SER 321 B Session

SI Session

Sunday, November 3rd 2024

7:00 pm - 8:00 pm MST

Agenda

OSI Review

Sockets!

- Informal Steps
- Formal 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

OSI Model

Unit	Layer	What we are *really* talking about	
Data	Application		Content/Payload Info
Data	Presentation		
Data	Session		
Segment	Transport		
Packet	Network		Transmission Info
Frame	Data Link		
Bits	Physical		

SER 321

OSI Model

Unit	Layer	What we are *really* talking about	
Data	Application		Content/Payload Info
Data	Presentation		
Data	Session		
Segment	Transport		
Packet	Network		Transmission Info
Frame	Data Link		
Bits	Physical		

SER 321

OSI Model

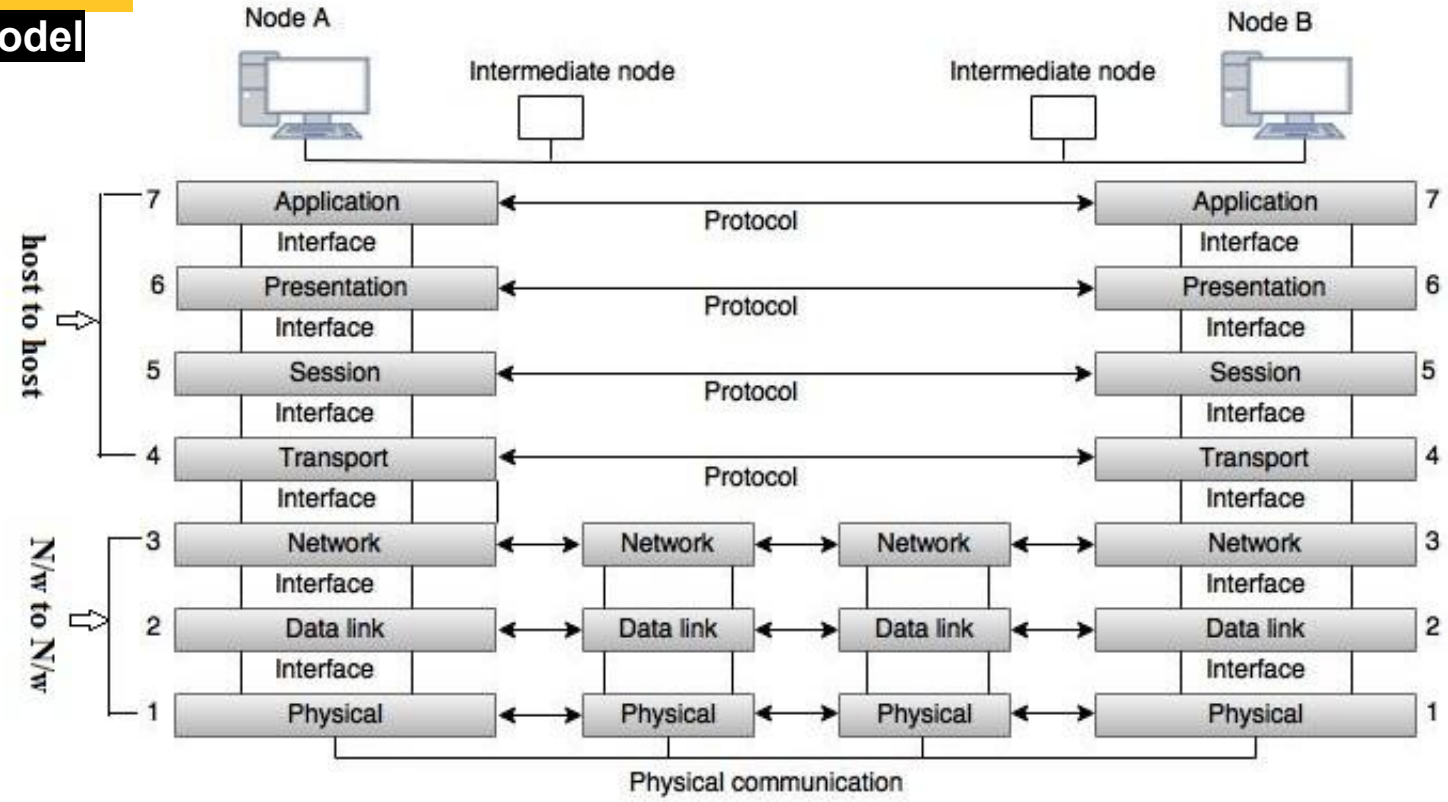


