SER 321 C Session

SI Session

Monday, June 17th 2024

6:00 pm - 7:00 pm MST

Agenda

Protobuf PSA & Demo

Theads!

- Single vs. Multi-Threaded Servers
- Why We Care
- Threading your Code

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
Protobufs

Quick PSA for Protobufs!

If you use IntelliJ, you must do this first!!

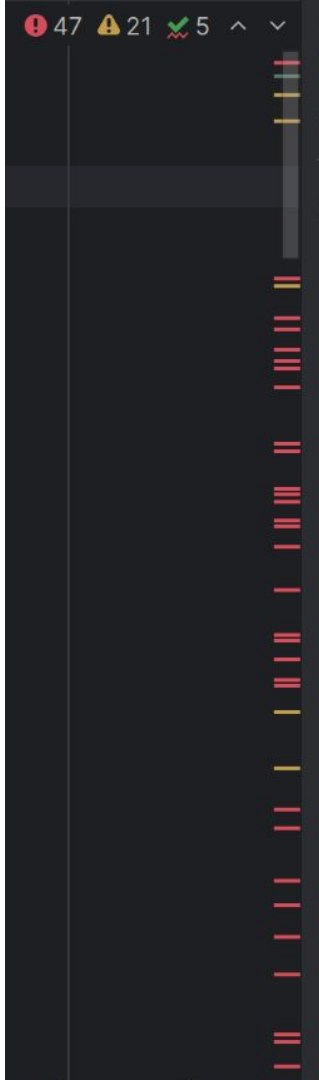
IntelliJ users!

- `gradle generateProto` will generate the protocol buffer files but may not be recognized by IntelliJ because it may not be able to locate them. Add this block of code into your build.gradle file →

```
sourceSets {
    main {
        java {
            srcDirs 'build/generated/source/proto/main/java/proto'
        }
    }
}
```

The file path needs to point to the generated Protobuf files so make sure the file path you are declaring in the sourceSets block matches the actual path to your generated Protobuf files for your project

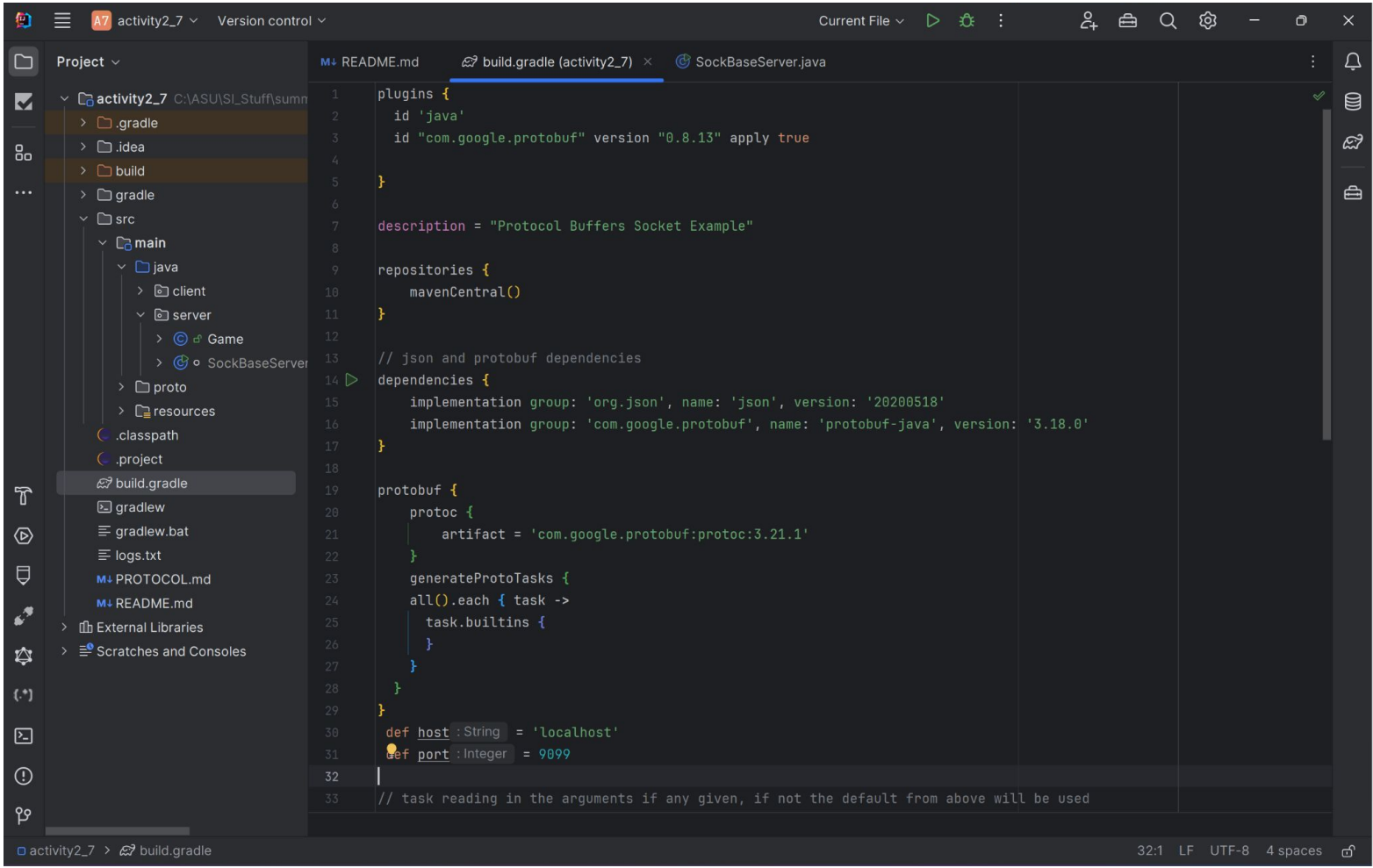
- In the protobuf files, the java_package declaration determines the package name for the generated files and java_outer_classname determines this Protobuf's generated file name → in the given starter code's Protobuf files the package name is declared as "proto" so generated Protobuf files will be placed in a package called "proto" (under build/generated/source/proto/main/java)



