SER 321 B Session

SI Session

Tuesday, April 8th 2025

10:00 am - 11:00 am MST

Agenda

- Serialization
  - JSON Review
  - Formats, Types, & Streams
- Client Port Examination
- Threading our System

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

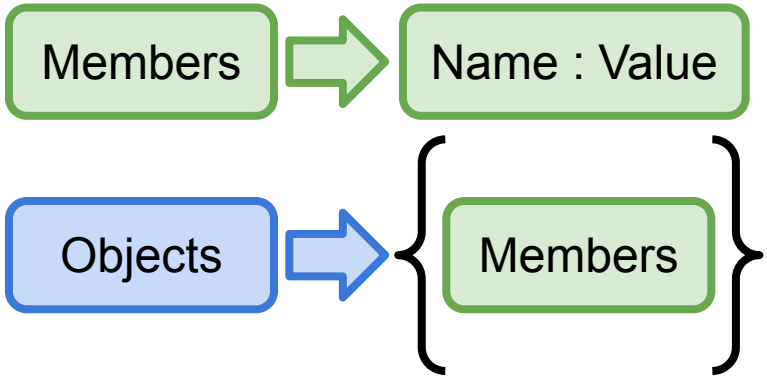
- Use the chat feature to interact with the presenter and respond to presenter's questions.
- Annotations are encouraged

org.json Docs

JSON Guide

SER 321

JSON Structure



org.json Docs

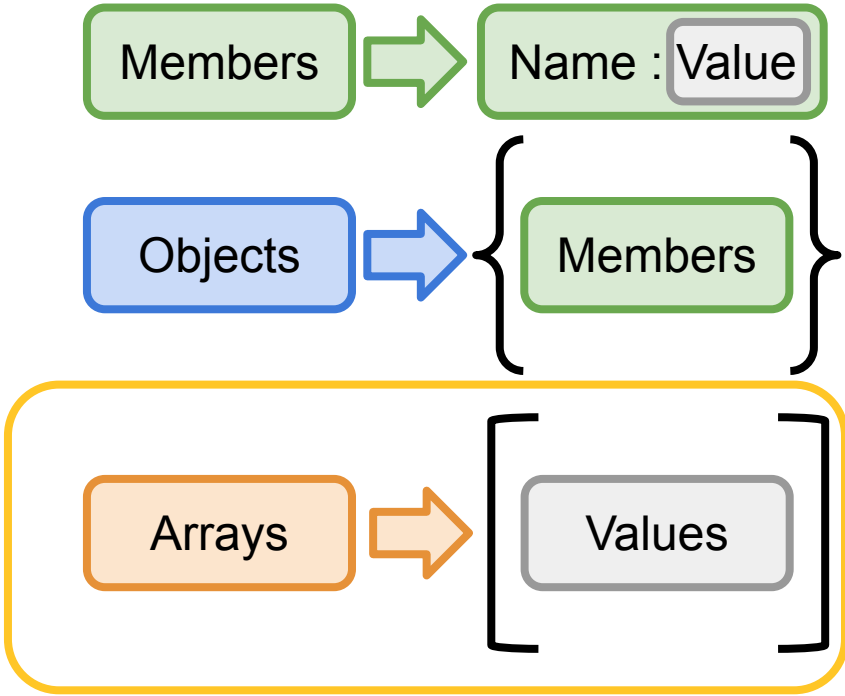
JSON Guide

SER 321

JSON Structure

What is a valid value?

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



org.json Docs

JSON Guide

SER 321

JSON Structure

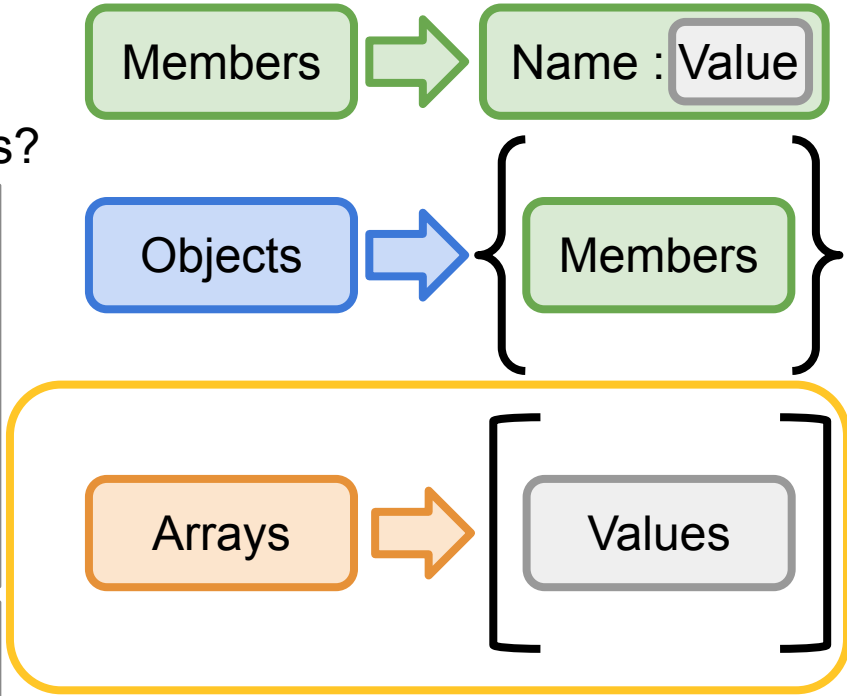
Which of the following are valid JSON arrays?

1.
```
{ "hooks": [
    "192.30.252.0/22",
    "185.199.108.0/22",
  ]
}
```

2.
```
{ "weather": [
    { "id": 803,
      "main": "Clouds",
      "icon": "04d"
    }
  ]
}
```

3.
```
{ "makes" : [
    true,
    "Ford",
    "Honda",
    "Toyota"
  ]
}
```

4.
```
{ "readings" : [
    58.93,
    76,
    65.81
  ]
}
```



org.json Docs

JSON Guide

SER 321

JSON Structure

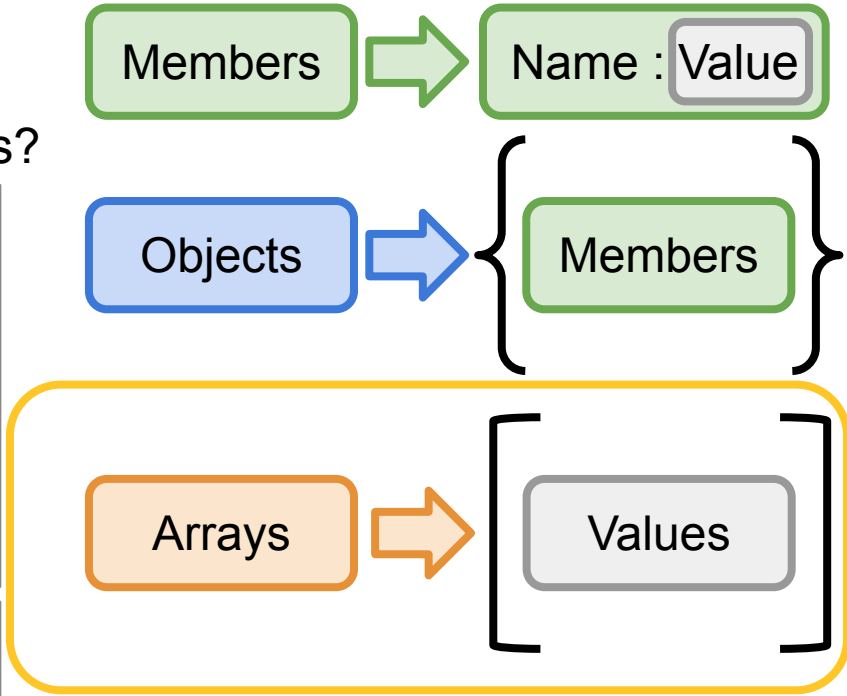
Which of the following are valid JSON arrays?