Fig: OSI Model

SER 321

TCP vs. UDP Matching

Unreliable

TCP

OR

UDP

SER 321

TCP vs. UDP Matching

Connection-Oriented

TCP

Reliable

OR

UDP

Unreliable

SER 321

TCP vs. UDP Matching

Uses Streams

TCP	OR	UDP
Reliable		Unreliable
Connection-Oriented		Connectionless

SER 321

TCP vs. UDP Matching

Has Less Overhead

TCP	OR	UDP
Reliable		Unreliable
Connection-Oriented		Connectionless
Uses Streams		Uses Datagrams

SER 321

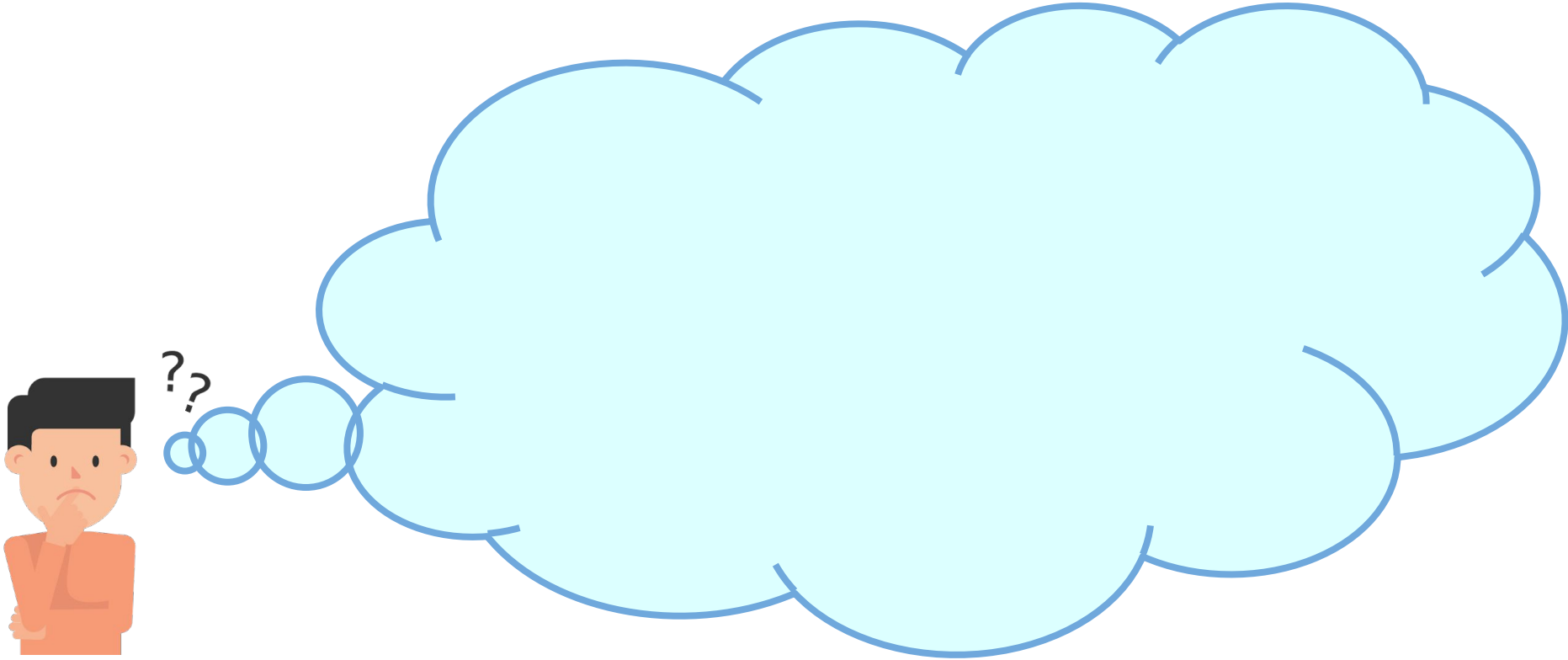
TCP vs. UDP Matching

Has Less Overhead

TCP	OR	UDP
Reliable		Unreliable
Connection-Oriented		Connectionless
Uses Streams		Uses Datagrams
Has More Overhead		Has Less Overhead

SER 321
Sockets!