4-2 Starter Code

SER 321
Protobufs

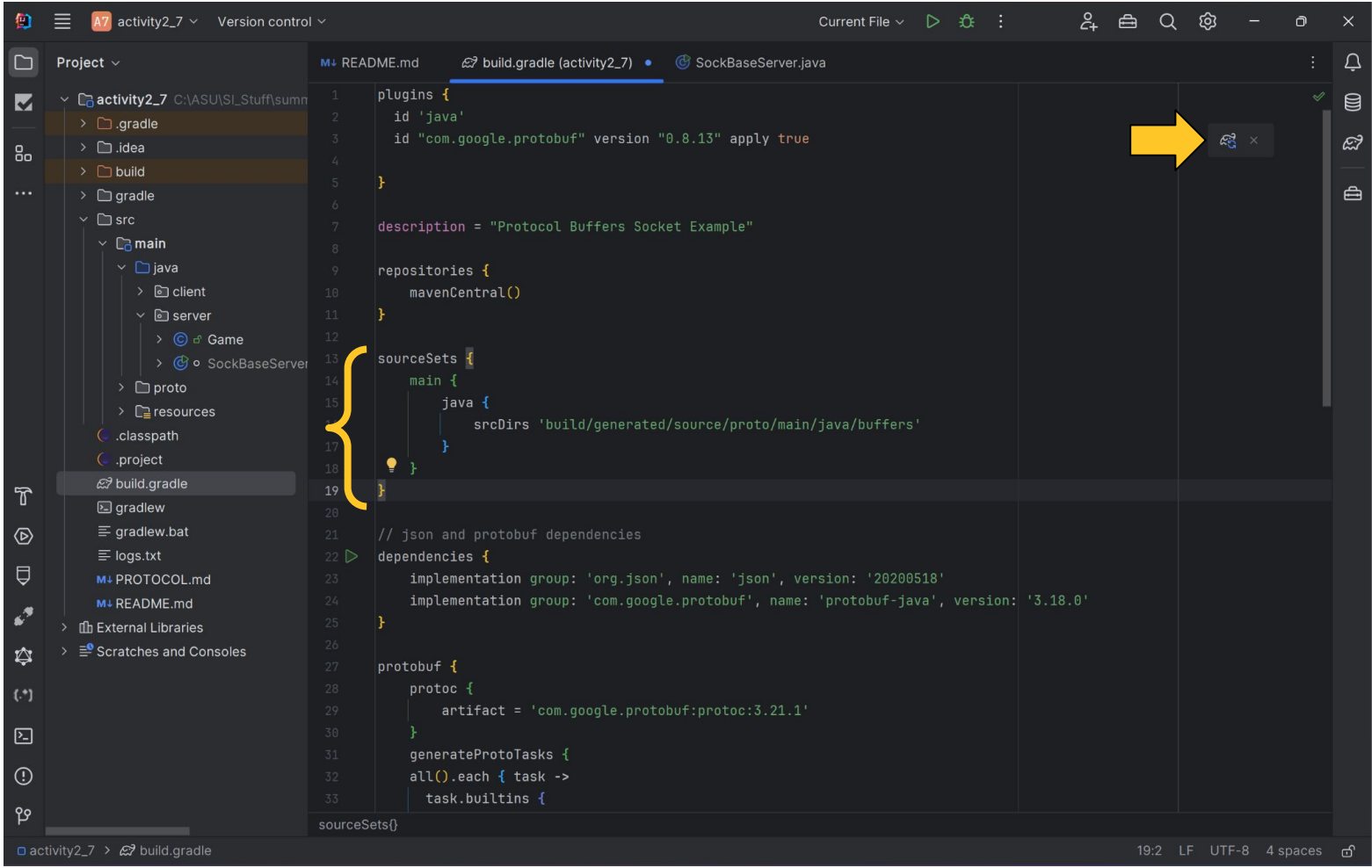
Step 1



4-2 Starter Code

SER 321
Protobufs

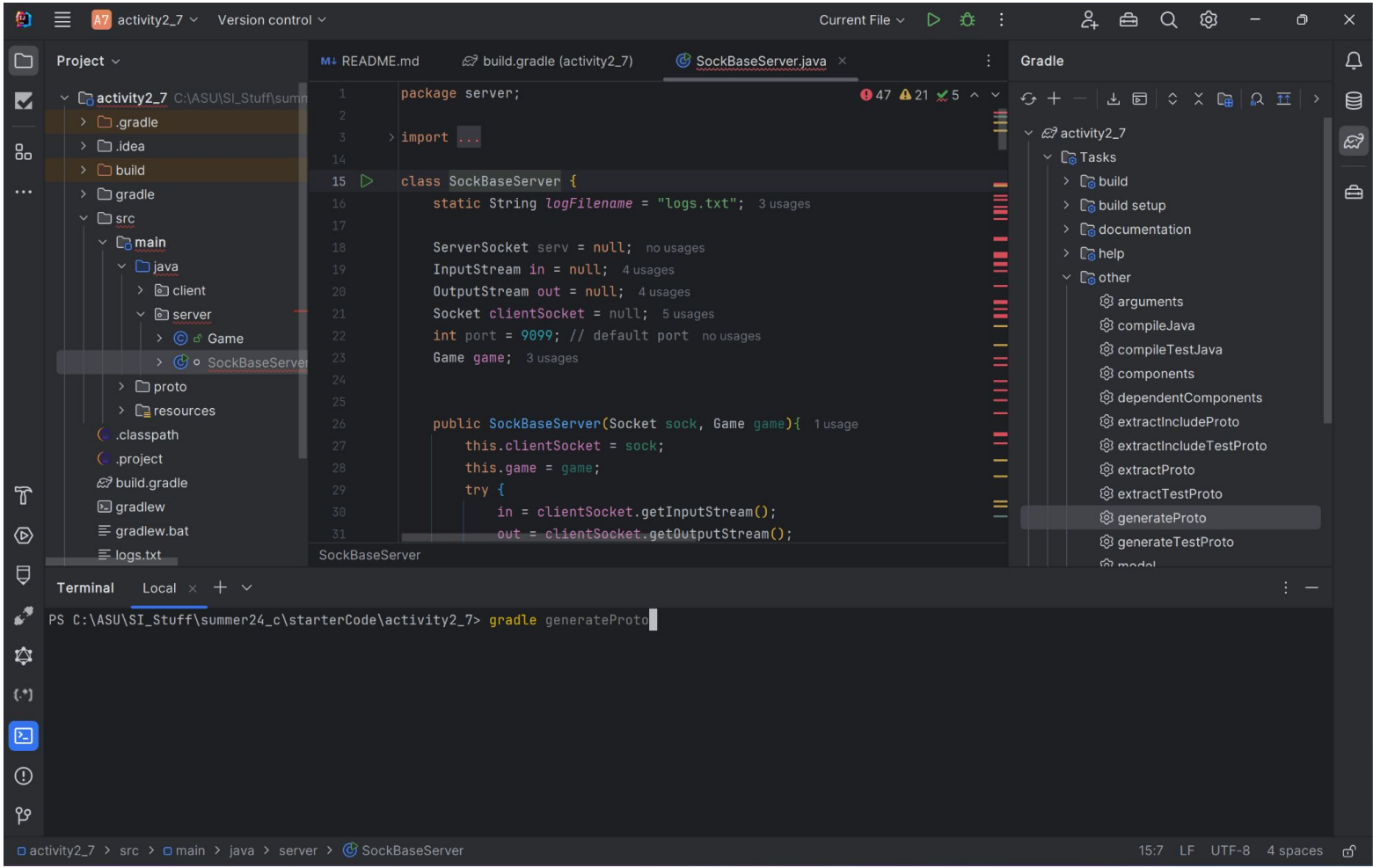
Step 1



4-2 Starter Code

SER 321
Protobufs

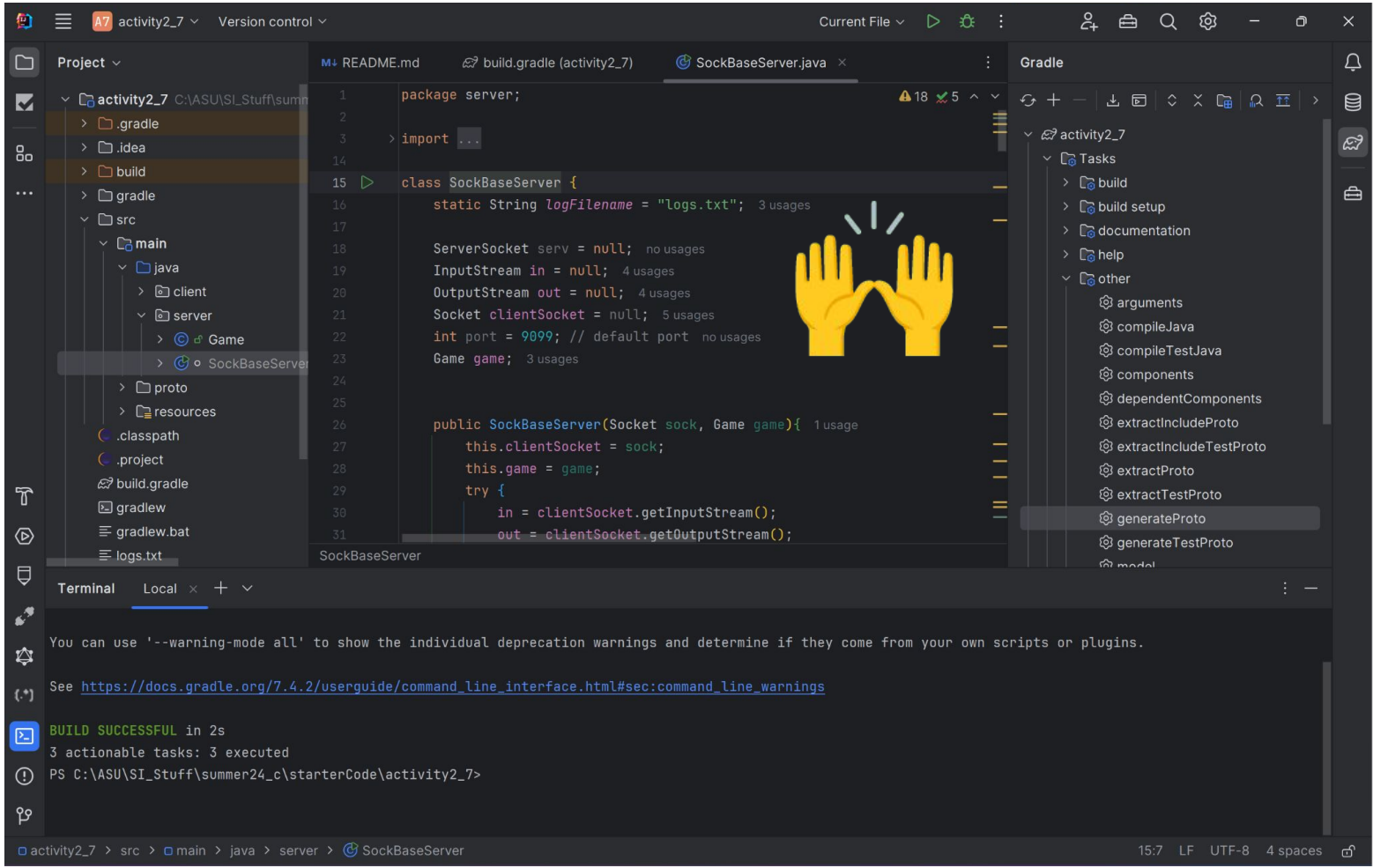
Step 2



4-2 Starter Code

SER 321
Protobufs

Step 2



4-2 Starter Code

SER 321
Protobufs

Options for Message Creation

```
Response response2 = Response.newBuilder()
    .setResponseType(Response.ResponseType.START)
    .setBoard(game.getImage())
    .setMessage("Select a row and column.")
    .build();
```