1.
```
{ "hooks": [
    "192.30.252.0/22",
    "185.199.108.0/22",
  ]
}
```

2.
```
{ "weather": [
    { "id": 803,
      "main": "Clouds",
      "icon": "04d"
    }
  ]
}
```

3.
```
{ "makes" : [
    true,
    "Ford",
    "Honda",
    "Toyota"
  ]
}
```

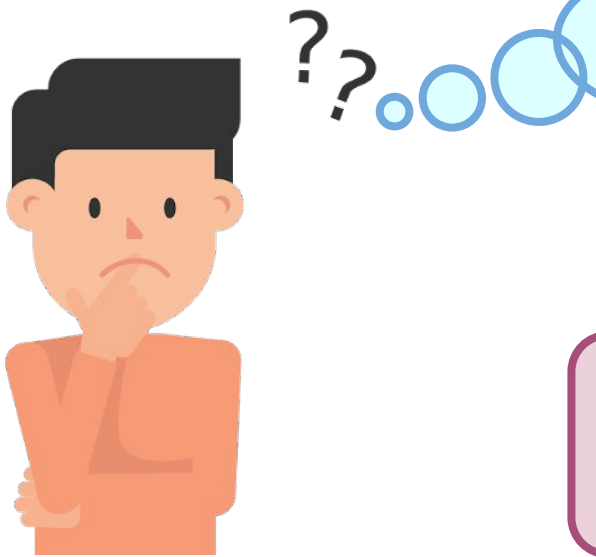
4.
```
{ "readings" : [
    58.93,
    76,
    65.81
  ]
}
```



All of them!

SER 321
Serialization

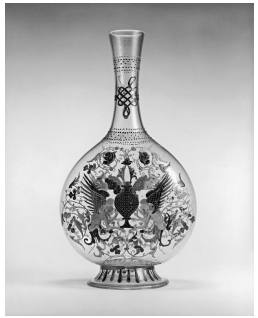
What is *serialization*?



"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"

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

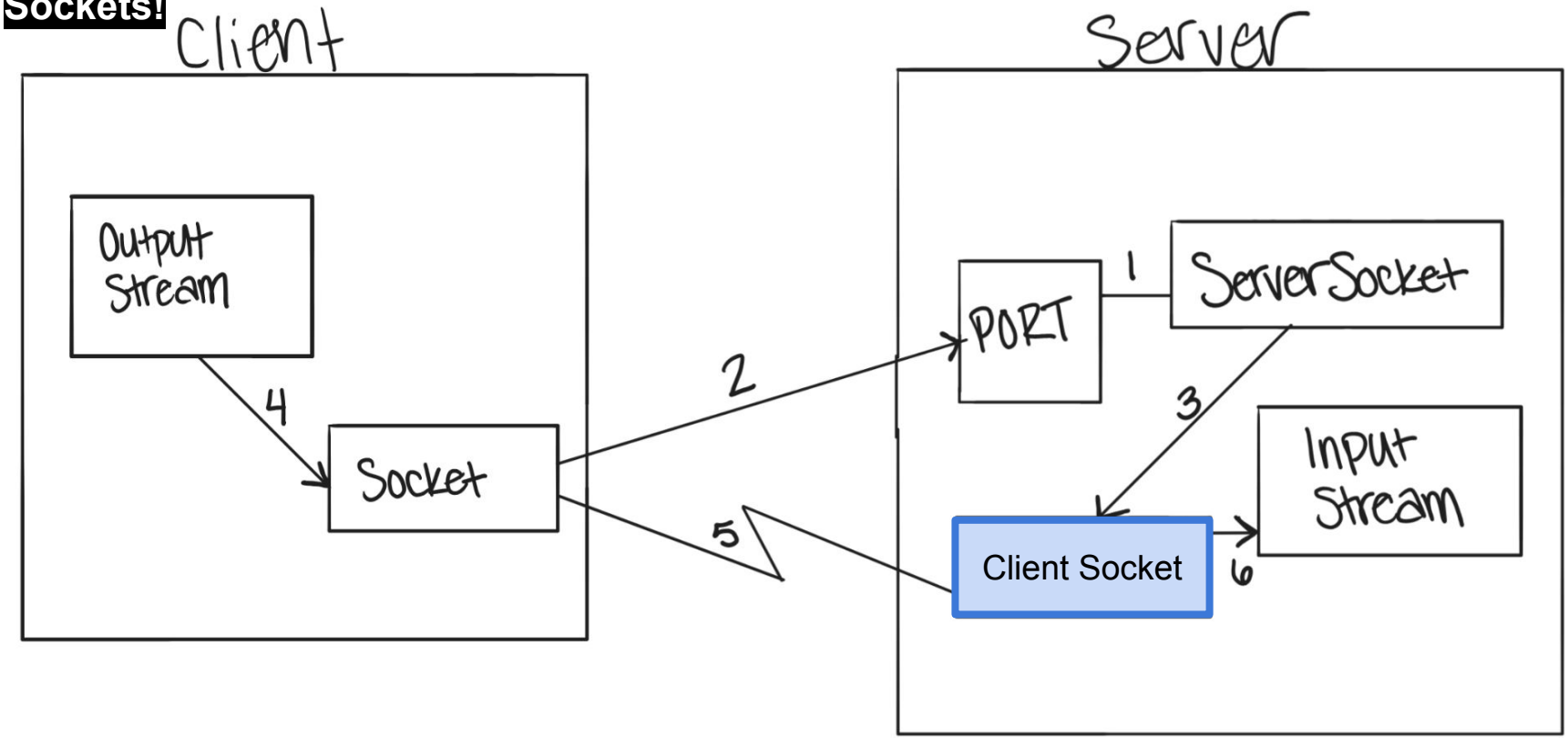
Primitive DATA Types

Object Stream

Java Objects

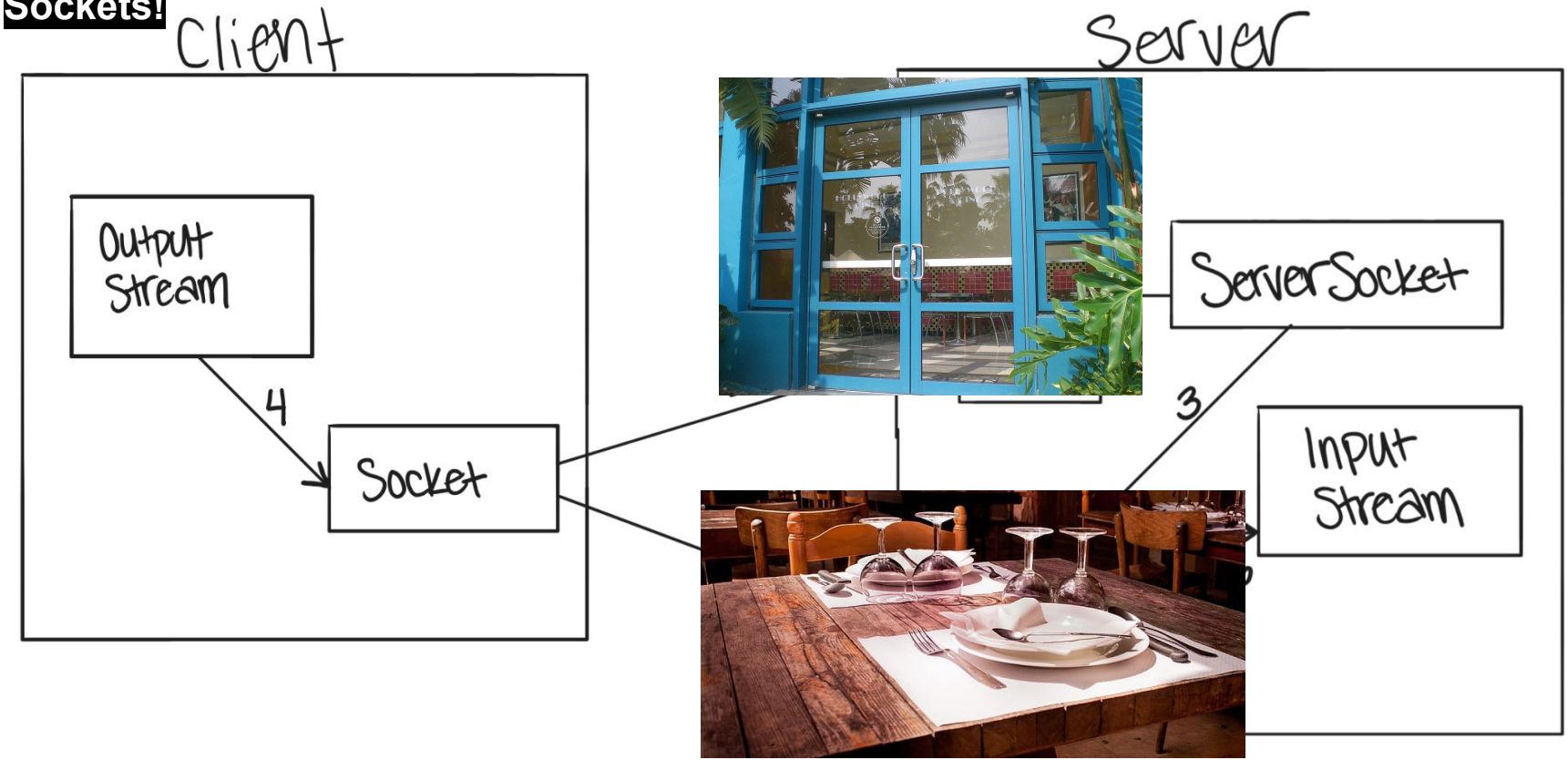
SER 321

Sockets!