What do we need for a client/server connection?



SER 321
Sockets!

Sockets allow our client and server to communicate!

Need to define **3 properties** before usage

Location	IP or DNS	142.251.46.206	www.google.com
Connection Semantics	TCP or UDP	Connection Oriented	Connectionless
Message Format	Protocol Specs	Synchronous	Asynchronous
		Stateless	Stateful
		Binary	Text
		Headers	No Headers

SER 321
Sockets!

Sockets allow our client and server to communicate!

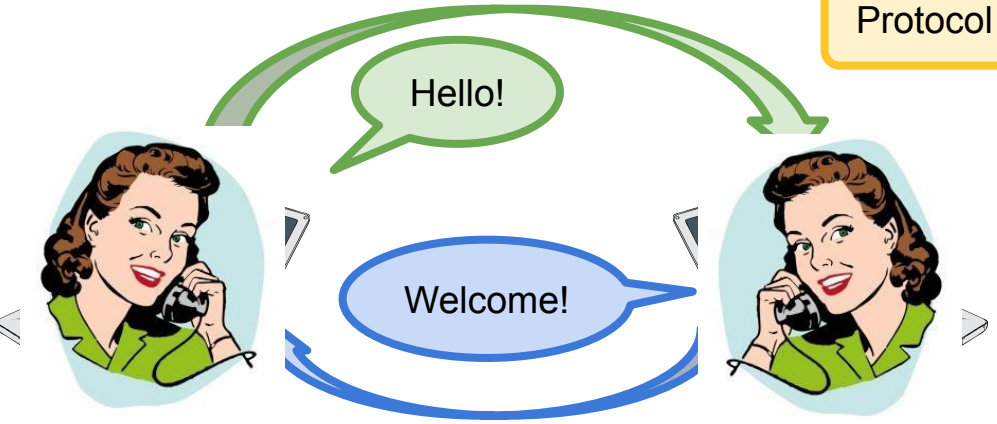
Person

Conversation Flow

Conversation Content

l to define **3 properties** before usage

IP or DNS	142.251.46.206	www.google.com
TCP or UDP	Connection Oriented	Connectionless
Protocol Specs	Synchronous	Asynchronous
	Stateless	Stateful
	Binary	Text
	Headers	No Headers



SER 321
Client Socket

Steps for the **Client Socket**

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

Assign 3-1 Starter Code

SER 321
Client Socket

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

```
class SockClient {
  static Socket sock = null;  4 usages
  static String host = "localhost";  2 usages
  static int port = 8888;  2 usages
  static OutputStream out;  2 usages
  // Using and Object Stream here and a Data Stream as return. Could both b
  // to show the difference. Do not change these types.
  static ObjectOutputStream os;  4 usages
  static DataInputStream in;  3 usages
  public static void main (String args[]) {

    if (args.length != 2) {...}

    try {
      host = args[0];
      port = Integer.parseInt(args[1]);
    } catch (NumberFormatException nfe) {
      System.out.println("[Port|sleepDelay] must be an integer");
      System.exit( status: 2);
    }

    try {
      connect(host, port); // connecting to server
      System.out.println("Client connected to server.");
      boolean requesting = true;
      while (requesting) {
        System.out.println("What would you like to do: 1 - echo, 2 - add, 3
        Scanner scanner = new Scanner(System.in);
        int choice = Integer.parseInt(scanner.nextLine());
        // You can assume the user put in a correct input, you do not need
        // You can assume the user inputs a String when asked and an int wh
        JSONObject json = new JSONObject(); // request object
        switch(choice) {
          case 0:
            System.out.println("Choose quit. Thank you for using our servic
            requesting = false;
            break;
          case 1:
            System.out.println("Choose echo, which String do you want to se
            String message = scanner.nextLine();
            json.put("type", "echo");
            json.put("data", message);
            break;
          case 2:
```

```
        if(!requesting) {
          continue;
        }

        // write the whole message
        os.writeObject(json.toString());
        // make sure it wrote and doesn't get cached in a buffer
        os.flush();

        // TODO: handle the response
        // - not doing anything other than printing payload
        // !! you will most likely need to parse the response for th
        String i = (String) in.readUTF();
        JSONObject res = new JSONObject(i);
        System.out.println("Got response: " + res);
        if (res.getBoolean( key: "ok")){
          if (res.getString( key: "type").equals("echo")) {
            System.out.println(res.getString( key: "echo"));
          } else {
            System.out.println(res.getInt( key: "result"));
          }
        } else {
          System.out.println(res.getString( key: "message"));
        }
      }
      // want to keep requesting services so don't close connection
      //overandout();

    } catch (Exception e) {
      e.printStackTrace();
    }
  }
```

SER 321
Server Socket

Steps for the **Server Socket**

1.
2.
3.
4.
5.
6.
7.
8.
9.