Create the message in a single statement

Create the message in increments

```
Response.Builder res = Response.newBuilder()
    .setResponseType(Response.ResponseType.LEADERBOARD);
res.addLeader(leader);
res.addLeader(leader2);
Response response3 = res.build();
```

4-2 Starter Code

SER 321
Protobufs

Options for Parsing Messages

```
System.out.println("Type: " + response2.getResponseType());
System.out.println("Board: \n" + response2.getBoard());
System.out.println("Task: \n" + response2.getMessage());
```

Fetch a single value

Fetch a *repeated* value

```
for (Entry lead: response3.getLeaderList()){
    System.out.println(lead.getName() + ": " + lead.getPoints());
}
```

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
```

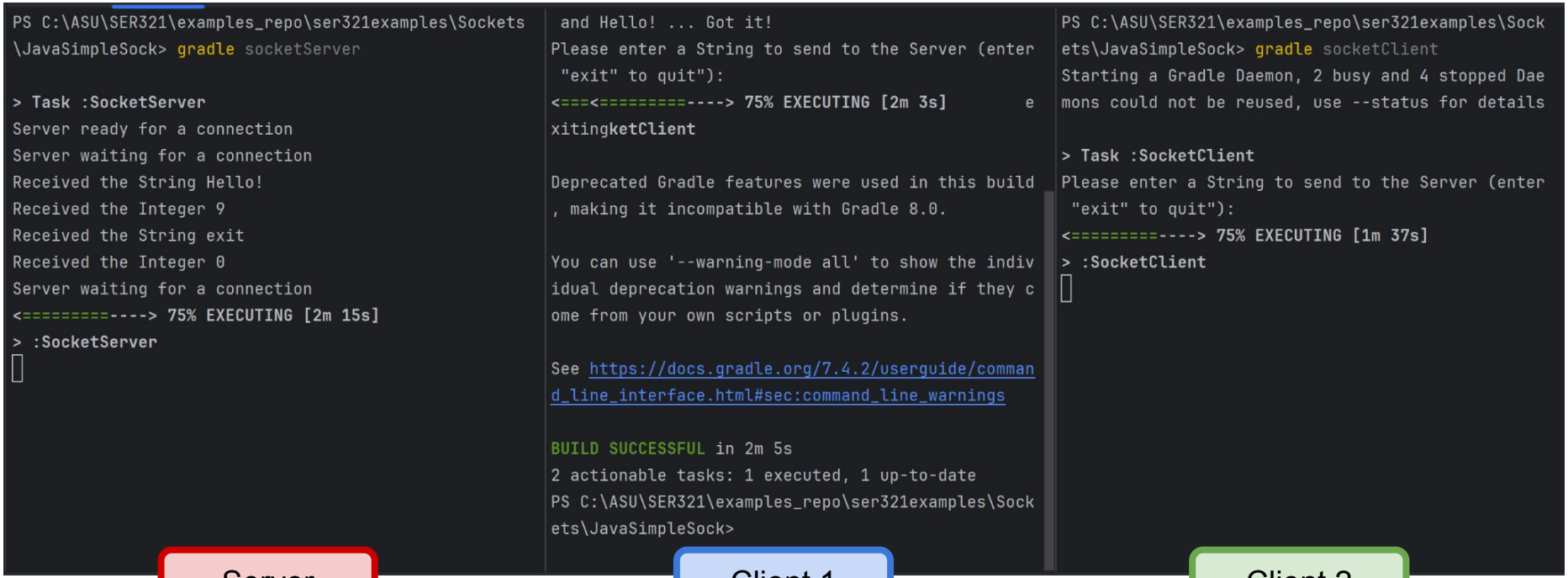
Client 2

What do we think will happen?

JavaSimpleSock

SER 321

Single Threaded Server



Server

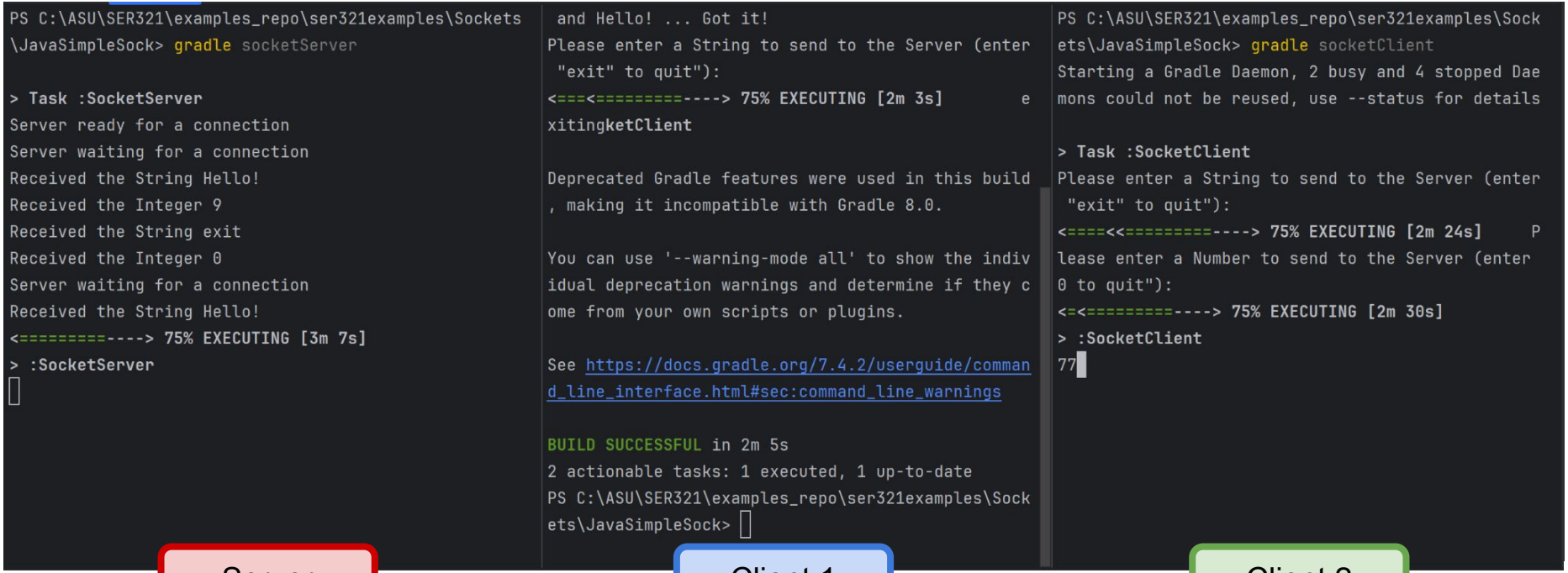
Client 1

Client 2

JavaSimpleSock

SER 321

Single Threaded Server

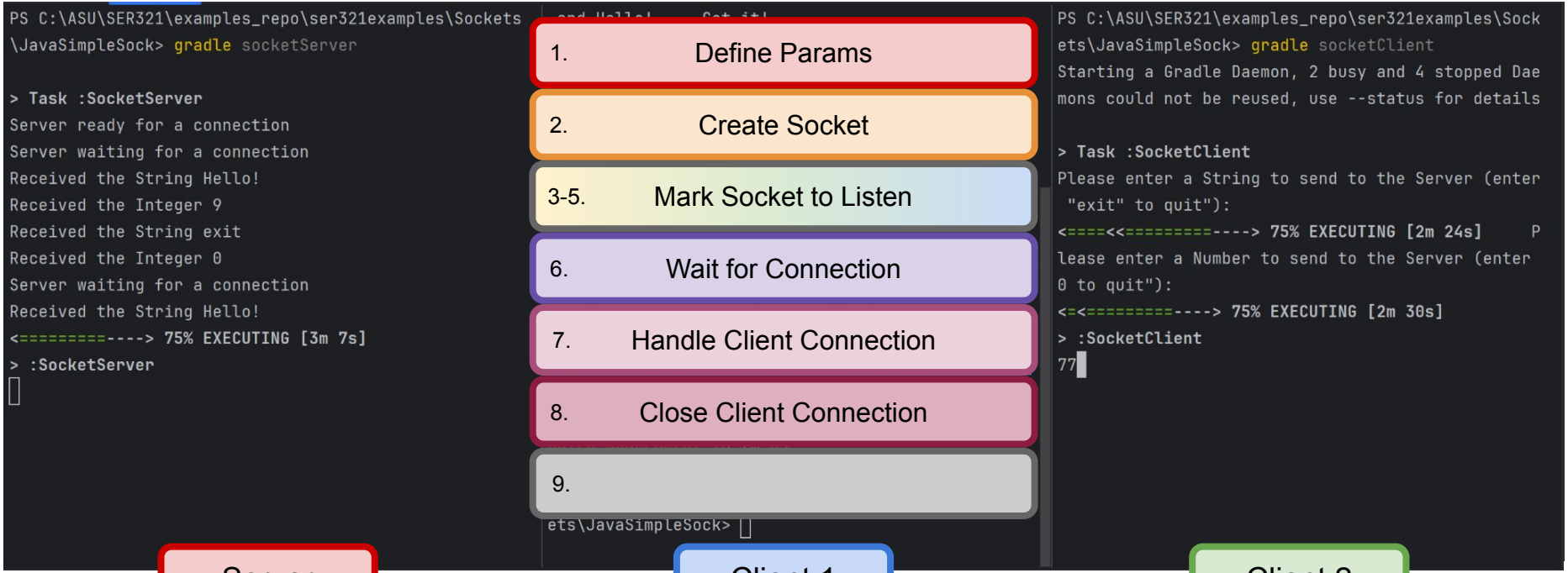


JavaSimpleSock

SER 321

Single Threaded Server

Why?



Server

Client 1

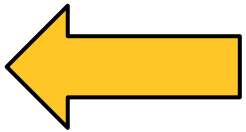
Client 2

SER 321
Threaded Server

Given the standard server socket steps…

Ideas on how we could introduce threads?

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



7. Send Client Socket to thread

JavaThreadSock

SER 321
Threads

1. Define Params
2. Create Socket
3-5. Mark Socket to Listen
6. Wait for Connection
7. Send Client **Socket** to Thread
8. Close Client Connection
9. Continue Listening