SER 321

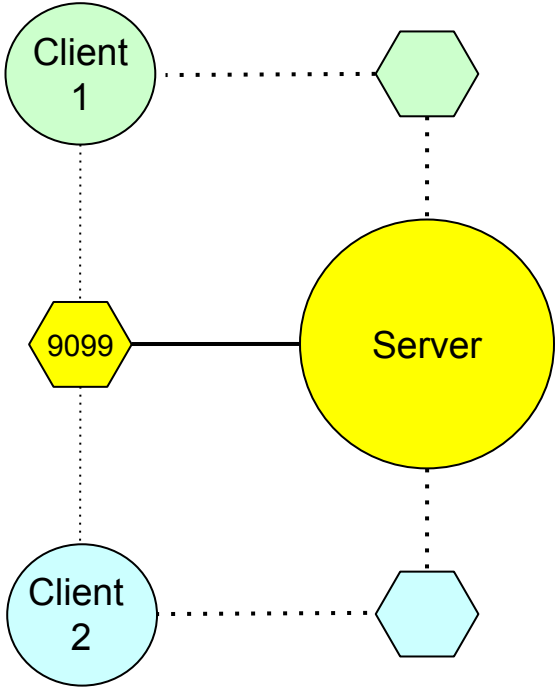
Sockets!

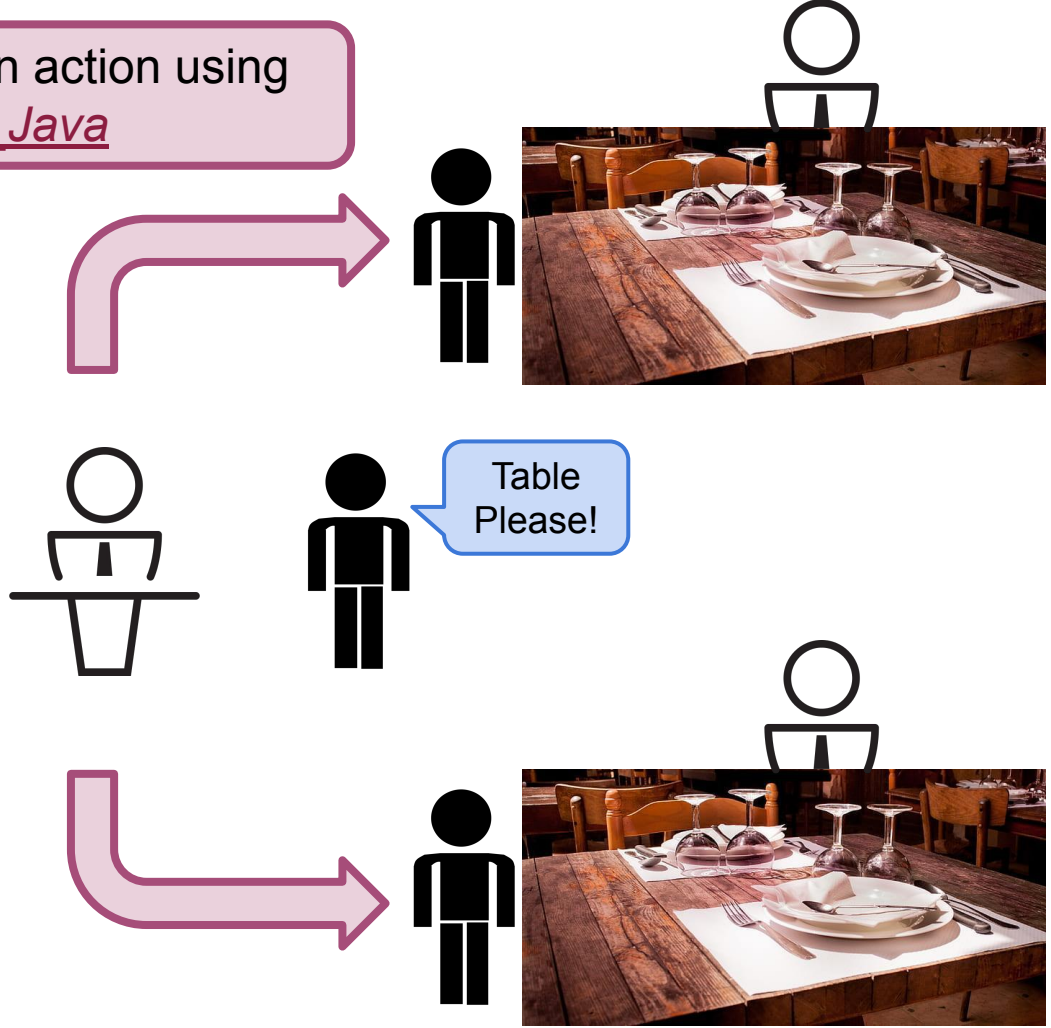


SER 321

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
```

```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
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

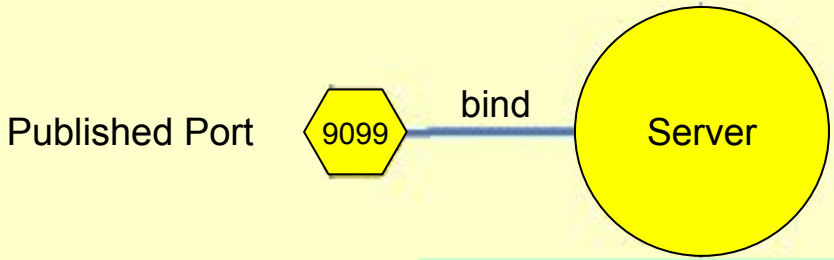
```
> Task :runClient
Connected to server at localhost:9099
Values of the Socket Object for the Server:
        Host: /127.0.0.1
        Port: 9099
        Local Port: 60296
String to send>
<==========----> 75% EXECUTING [31s]
> :runClient
```

Server

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

        } catch

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

        int buf
        byte cl
         hile(t
                                                                                    ıt
                                                                                    ı));
        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!

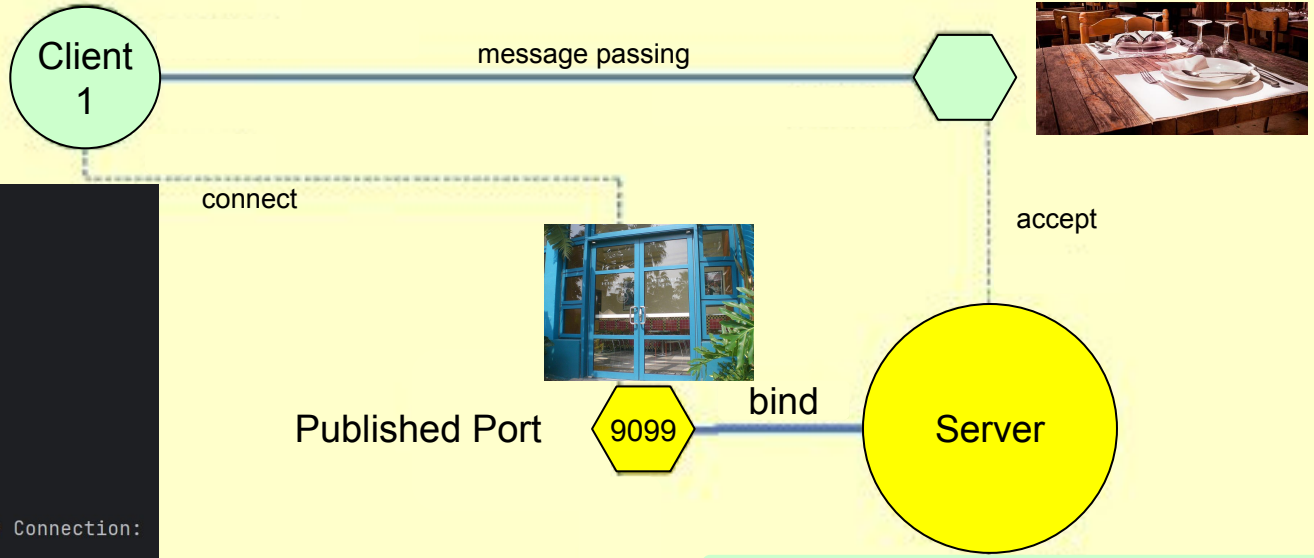
```
> Task :runServer
Server ready for connections
Server is listening on port: 9099
-----
Values of the ServerSocket Object:
Inet Address: 0.0.0.0/0.0.0.0
Local Port: 9099
Server waiting for a connection
Server connected to client
-----
Values of the Client Socket Object after Connection:
        Inet 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
```