SER 321
Server Socket

Java handles a few steps for us…

1. Define Params
2. Create Socket
3. **C ONLY** Create a struct for the address
3-5. Mark Socket to Listen
5. Mark Socket to Listen for Connections
6. Wait for Connection
7. Handle Client Connection
8. Close Client Connection
9. Continue Listening for Connections

Assign 3-1 Starter Code

SER 321

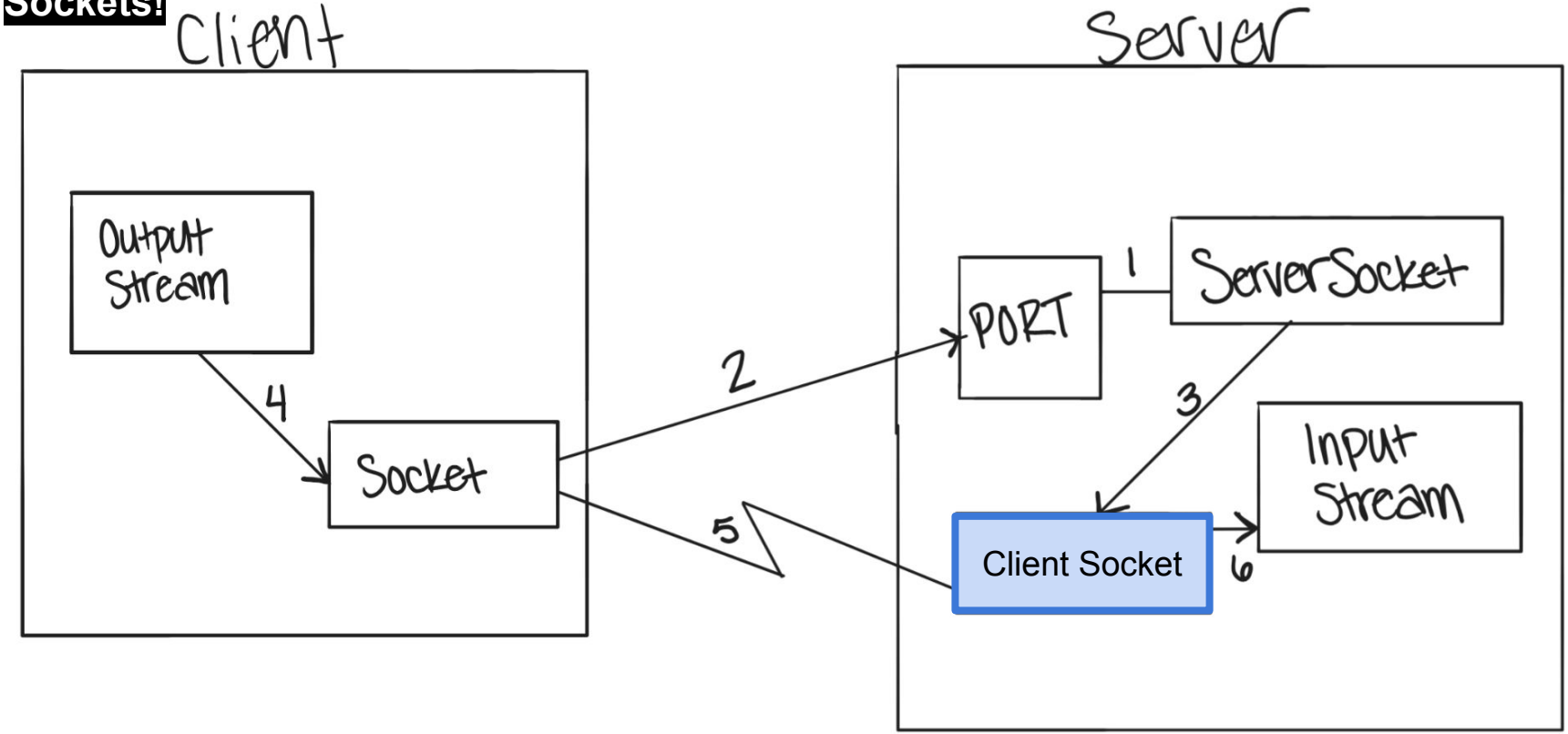
Server Socket

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. Continue Listening

```
public static void main (String args[]) {

  if (args.length != 1) {
    System.out.println("Expected arguments: <port(int)>");
    System.exit( status: 1);
