

# SER 321 A Session

**SI Session**

**Sunday, September 10th 2023**

*6:00 - 7:00 pm MST*

# Agenda



Session Expectations

Gradle Review

Threading

Threading Pitfalls

Synchronization

# 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](https://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 A Session

## Gradle Review

build.gradle from Assign 4 Act 1  
starter code

```
▶ task runServer(type: JavaExec) {  
    group 'server'  
    description 'Creates Server socket waits for messages'  
  
    classpath = sourceSets.main.runtimeClasspath  
  
    main = 'taskone.Server'  
    standardInput = System.in  
  
    if (project.hasProperty('port')) {  
        | args(project.getProperty('port'));  
    }  
}
```

# SER 321 A Session

## Gradle Review

build.gradle from Assign 4 Act 1  
starter code

```
▶ task runServer(type: JavaExec) {  
    group 'server'  
    description 'Creates Server socket waits for messages'  
  
    classpath = sourceSets.main.runtimeClasspath  
  
    main = 'taskone.Server'  
    standardInput = System.in  
  
    if (project.hasProperty('port')) {  
        | args(project.getProperty('port'));  
    }  
}
```

# SER 321 A Session

## Gradle Review

build.gradle from Assign 4 Act 1  
starter code

Can we do this?

```
gradle runServer
```

```
task runServer(type: JavaExec) {  
    group 'server'  
    description 'Creates Server socket waits for messages'  
  
    classpath = sourceSets.main.runtimeClasspath  
  
    main = 'taskone.Server'  
    standardInput = System.in  
  
    if (project.hasProperty('port')) {  
        args(project.getProperty('port'));  
    }  
}
```

# SER 321 A Session

## Gradle Review

build.gradle from Assign 4 Act 1  
starter code

Can we do this?

```
task runServer(type: JavaExec) {  
    group 'server'  
    description 'Creates Server socket waits for messages'  
  
    classpath = sourceSets.main.runtimeClasspath  
  
    main = 'taskone.Server'  
    standardInput = System.in  
  
    if (project.hasProperty('port')) {  
        args(project.getProperty('port'));  
    }  
}
```

```
gradle runServer
```

> Task :runServer FAILED

Usage: gradle runServer -Pport=9099 -q --console=plain



# SER 321 A Session

## Gradle Review

Which ones correctly set the default arguments?

A

```
if (project.hasProperty('port')) {  
    args(project.getProperty('port'));  
}  
args 8000;
```

B

```
if (project.hasProperty('port')) {  
    args(project.getProperty('port'));  
} else {  
    args 8000;  
}
```

C

```
args 8000;  
if (project.hasProperty('port')) {  
    args(project.getProperty('port'));  
}
```

D. NONE

*Check out the recording for the solution!*

# SER 321 A Session

## Threading

### Things to remember:

- Threads *share* resources of the parent process - including memory!
- *Less* protection than processes
- Allows for greater responsiveness
- Complex to debug

But hope is not lost!

# SER 321 A Session

## Socket Server - No Threads

Make Socket

Wait for connections

Handle the connection

Perform the task

Clean up - what is that again?

`in.close();`

`out.close();`

`sock.close();`

```
public static void main (String args[]) {  
    Socket sock;  
    try {  
        //open socket  
        ServerSocket serv = new ServerSocket( port 8888); // create server socket on port 8888  
        System.out.println("Server ready for 3 connections");  
        // only does three connections then closes  
        // NOTE: SINGLE-THREADED, only one connection at a time  
        for (int rep = 0; rep < 3; rep++){  
            System.out.println("Server waiting for a connection");  
            sock = serv.accept(); // blocking wait  
            // setup the object reading channel  
            ObjectInputStream in = new ObjectInputStream(sock.getInputStream());  
  
            // read in one object, the message. we know a string was written only by knowing what the client sent.  
            // must cast the object from Object to desired type to be useful  
            String s = (String) in.readObject();  
            System.out.println("Received the String "+s);  
            // read in the number, we know it's an integer because that's the second thing sent by the client.  
            Integer i = (Integer) in.readObject();  
            System.out.println("Received the Integer "+ i);  
  
            // generate an output  
            // get output channel  
            OutputStream out = sock.getOutputStream();  
            // create an object output writer (Java only)  
            ObjectOutputStream os = new ObjectOutputStream(out);  
            // write the whole message  
            os.writeObject("Got it!");  
            // make sure it wrote and doesn't get cached in a buffer  
            os.flush();  
        }  
    } catch (Exception e) {e.printStackTrace();}  
}
```