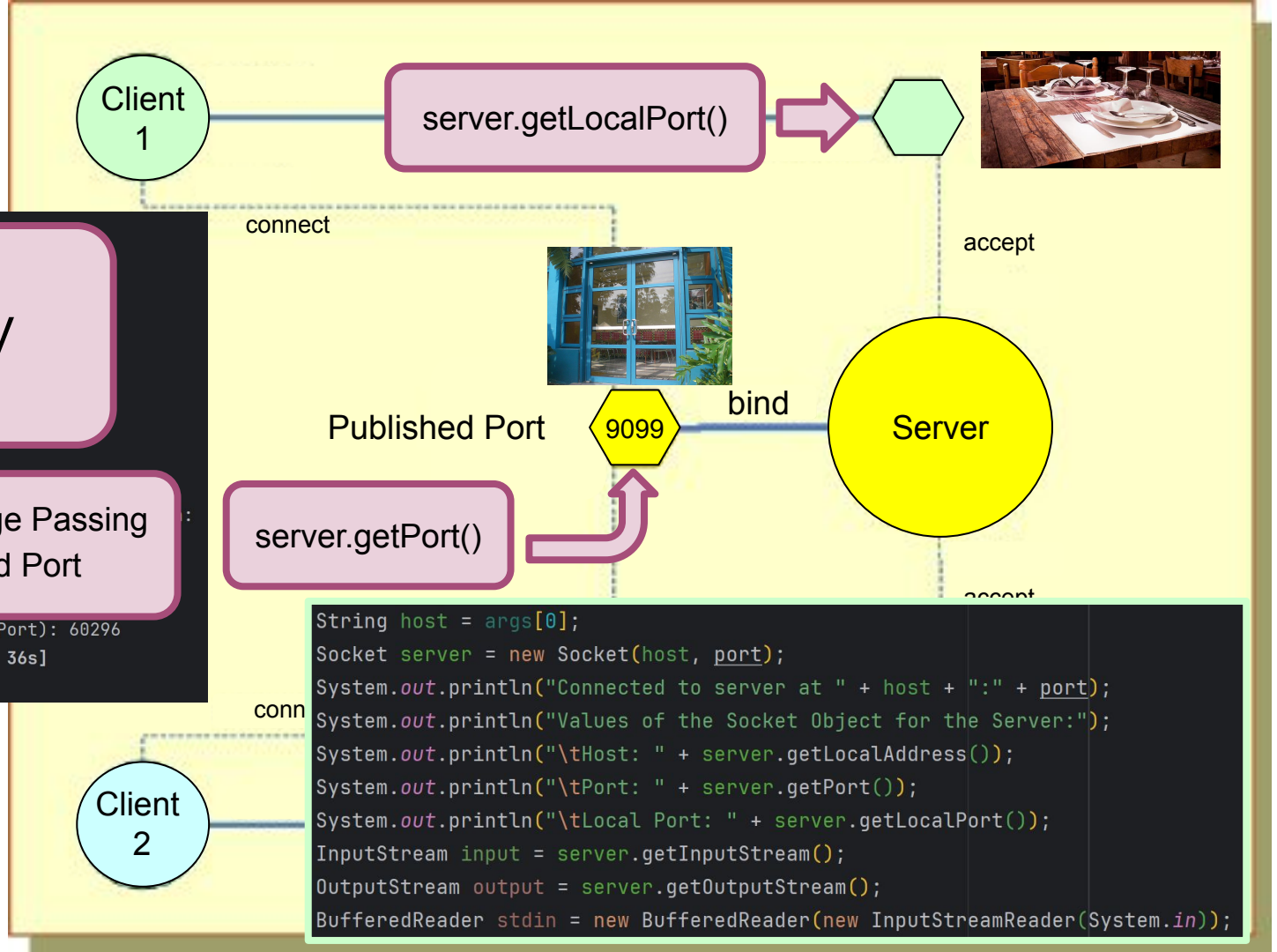
Design of an RFID Vehicle Authentication System: A Case Study for Al-Nahrain University Campus - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Client-and-Server-Socket-Ports_fig4_282671198

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));
```

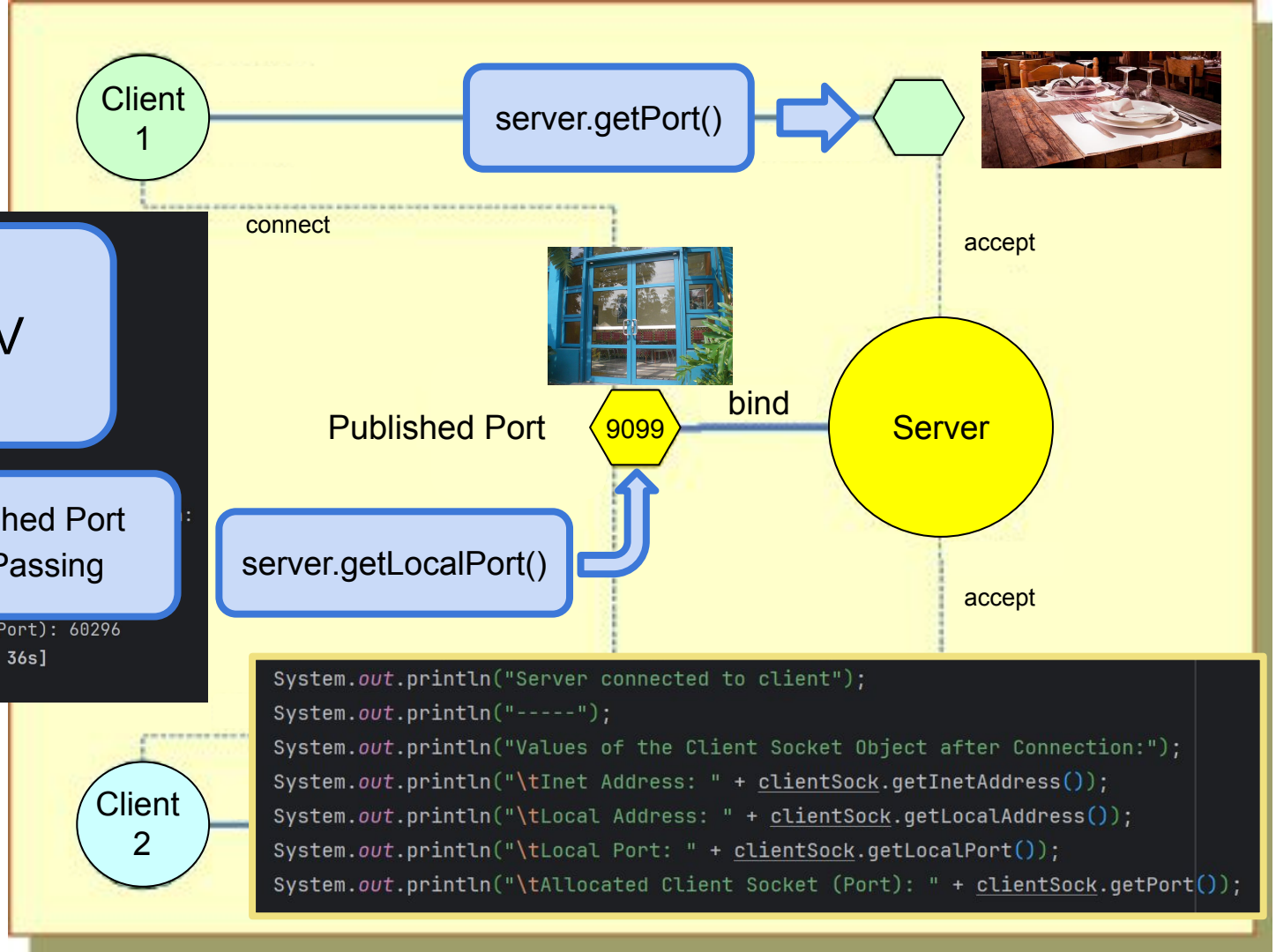
Design of an RFID Vehicle Authentication System: A Case Study for Al-Nahrain University Campus - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Client-and-Server-Socket-Ports_fig4_282671198

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
```