```
public static void main(String args[]) throws IOException {
  Socket sock = null;
  int id = 0;
  try {
    if (args.length != 1) {
      System.out.println
          ("Usage: gradle ThreadedSockServer --args=<port num>");
      System.exit( code: 0);
    }
    int portNo = Integer.parseInt(args[0]);
    if (portNo <= 1024)
      portNo = 8888;
    ServerSocket serv = new ServerSocket(portNo);

    while (true) {
      System.out.println
          ("Threaded server waiting for connects on port " + portNo);
      sock = serv.accept();
      System.out.println
          ("Threaded server connected to client-" + id);
      // create thread
      ThreadedSockServer myServerThread =
          new ThreadedSockServer(sock, id++);
      // run thread and don't care about managing it
      myServerThread.start();
    }
  } catch (Exception e) {
    e.printStackTrace();
  } finally {
    if (sock != null) sock.close();
  }
}
}
```

1

2 & 3-5

9

6

7

8

JavaThreadSock

SER 321

Threads

```
public void run() {
  try {
    // setup read/write channels for connection
    ObjectInputStream in = new ObjectInputStream(conn.getInputStream());
    ObjectOutputStream out = new ObjectOutputStream(conn.getOutputStream());

    // read the digit being send
    String s = (String) in.readObject();
    int index;
    // while client hasn't ended
    while (!s.equals("end")) {
      Boolean validInput = true;

      // checks if input only contains digits
      if (!s.matches( expr: "\\d+")) {
        validInput = false;
        out.writeObject("Not a number: https://qph.is/2yDymkn");
      }
      // if it contains only numbers
      if (validInput) {
        // convert to an integer
        index = Integer.valueOf(s);
        System.out.println("From client " + id + " get string " + index);
        if (index > -1 & index < buf.length) {
          // if valid, pull the line from the buffer array above and write it to socket
          out.writeObject(buf[index]);
        } else if (index == 5) {
          // fun surprise for mostly correct
          out.writeObject("Close but out of range: https://youtu.be/dQw4w9WgXcQ");
        } else {
          // really wrong
          out.writeObject("index out of range");
        }
      }
      //  wait for next token from the user
      s = (String) in.readObject();
    }
    // on close, clean up
    System.out.println("Client " + id + " closed connection.");
    in.close();
    out.close();
    conn.close();
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