[SockServer](#) from [JavaSimpleSock2](#) in [examples Repo](#)

# SER 321 A Session

## Threading your Server

Make Socket

Wait for connections

Start Thread

Handle the connection

Perform the task

Clean up

JavaThreadedSock in Sockets

```
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();
}
```

```
public ThreadedSockServer(Socket sock, int id) {
    this.conn = sock;
    this.id = id;
}
```

```
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();
    }
}
```

# SER 321 A Session

## Threading

Make Socket

Wait for connections

Start Thread

Handle the connection

Perform the task

Clean up

```
in.close();  
out.close();  
conn.close();
```

```
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://gph.is/2vDymkn");  
    }  
  
    // 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();  
}
```

# SER 321

## Threading Pitfalls

- Race Condition
- Starvation
- Deadlock
- One thread never gets access to the resource it needs
- A thread is only able to acquire access to part of its resources
- More than one thread accesses a single resource at one time

*Check out the recording for the solution!*

# SER 321

## Synchronization

- Locks
- Synchronization

# SER 321

## Synchronization

- Locks
- Synchronization

```
public LockThread (int id, Lock mutex, int sd, int lc) {  
    this.id = id;  
    this.mutex = mutex;           mutex.lock();  
    this.loopCount = lc;          Danger zone  
    this.sleepDelay = sd;         mutex.unlock();  
}
```



# SER 321

## Synchronization

- Locks
- Synchronization

```
public LockThread (int id, Lock mutex, int sd, int lc) {  
    this.id = id;  
    this.mutex = mutex;           mutex.lock();  
    this.loopCount = lc;          Danger zone  
    this.sleepDelay = sd;         mutex.unlock();  
}
```

Locks

```
public synchronized void bow(Friend bower) {
```

# SER 321

## Synchronization and Deadlocks

This is Deadlock from the examples repo

Bow and BowBack are both synchronized

What happens when we run this as is?

```
public class Deadlock {
    static class Friend {
        private final String name;
        public Friend(String name) {
            this.name = name;
        }
        public String getName() {
            return this.name;
        }
        /* See the README.md for a reference on 'synchronized' methods */
        public synchronized void bow(Friend bower) {
            System.out.format("%s: %s"
                + " has bowed to me!\n",
                this.name, bower.getName());
            System.out.format("%s: waiting to bow back\n", bower.getName());
            bower.bowBack(this);
        }
        public synchronized void bowBack(Friend bower) {
            System.out.format("%s: waiting\n", this.name);
            System.out.format("%s: %s"
                + " has bowed back to me!\n",
                this.name, bower.getName());
        }
    }

    public static void main(String[] args) {
        final Friend alphonse =
            new Friend("Alphonse");
        final Friend gaston =
            new Friend("Gaston");
        /* start two threads - both operating on the same objects */
        new Thread(new Runnable() {
            public void run() { alphonse.bow(gaston); }
        }).start();
        new Thread(new Runnable() {
            public void run() { gaston.bow(alphonse); }
        }).start();
    }
}
```