  }

  try {
1   port = Integer.parseInt(args[0]);
  } catch (NumberFormatException nfe) {
    System.out.println("[Port|sleepDelay] must be an integer");
    System.exit( status: 2);
  }

  try {
    //open socket
2 & 3-5 ServerSocket serv = new ServerSocket(port);
    System.out.println("Server ready for connections");

    /** Simple loop accepting one client and calling handling one request. */

9   while (true){
      System.out.println("Server waiting for a connection");
6     sock = serv.accept(); // blocking wait
      System.out.println("Client connected");
```

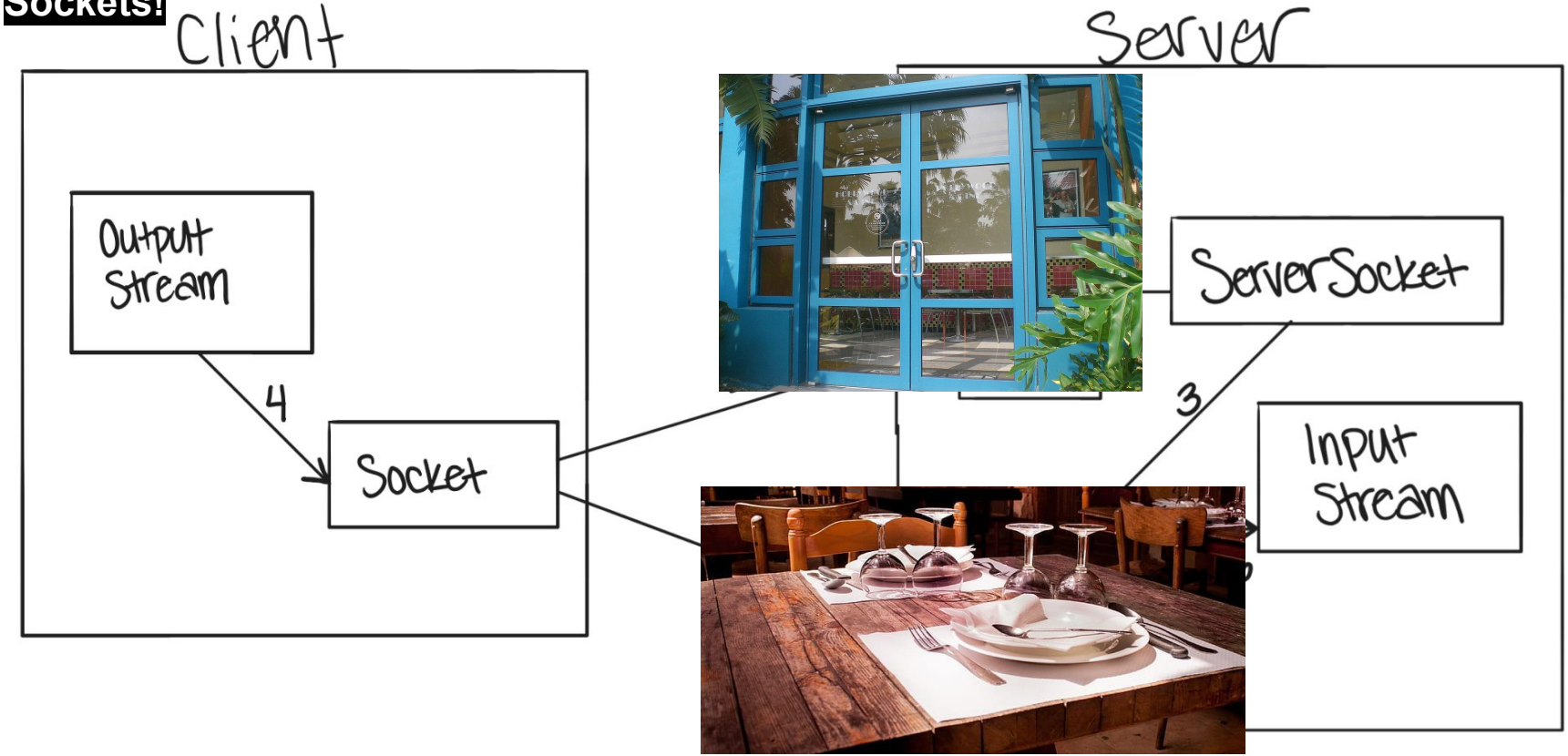
SER 321

Sockets!