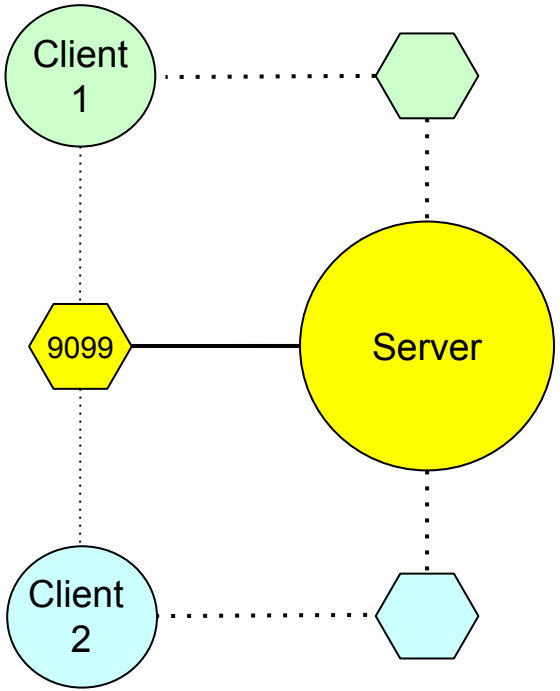


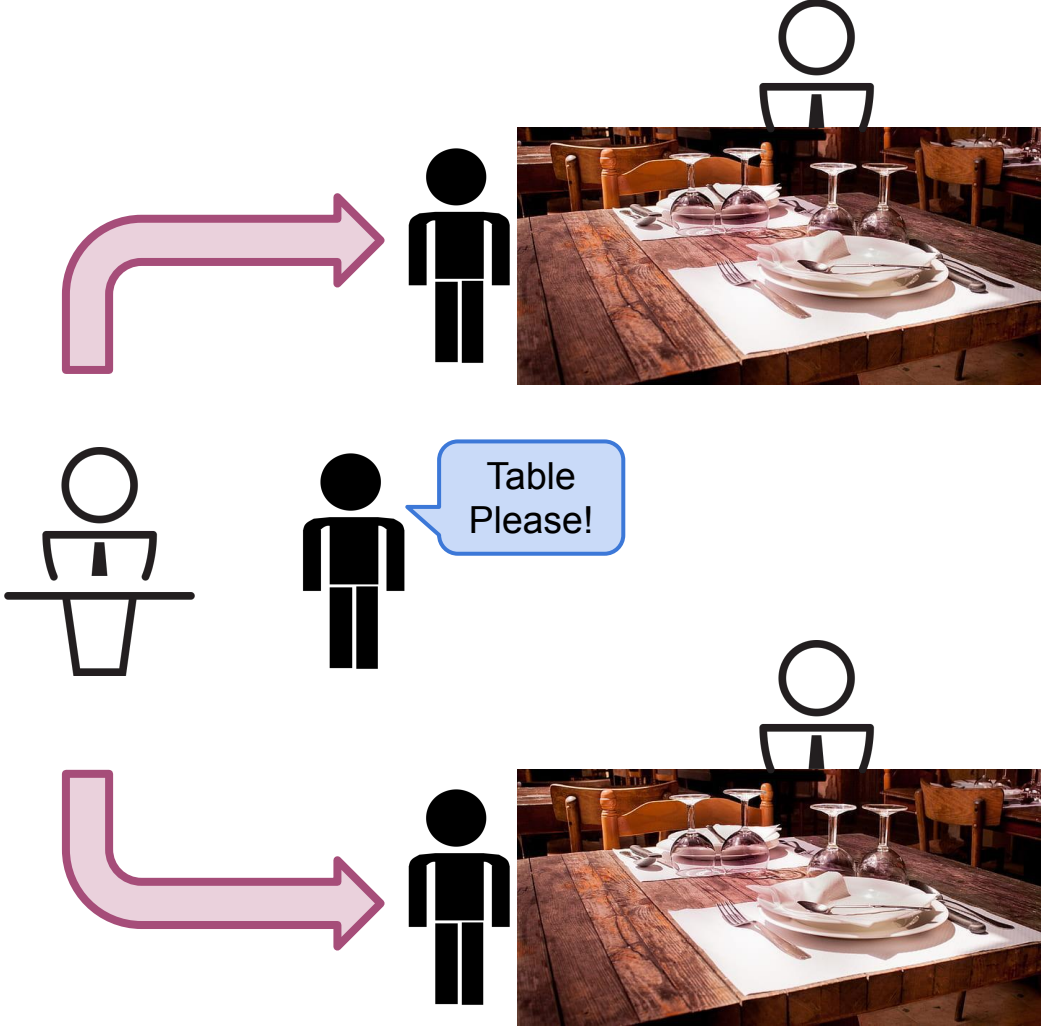

```
System.out.println("Server connected to client");
System.out.println("-----");
System.out.println("Values of the Client Socket Object after Connection:");
System.out.println("\tInet Address: " + clientSock.getInetAddress());
System.out.println("\tLocal Address: " + clientSock.getLocalAddress());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

You have two systems…

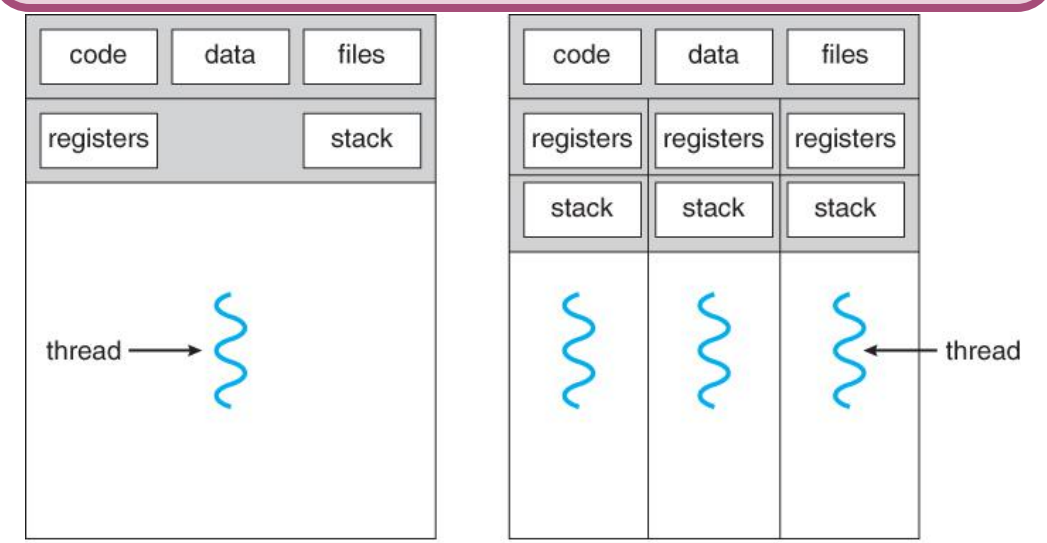
How can we test our server with multiple clients?

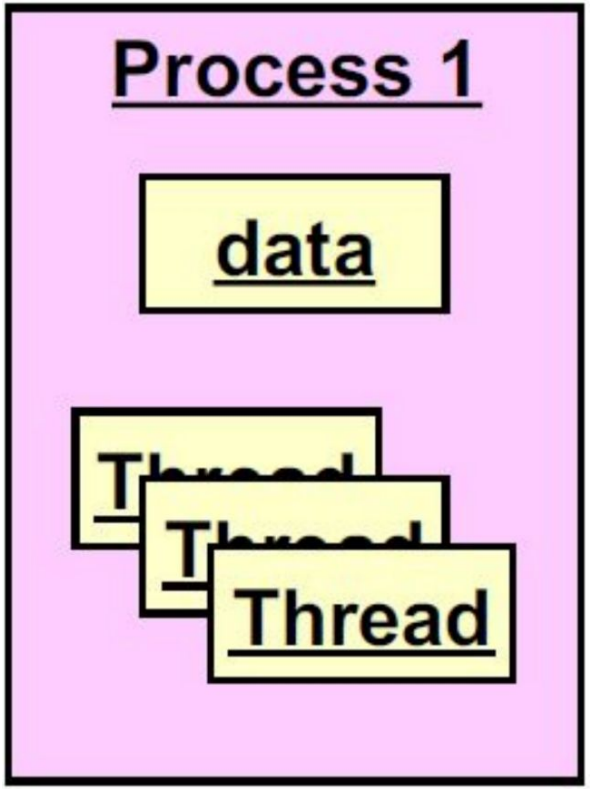
?

SER 321
Threads

What does that imply?

Remember that they exist *within* the parent process





SER 321

Threading Pitfalls

Race Condition	A thread never gains access to the resource it needs
Starvation	A thread is only able to acquire some of the resources it needs
Deadlock	More than one thread accesses a single resource at the same time

SER 321

Threading Pitfalls

Race Condition	A thread never gains access to the resource it needs
Starvation	A thread is only able to acquire some of the resources it needs
Deadlock	More than one thread accesses a single resource at the same time

SER 321
Threaded Pitfalls

What's the difference?

Starvation	vs.	Deadlock
A thread never gains access to the resource it needs		A thread is only able to acquire some of the resources it needs
Waiting to access the ***CPU***		Waiting to access another ***resource***
Ready to go; never gets a chance		***Not*** ready to go

NetworkDeadlock

SER 321
Threading Pitfalls

As the project name implies, we encounter a ***deadlock***.

But what happened?

Client

