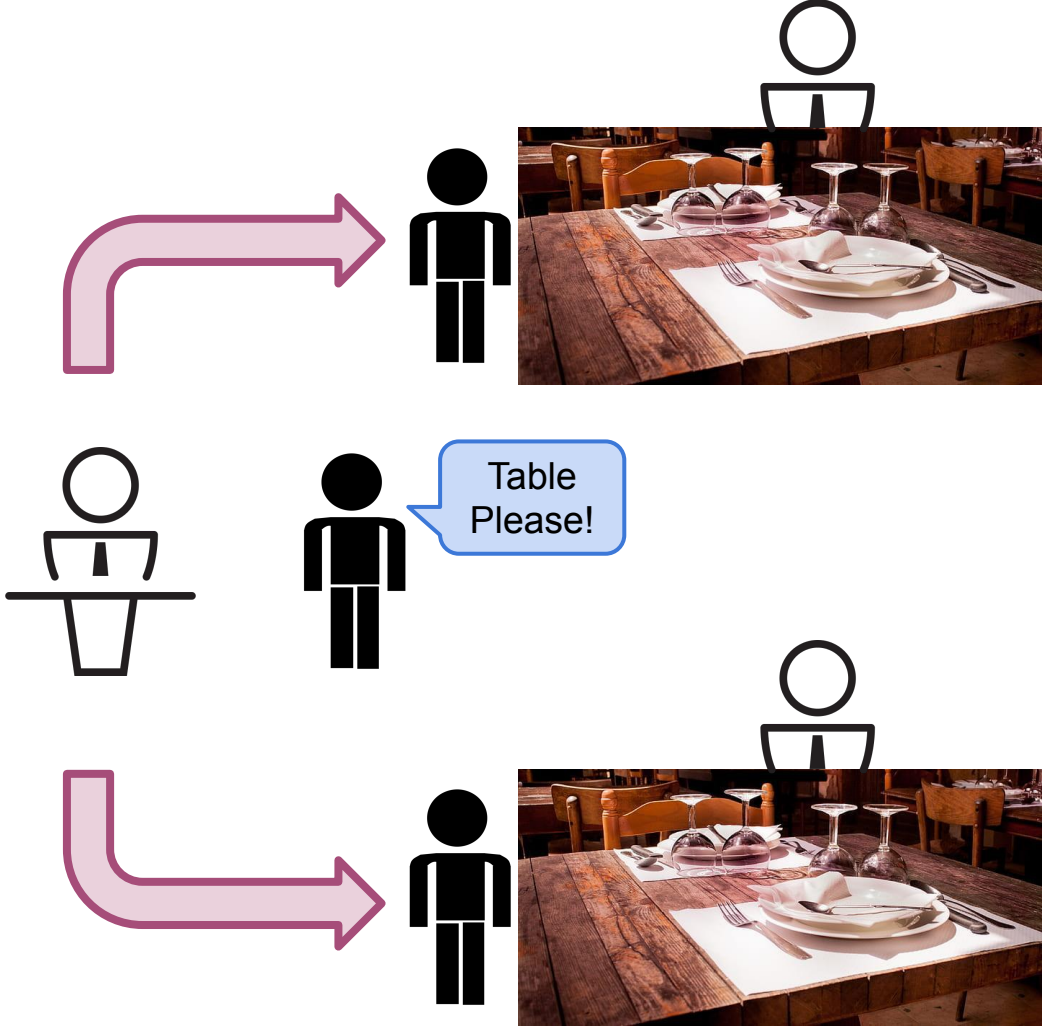

```
System.out.println("Server connected to client");
System.out.println("-----");
System.out.println("Values of the Client Socket Object after Connection:");
System.out.println("\tInet Address: " + clientSock.getInetAddress());
System.out.println("\tLocal Address: " + clientSock.getLocalAddress());
System.out.println("\tLocal Port: " + clientSock.getLocalPort());
System.out.println("\tAllocated Client Socket (Port): " + clientSock.getPort());
```

Design of an RFID Vehicle Authentication System: A Case Study for Al-Nahrain University Campus - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Client-and-Server-Socket-Ports_fig4_282671198

SER 321

Port Examination





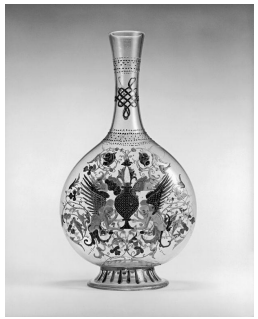
SER 321
Serialization

What is *serialization*?

"Translating data structures or object states for storage or transmission"

SER 321
Serialization

What is *serialization*?



Data

"Translating data structures or object states for storage or transmission"

SER 321
Serialization

What is *serialization*?