Client A

Server

```
public static void main(String args[]) throws IOException {
  Socket sock = null;
  int id = 0;
  try {
    if (args.length != 1) {
      System.out.println
          ("Usage: gradle ThreadedSockServer --args=<port num>");
      System.exit( code: 0);
    }
    int portNo = Integer.parseInt(args[0]);
    if (portNo <= 1024)
      portNo = 8888;
    ServerSocket serv = new ServerSocket(portNo);

    while (true) {
      System.out.println
          ("Threaded server waiting for connects on port " + portNo);
      sock = serv.accept();
      System.out.println
          ("Threaded server connected to client-" + id);
      // create thread
      ThreadedSockServer myServerThread =
          new ThreadedSockServer(sock, id++);
      // run thread and don't care about managing it
      myServerThread.start();
    }
  } catch (Exception e) {
    e.printStackTrace();
  } finally {
    if (sock != null) sock.close();
  }
}
```

JavaThreadSock

SER 321

Threads

```
public void run() {
  try {
    // setup read/write channels for connection
    ObjectInputStream in = new ObjectInputStream(conn.getInputStream());
    ObjectOutputStream out = new ObjectOutputStream(conn.getOutputStream
    // read the digit being send
    String s = (String) in.readObject();
    int index;
    // while client hasn't ended
    while (!s.equals("end")) {
      Boolean validInput = true;
      // checks if input only contains digits
      if (!s.matches( expr: "\\d+")) {
        validInput = false;
        out.writeObject("Not a number: https://qph.is/2yDymkn");
      }
      // if it contains only numbers
      if (validInput) {
        // convert to an integer
        index = Integer.valueOf(s);
        System.out.println("From client " + id + " get string " + index);
        if (index > -1 & index < buf.length) {
          // if valid, pull the line from the buffer array above and write it to socket
          out.writeObject(buf[index]);
        } else if (index == 5) {
          // fun surprise for mostly correct
          out.writeObject("Close but out of range: https://youtu.be/dQw4w9WgXcQ");
        } else {
          // really wrong
          out.writeObject("index out of range");
        }
      }
      //  wait for next token from the user
      s = (String) in.readObject();
    }
    // on close, clean up
    System.out.println("Client " + id + " closed connection.");
    in.close();
    out.close();
    conn.close();
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

Client A

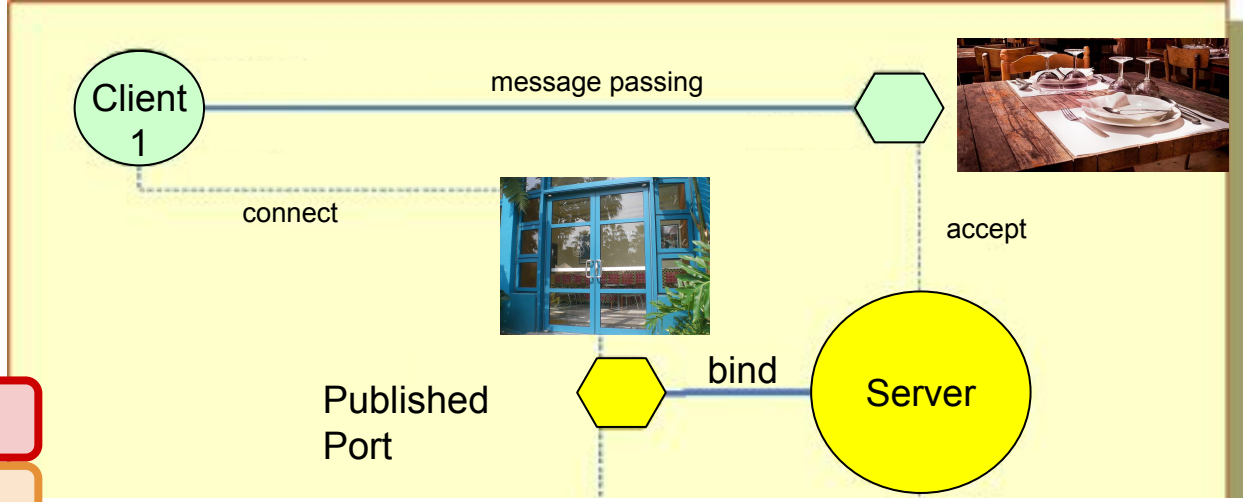
Client B

Server

```
public static void main(String args[]) throws IOException {
  Socket sock = null;
  int id = 0;
  try {
    if (args.length != 1) {
      System.out.println
          ("Usage: gradle ThreadedSockServer --args=<port num>");
      System.exit( code: 0);
    }
    int portNo = Integer.parseInt(args[0]);
    if (portNo <= 1024)
      portNo = 8888;
    ServerSocket serv = new ServerSocket(portNo);