```java
class SockClient {
    public static void main (String args[]) throws Exception {
        Socket      sock = new Socket( host: "localhost",  port: 8888);    //Any IP name

        ObjectInputStream in = new ObjectInputStream(sock.getInputStream());
        ObjectOutputStream out = new ObjectOutputStream(sock.getOutputStream());

        String s = (String) in.readObject();
        out.writeObject("Back at you");

        in.close();
        out.close();
        sock.close();
    }
}
```

Server

```java
class SockServer {
    public static void main (String args[]) throws Exception {

        int count = 0;
        ServerSocket    serv = new ServerSocket( port: 8888);

        Socket sock = serv.accept();

        ObjectInputStream in = new ObjectInputStream(sock.getInputStream());
        ObjectOutputStream out = new ObjectOutputStream(sock.getOutputStream());

        String s = (String) in.readObject();
        System.out.println("Received " + s);
        out.writeObject("Back at you");
        System.out.println("Received " + s);

        in.close();
        out.close();
        sock.close();
    }
}
```

```
PS C:\ASU\SER321\examples_repo\ser321examples\Threads\NetworkDeadlock> gradle
server
<=========----> 75% EXECUTING [1m 33s]
> :server
```

```
PS C:\ASU\SER321\examples_repo\ser321examples\Threads\NetworkDeadlock> gradle
client
Starting a Gradle Daemon, 1 busy and 1 stopped Daemons could not be reused, us
e --status for details
<=========----> 75% EXECUTING [53s]
> :client
```

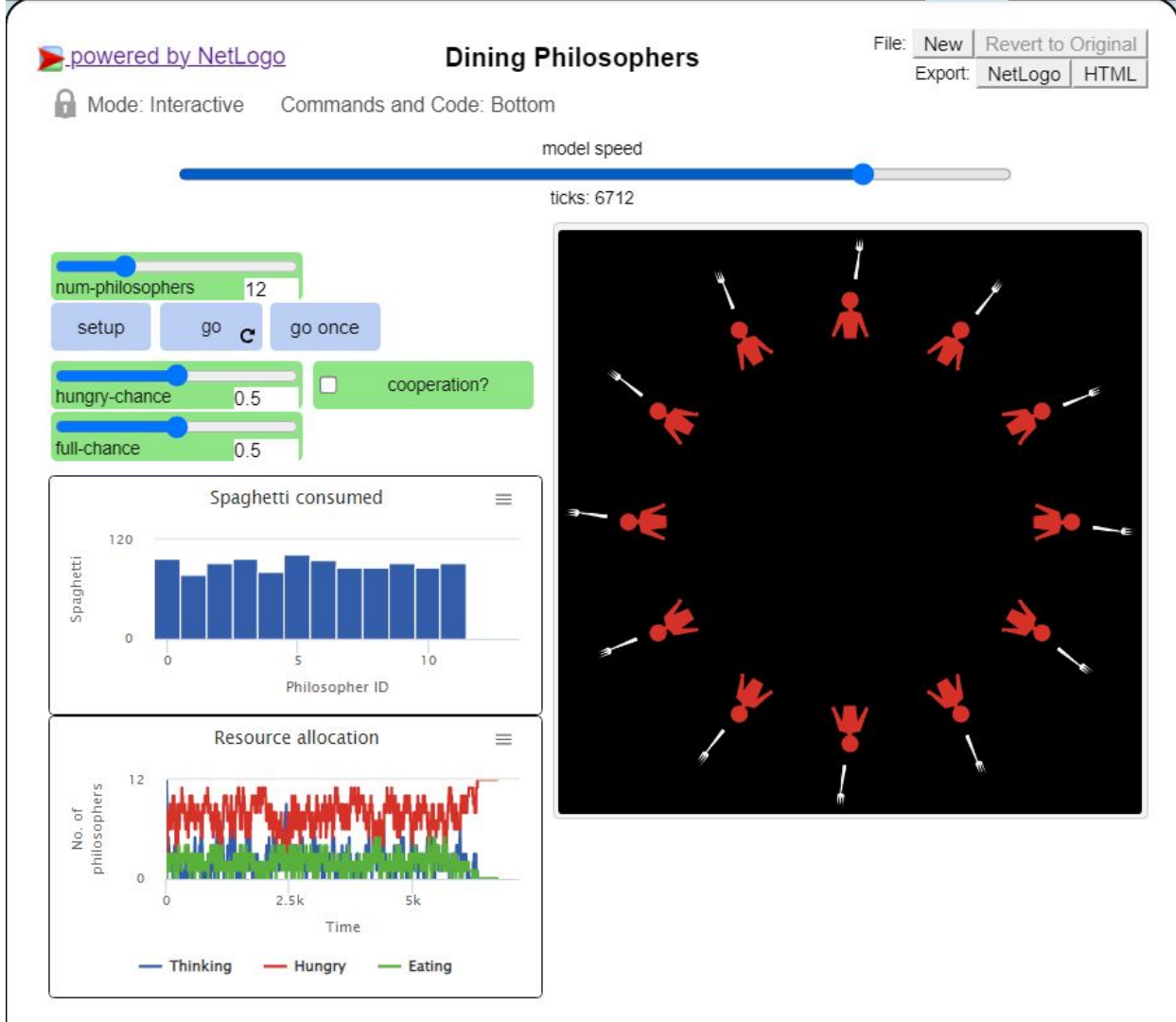
Dining Philosophers

SER 321
Threading Pitfalls

What does *Spaghetti Consumed* represent?

What does *Thinking* represent?

What does *Hungry* represent?



SER 321

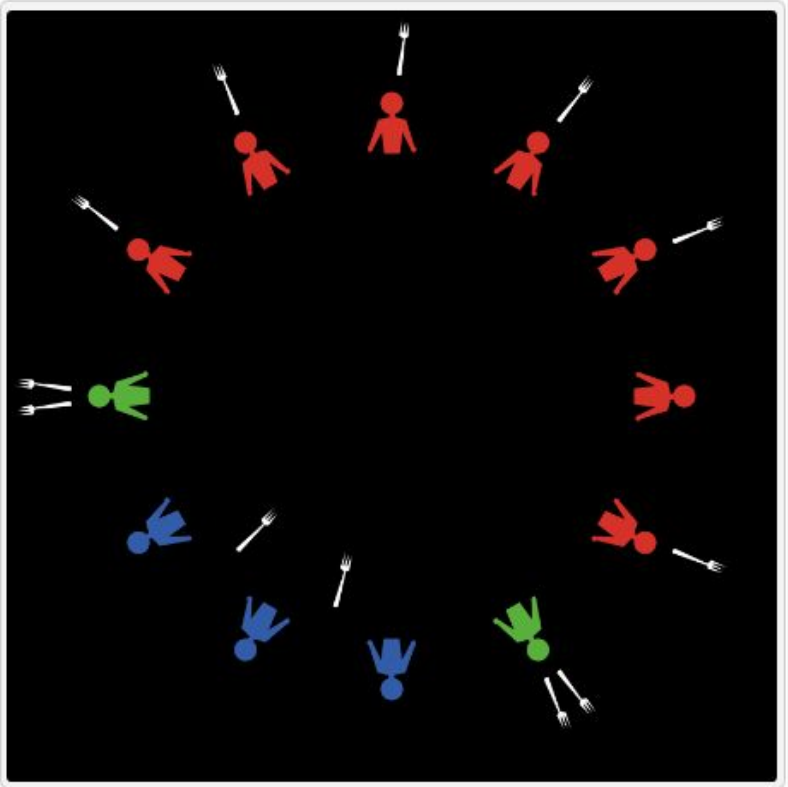
Dining Philosophers

Can we take a guess at what is happening here?

What are the **BLUE** people doing?

What are the **GREEN** people doing?

What are the **RED** people doing?



Austin Walter's Traffic Comparison

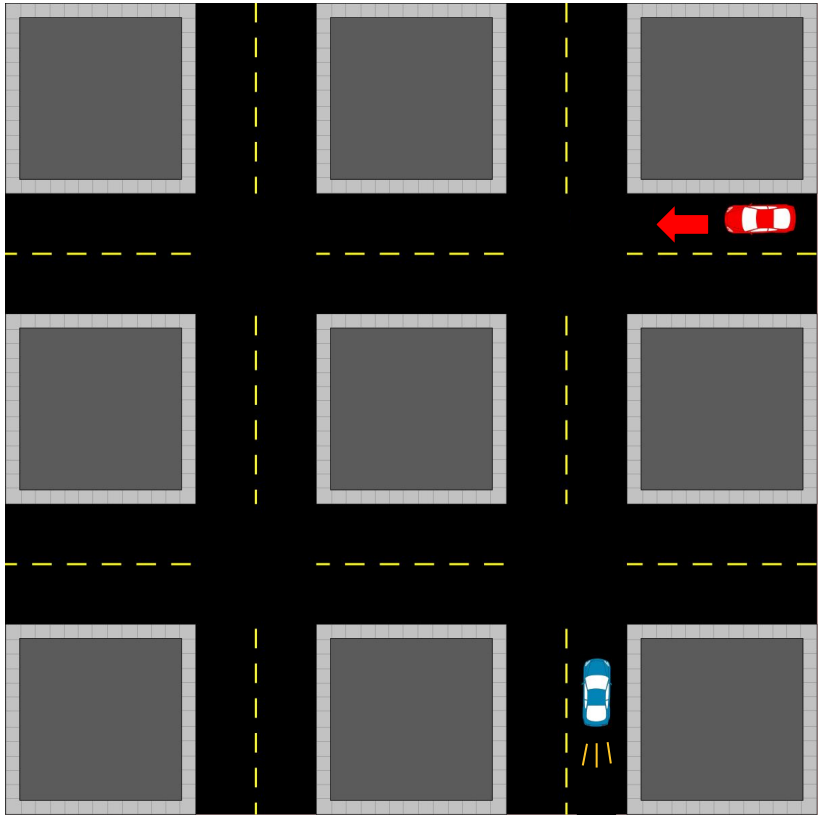
SER 321

Threading Pitfalls

Race Condition

Crash

More than one thread accesses a single resource at once



Austin Walter's Traffic Comparison

SER 321

Threading Pitfalls

Race Condition

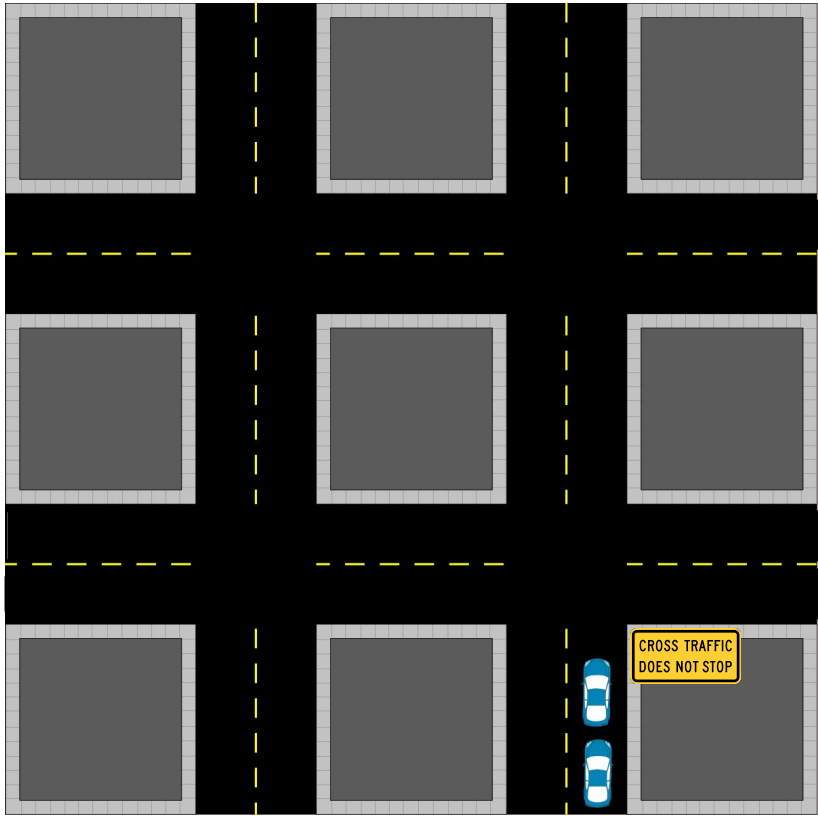
Crash

More than one thread accesses a single resource at once

Starvation

Cross Traffic

A thread never gains access to the resource it needs



Austin Walter's Traffic Comparison

SER 321

Threading Pitfalls

Race Condition

Crash

More than one thread accesses a single resource at once

Starvation