SER 321

Sockets!



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!

```
> 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
```

Sockets/Echo_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

```
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!

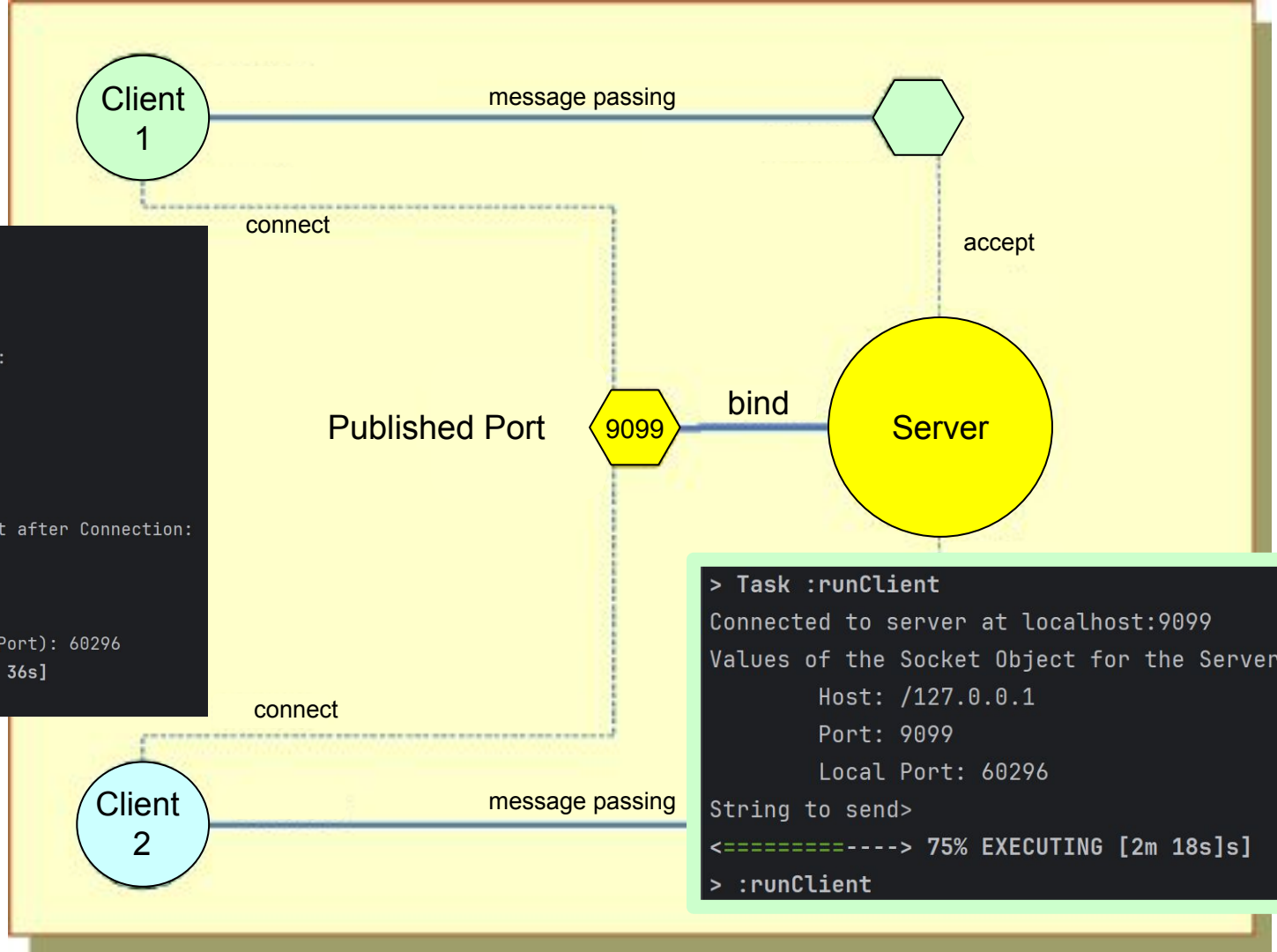
```
> Task :runServer
Server ready for connections
Server is listening on port: 9099
-----
Values of the ServerSocket Object:
Inet Address: 0.0.0.0/0.0.0.0
Local Port: 9099
Server waiting for a connection
Server connected to client
-----
Values of the Client Socket Object after Connection:
        Inet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