Serialized Data

"Translating data structures or object states for storage or transmission"

SER 321
Serialization

Can we recall some of the formats?

JSON

Java Object Serialization

Protocol Buffers

XML

SER 321
Serialization

Binary

Text

Two main approaches for storing the content…

What about the data format?

JSON

Java Object Serialization

Protocol Buffers

XML

SER 321
Serialization

Binary

Text

Who uses *TEXT*?

Text

JSON

Java Object Serialization

Protocol Buffers

Text

XML

SER 321
Serialization

Binary

Text

What does this imply?

Who uses ***BINARY***?

Text JSON

Binary Java Object Serialization

Binary Protocol Buffers

Text XML

SER 321
Serialization

Streams and their types

Generic Superclass

```
OutputStream out = sock.getOutputStream();
```

Buffered Stream

Bytes

Data Stream

Primitive DATA Types

Object Stream

Java Objects

SER 321

Scratch Space

Upcoming Events

SI Sessions:

- Sunday, April 6th at 7:00 pm MST
- Tuesday, April 8th at 10:00 am MST
- Thursday, April 10th at 7:00 pm MST

Review Sessions:

- Sunday, April 27th at **6:00 pm** MST - **2 hour Exam Review Session**
- Tuesday, April 29th, at 10:00 am MST - **Q&A Session**

Questions?

Survey:

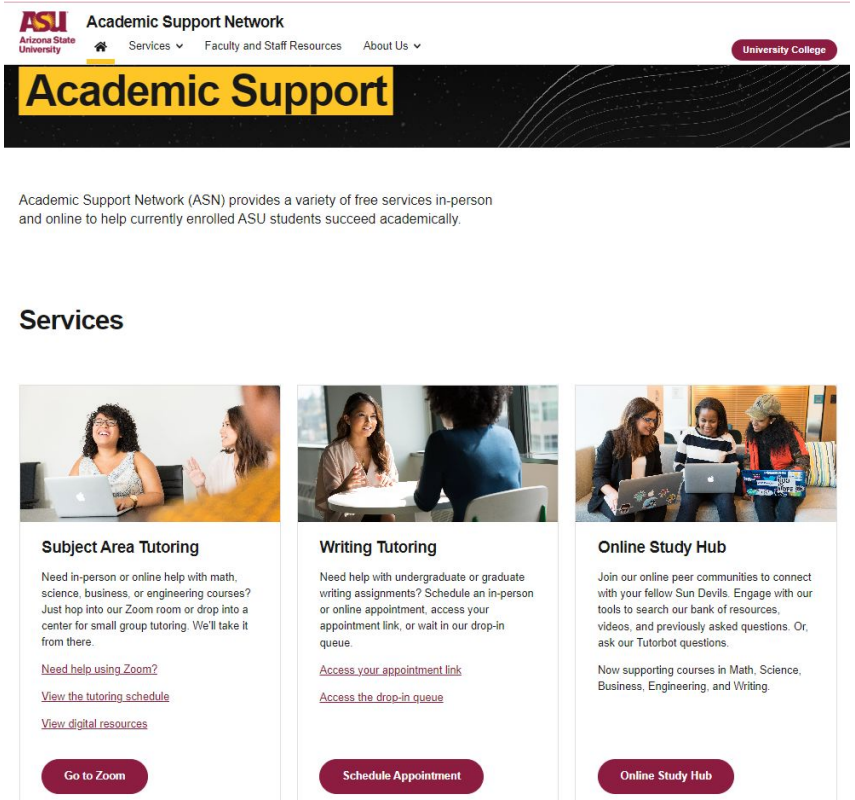
https://asuasn.info/ASNSurvey

48

More Questions?

Check out our other resources!