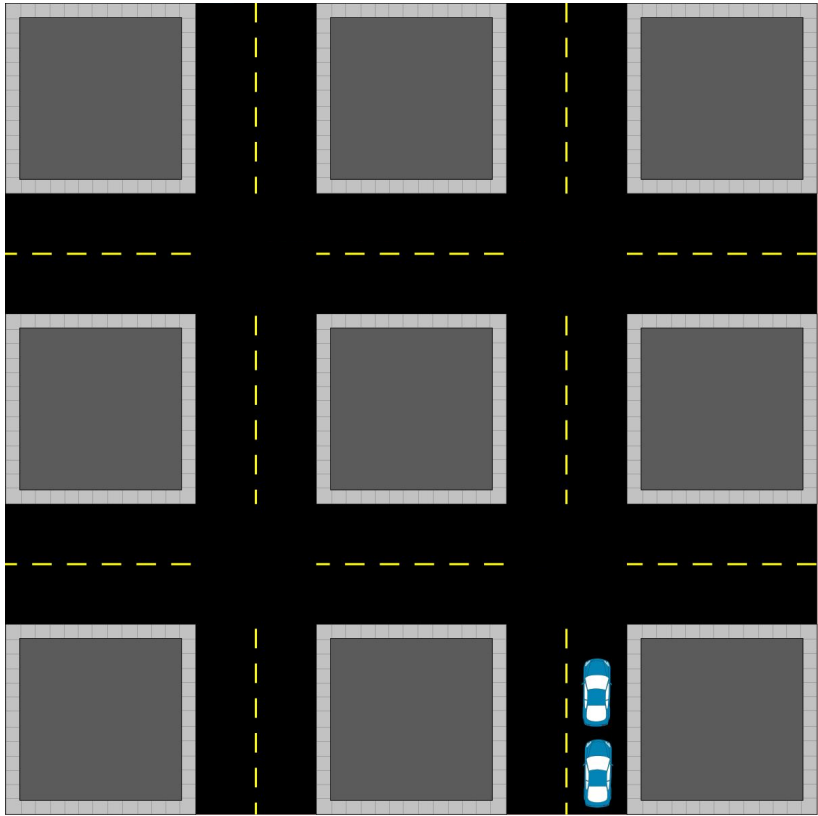
Cross Traffic

A thread never gains access to the resource it needs

Deadlock

Gridlock

A thread is only able to acquire some of the needed resources



SER 321

Scratch Space

Upcoming Events

SI Sessions:

- Thursday, April 10th at 7:00 pm MST
- Sunday, April 13th at 7:00 pm MST
- Tuesday, April 15th at 10:00 am 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:

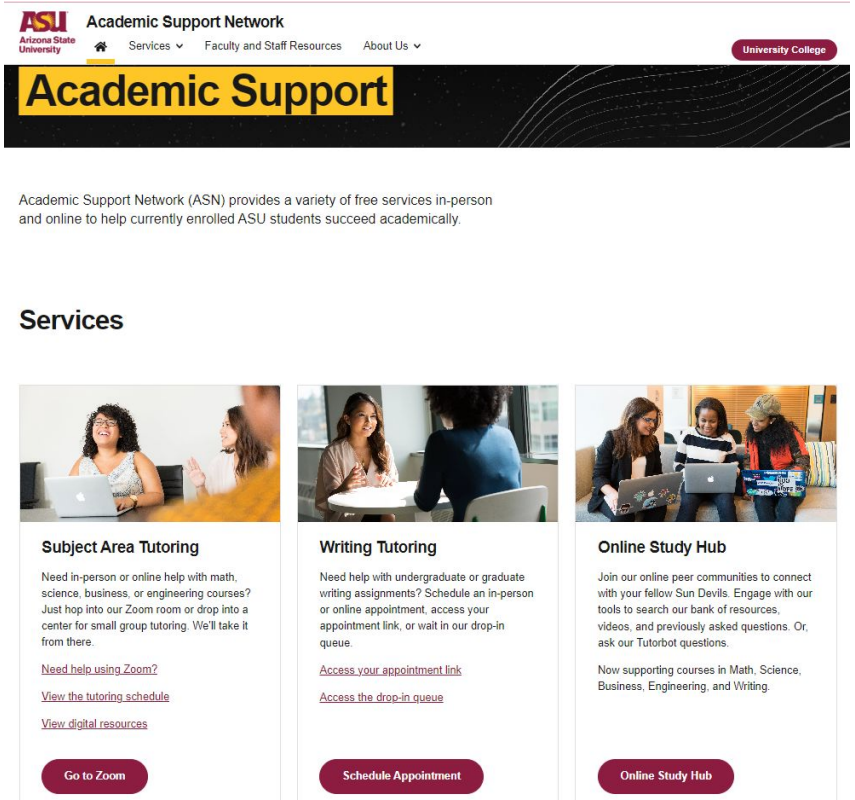
https://asuasn.info/ASNSurvey

42

More Questions?

Check out our other resources!

tutoring.asu.edu



1 – Go to Zoom

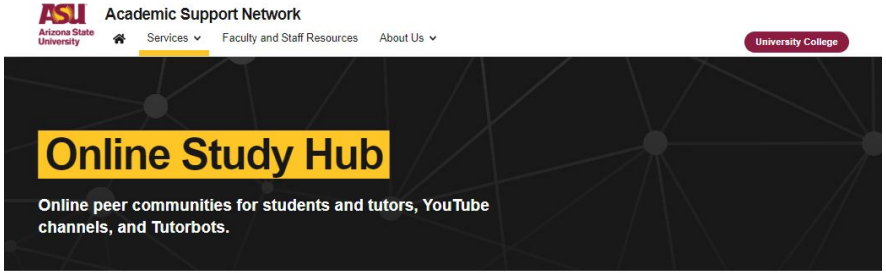
2- Need help using Zoom?
View the tutoring schedule
View digital resources

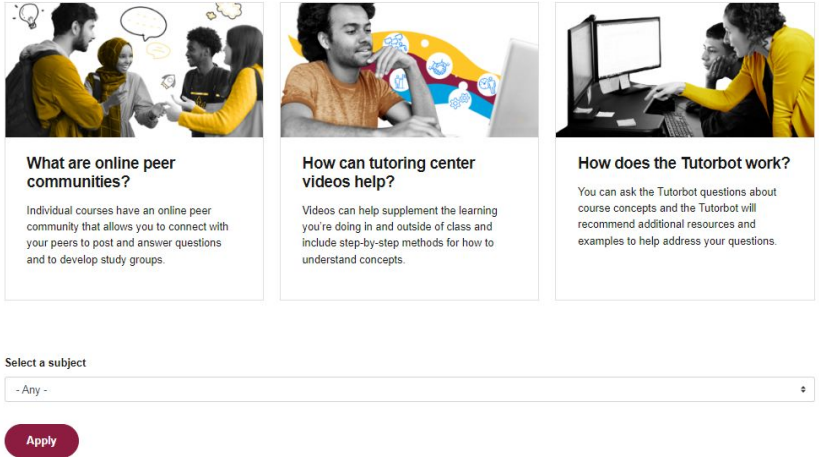
1. **Click on 'Go to Zoom' to log onto our Online Tutoring Center.**
2. **Click on 'View the tutoring schedule' to see when tutors are available for specific courses.**

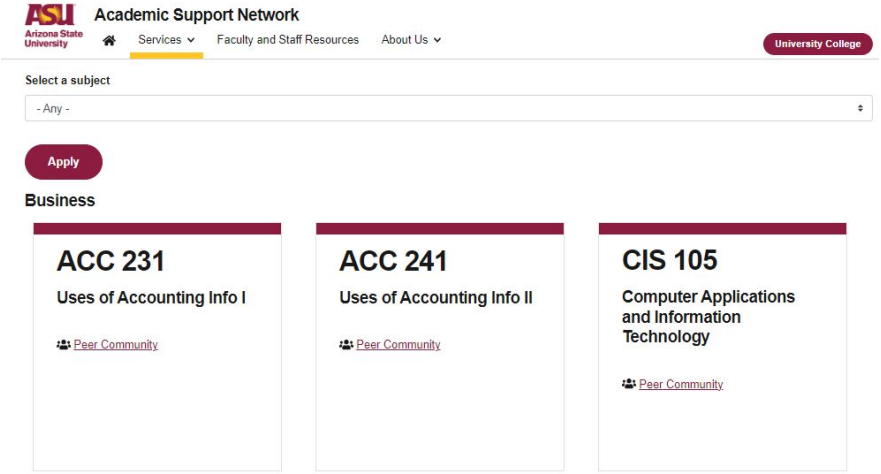
More Questions?

Check out our other resources!

tutoring.asu.edu/online-study-hub



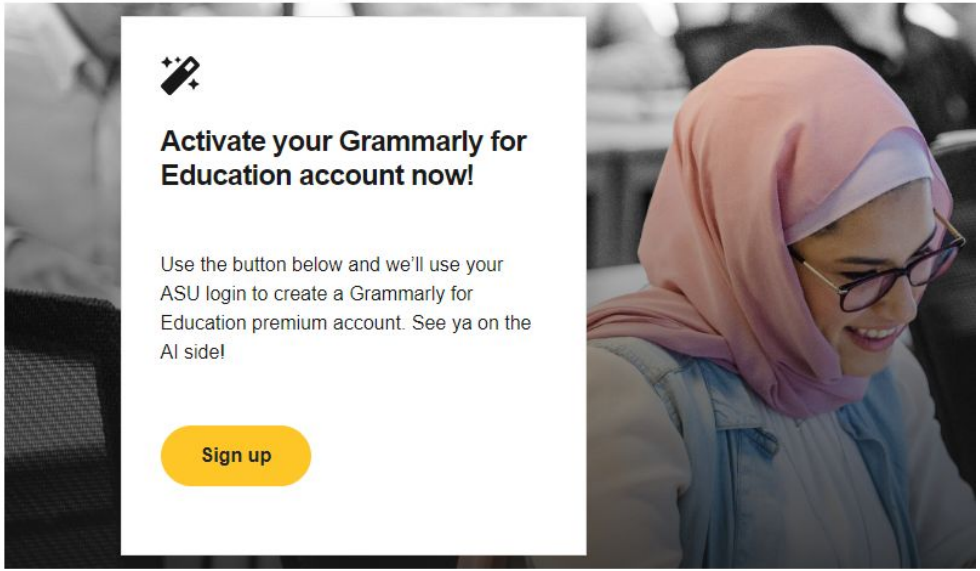




Don't forget to check out the Online Study Hub for additional resources!

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
- Dining Philosophers Interactive
- Austin G Walters Traffic Comparison