Sockets/Echo_Java

```
> 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
                                                    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

Scratch Space

Upcoming Events

SI Sessions:

- Tuesday, November 5th 2024 at 10:00 am MST
- Thursday, November 7th at 7:00 pm MST
- ~~Sunday, November 10th at 7:00 pm MST~~ **CANCELLED - Happy Veteran's Day!**
- Tuesday, November 12th at 10:00 am MST

Review Sessions:

- TBD - Waiting for approval

Questions?

Survey:

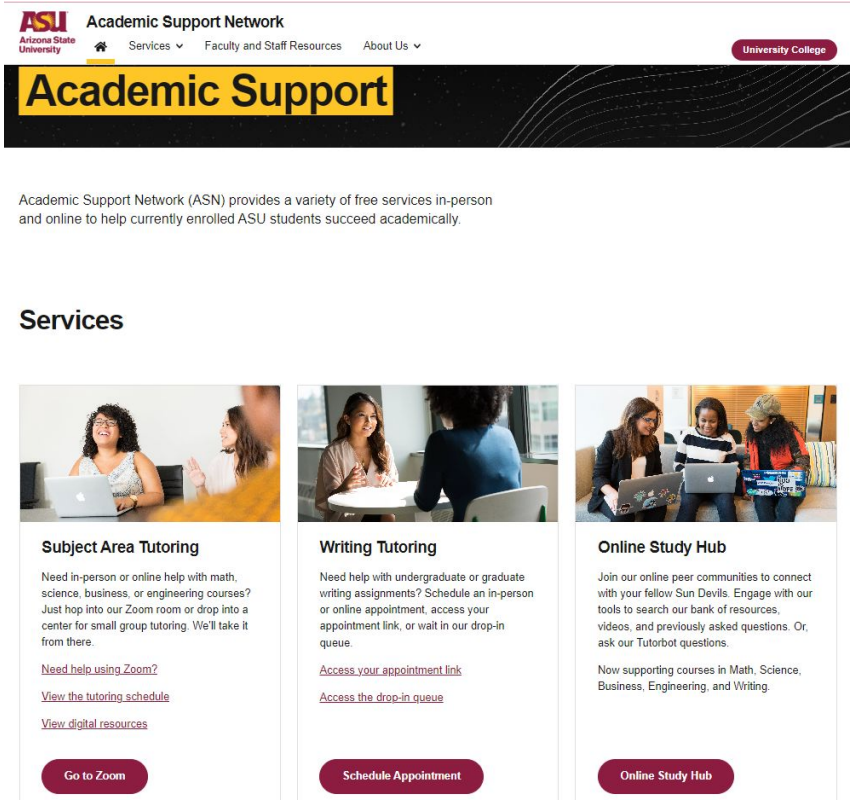
https://asuasn.info/ASNSurvey

30

More Questions?