tutoring.asu.edu

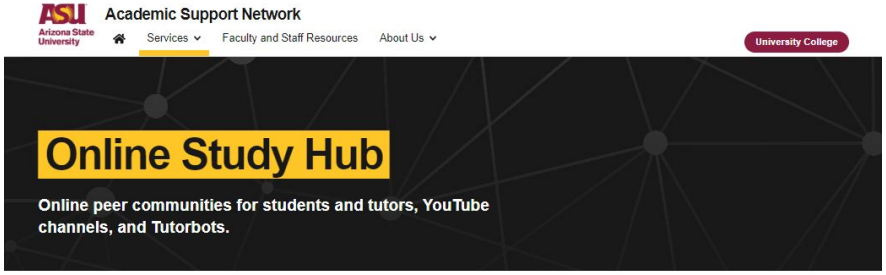


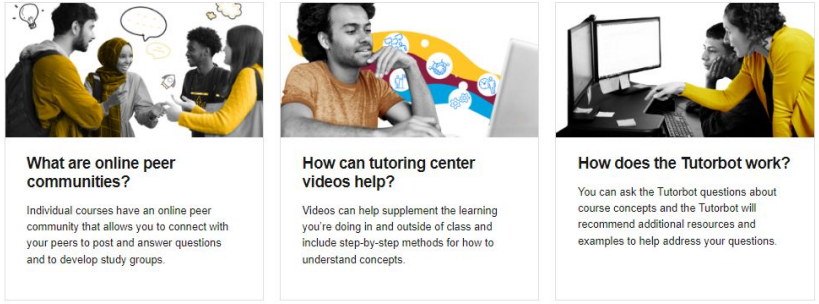
1. **Click on ‘Go to Zoom’ to log onto our Online Tutoring Center.**
2. **Click on ‘View the tutoring schedule’ to see when tutors are available for specific courses.**

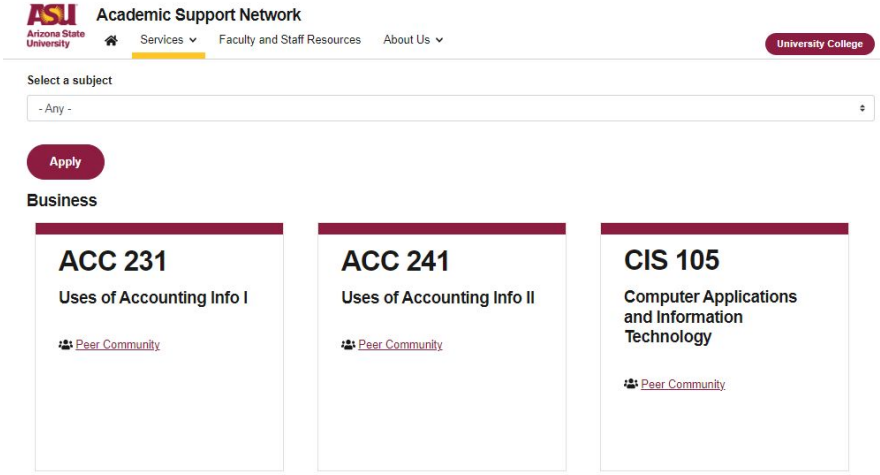
More Questions?

Check out our other resources!

tutoring.asu.edu/online-study-hub



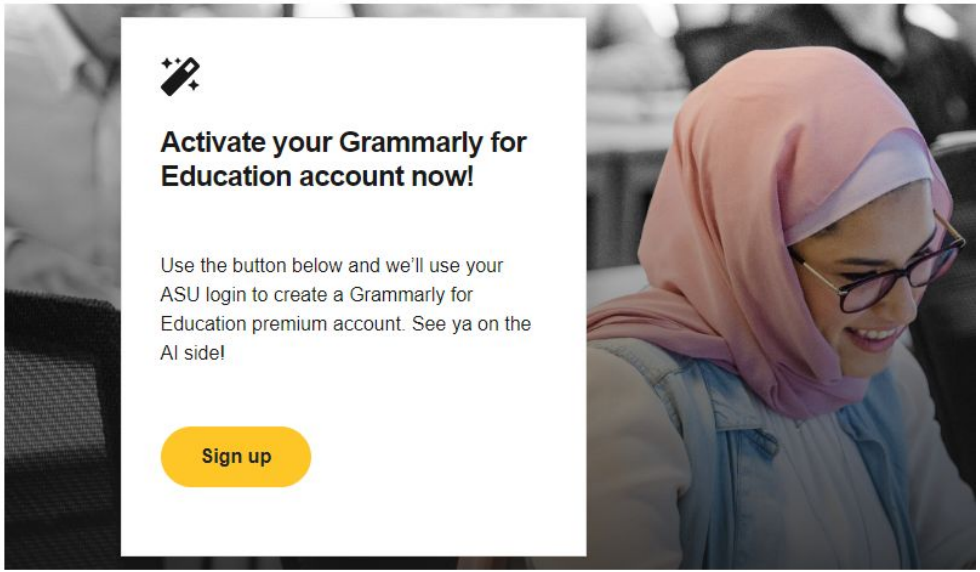




Don't forget to check out the Online Study Hub for additional resources!

Expanded Writing Support Available

Including Grammarly for Education, at no cost!



tutoring.asu.edu/expanded-writing-support

*Available slots for this pilot are limited

Additional Resources

- Course Repo
- Gradle Documentation
- GitHub SSH Help
- Linux Man Pages
- OSI Interactive
- MDN HTTP Docs
  - Requests
  - Responses
- JSON Guide
- org.json Docs
- javax.swing package API
- Swing Tutorials