    while (true) {
      System.out.println
          ("Threaded server waiting for connects on port " + portNo);
      sock = serv.accept();
      System.out.println
          ("Threaded server connected to client-" + id);
      // create thread
      ThreadedSockServer myServerThread =
          new ThreadedSockServer(sock, id++);
      // run thread and don't care about managing it
      myServerThread.start();
    }
  } catch (Exception e) {
    e.printStackTrace();
  } finally {
    if (sock != null) sock.close();
  }
}
```

SER 321
Threads



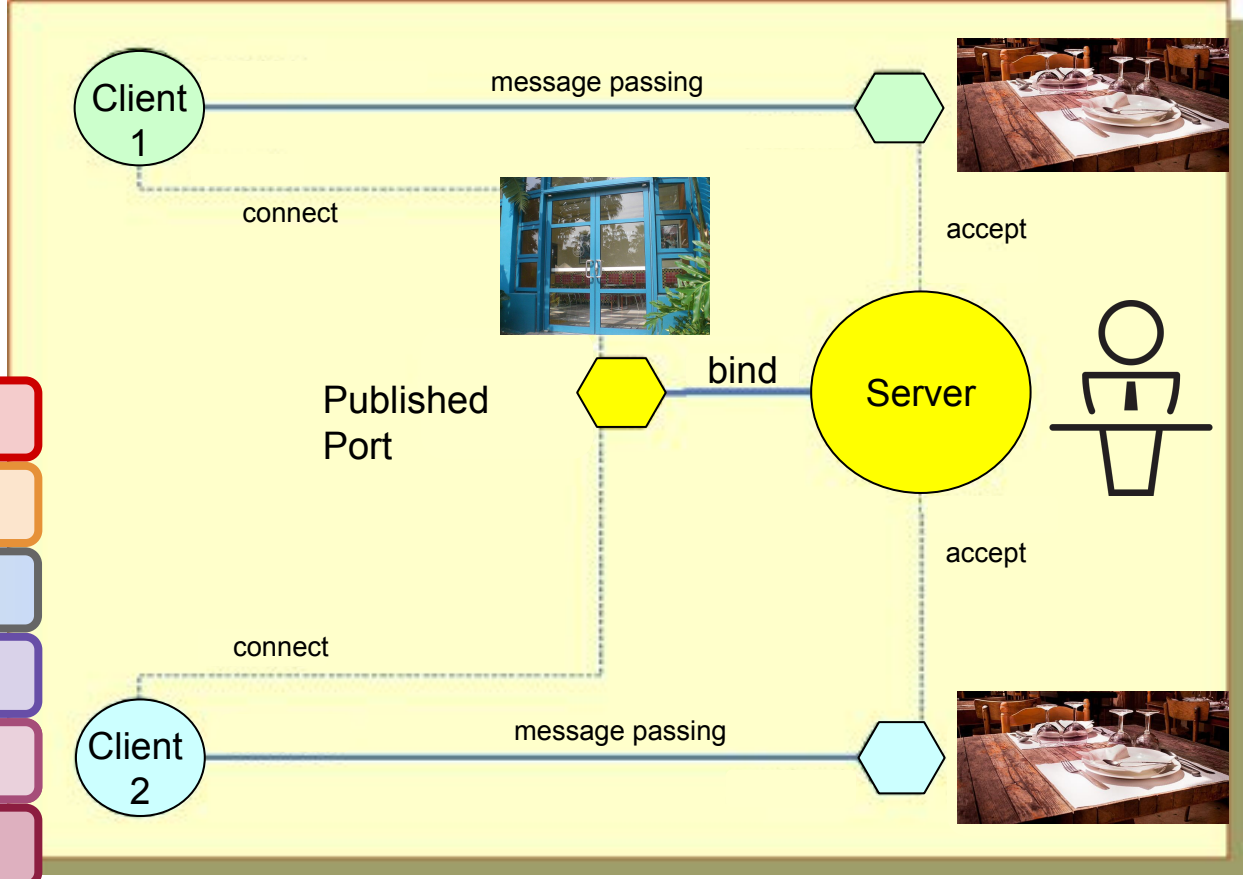
1. Define Params
2. Create Socket
3-5. Mark Socket to Listen
6. Wait for Connection
7. Send Client **Socket** to Thread
8. Close Client Connection
9. Continue Listening

Why do we send the client **socket** to the thread?

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
Threads

1. Define Params
2. Create Socket
3-5. Mark Socket to Listen
6. Wait for Connection
7. Send Client **Socket** to Thread
8. Close Client Connection
9. Continue Listening



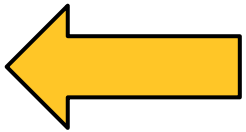
Design of an RFID Vehicle Authentication System: A Case Study for Al-Nahrain University Campus - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Client-and-Server-Socket-Ports_fig4_282671198

SER 321
Threaded Server

Now What?