Check out our other resources!

tutoring.asu.edu

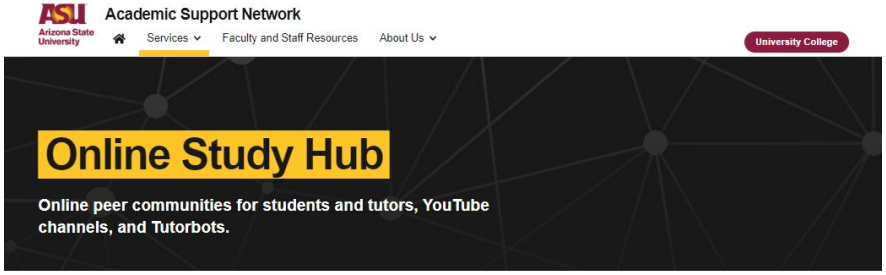


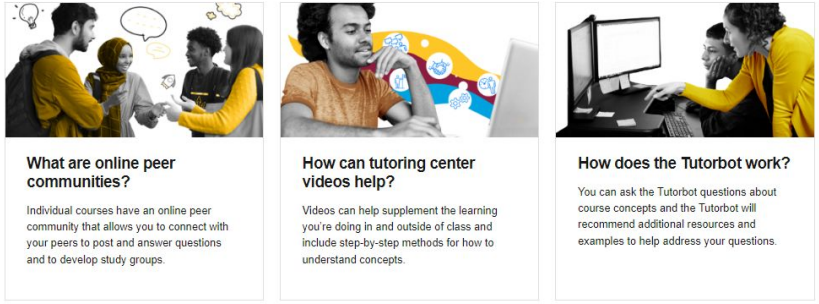
1. **Click on ‘Go to Zoom’ to log onto our Online Tutoring Center.**
2. **Click on ‘View the tutoring schedule’ to see when tutors are available for specific courses.**

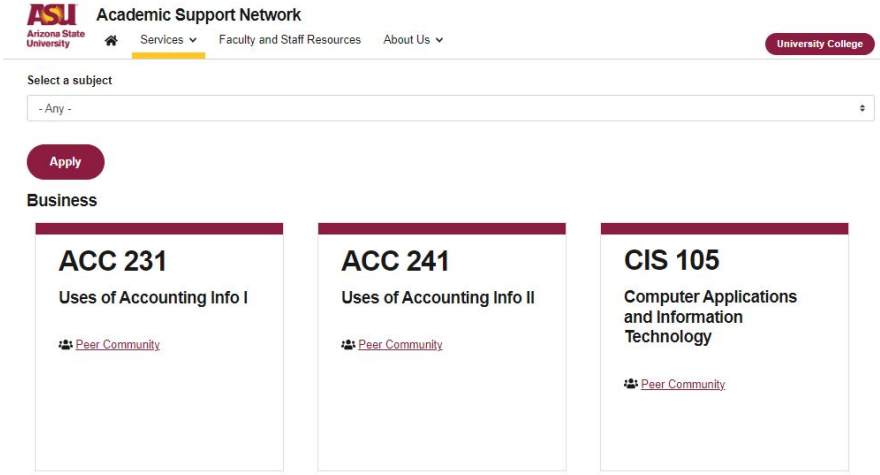
More Questions?

Check out our other resources!

tutoring.asu.edu/online-study-hub



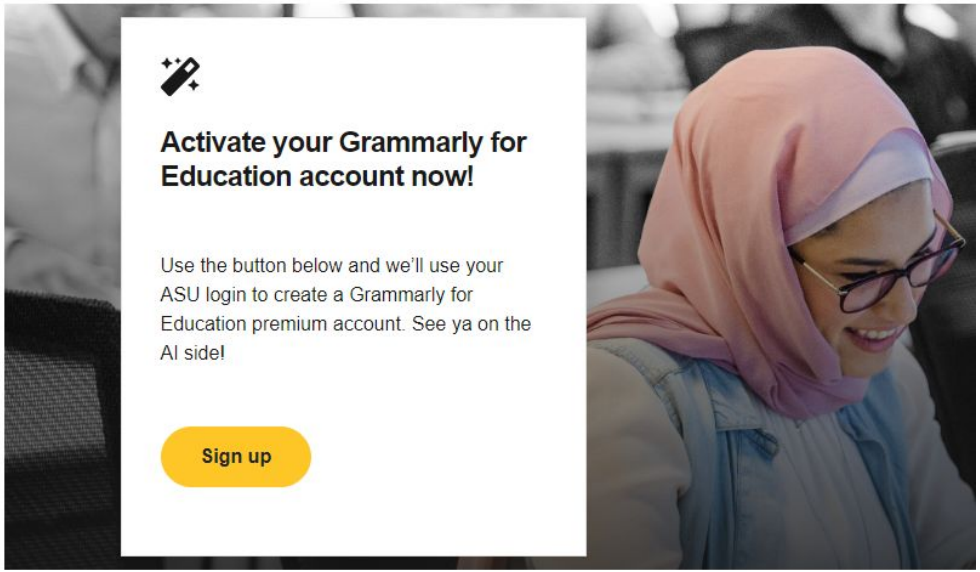




Don't forget to check out the Online Study Hub for additional resources!

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