Handle the Client just like before!

1. Define Params
2. Create Socket
3-5. Mark Socket to Listen
6. Wait for Connection
7. Send Client **Socket** to Thread
8. Close Client Connection
9. Continue Listening



1

2

3

4

5

SER 321

Scratch Space

Questions?

Survey:

http://bit.ly/ASN2324

26

Upcoming Events

SI Sessions:

- Thursday, June 20th at 6:00 pm MST
- Sunday, June 23rd at 6:00 pm MST
- Monday, June 24th at 6:00 pm MST

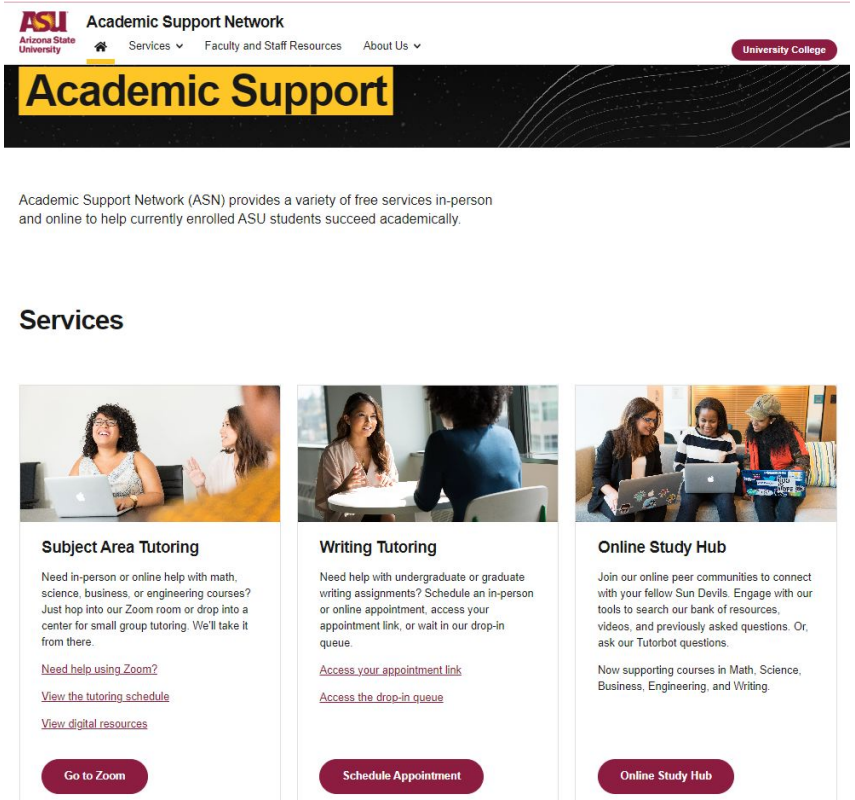
Review Sessions:

- Review Session - **Wednesday**, July 3rd at 6:00 pm MST (2 hr Session)
- Q&A Session - Sunday, July 7th at 6:00 pm MST (Final Session)

More Questions?

Check out our other resources!

tutoring.asu.edu

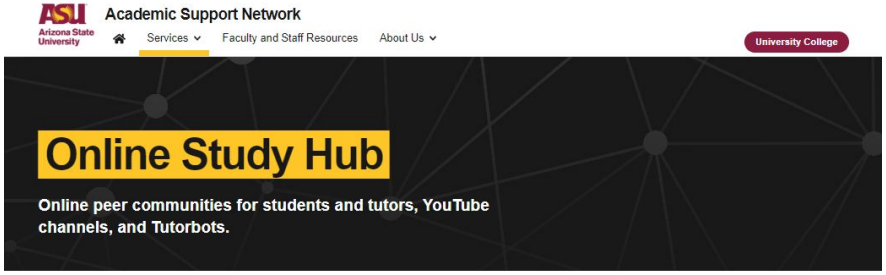


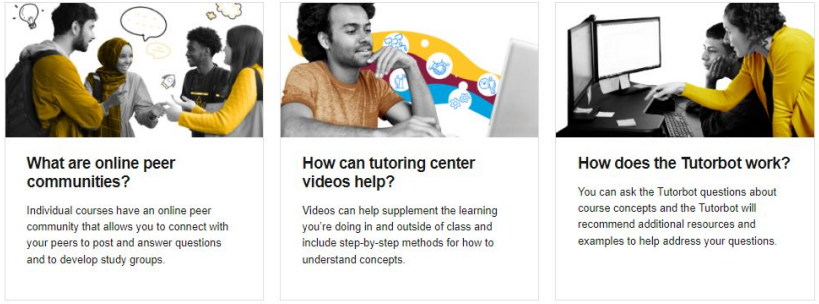
1. **Click on ‘Go to Zoom’ to log onto our Online Tutoring Center.**
2. **Click on ‘View the tutoring schedule’ to see when tutors are available for specific courses.**

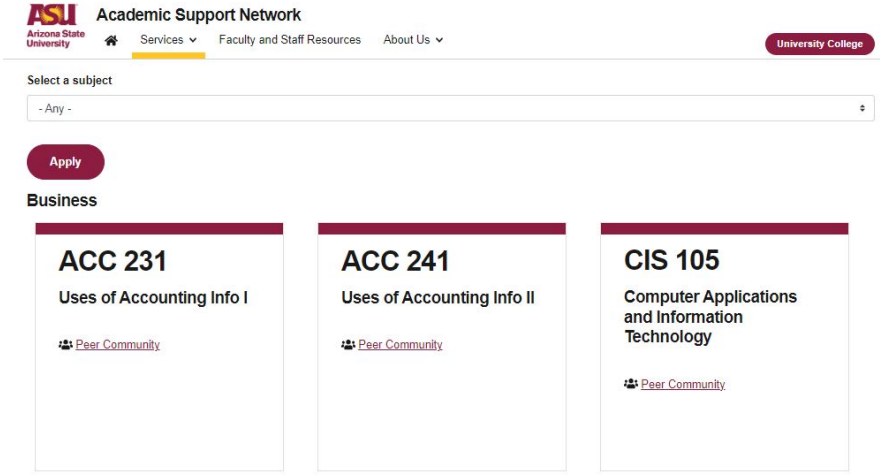
More Questions?

Check out our other resources!

tutoring.asu.edu/online-study-hub



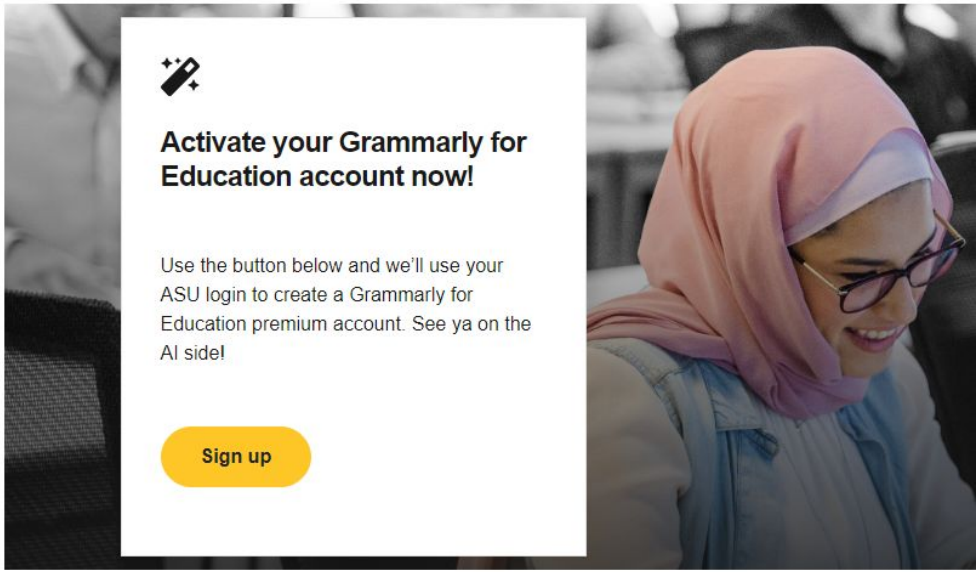




Don't forget to check out the Online Study Hub for additional resources!

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
- Dining Philosophers Interactive
- Austin G Walters Traffic Comparison