# SER 321

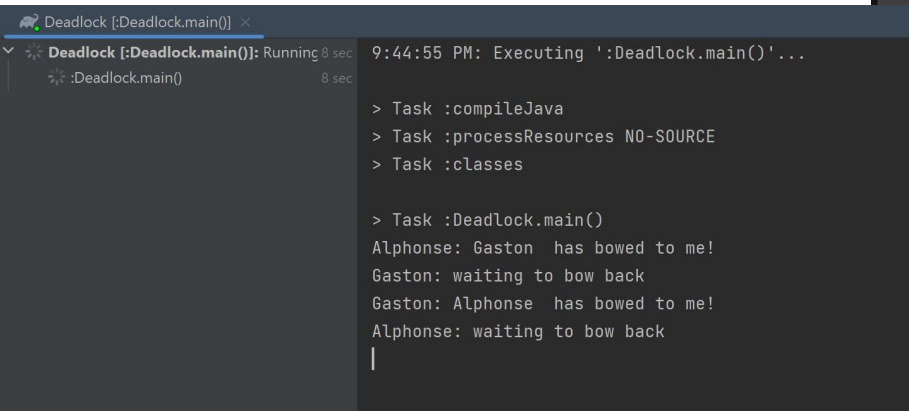
## Synchronization and Deadlocks

This is Deadlock from the examples repo

Bow and BowBack are both synchronized

What happens when we run this as is?

```
public class Deadlock {  
    static class Friend {  
        private final String name;  
        public Friend(String name) {  
            this.name = name;  
        }  
        public String getName() {  
            return this.name;  
        }  
        /* See the README.md for a reference on 'synchronized' methods */  
        public synchronized void bow(Friend bower) {  
            System.out.format("%s: %s" +  
                " has bowed to me!\n",  
                    this.name, bower.getName());  
            System.out.format("%s: waiting to bow back\n", bower.getName());  
            bower.bowBack(this);  
        }  
        public synchronized void bowBack(Friend bower) {  
            System.out.format("%s: waiting\n", this.name);  
            System.out.format("%s: %s" +  
                " has bowed back to me!\n",  
                    this.name, bower.getName());  
        }  
    }  
    public static void main(String[] args) {  
        final Friend alphonse =  
            new Friend("Alphonse");  
        final Friend gaston =  
            new Friend("Gaston");  
        /* start two threads - both operating on the same objects */  
        new Thread(new Runnable() {  
            public void run() { alphonse.bow(gaston); }  
        }).start();  
        new Thread(new Runnable() {  
            public void run() { gaston.bow(alfonse); }  
        }).start();  
    }  
}
```



```
Deadlock [Deadlock.main()] x  
Deadlock [Deadlock.main()]: Running 8 sec 9:44:55 PM: Executing 'Deadlock.main()'...  
Deadlock.main() 8 sec  
Task :compileJava  
Task :processResources NO-SOURCE  
Task :classes  
Task :Deadlock.main()  
Alphonse: Gaston has bowed to me!  
Gaston: waiting to bow back  
Gaston: Alphonse has bowed to me!  
Alphonse: waiting to bow back  
|
```

# SER 321

## Synchronization and Deadlocks

This is Deadlock from the examples repo

Synchronizing **methods** is not always the  
best solution

How do we fix it?

```
public class Deadlock {  
    static class Friend {  
        private final String name;  
        public Friend(String name) {  
            this.name = name;  
        }  
        public String getName() {  
            return this.name;  
        }  
        /* See the README.md for a reference on 'synchronized' methods */  
        public synchronized void bow(Friend bower) {  
            System.out.format("%s: %s"  
                + " has bowed to me!\n",  
                    this.name, bower.getName());  
            System.out.format("%s: waiting to bow back\n", bower.getName());  
            bower.bowBack(this);  
        }  
        public synchronized void bowBack(Friend bower) {  
            System.out.format("%s: waiting\n", this.name);  
            System.out.format("%s: %s"  
                + " has bowed back to me!\n",  
                    this.name, bower.getName());  
        }  
    }  
  
    public static void main(String[] args) {  
        final Friend alphonse =  
            new Friend("Alphonse");  
        final Friend gaston =  
            new Friend("Gaston");  
        /* start two threads - both operating on the same objects */  
        new Thread(new Runnable() {  
            public void run() { alphonse.bow(gaston); }  
        }).start();  
        new Thread(new Runnable() {  
            public void run() { gaston.bow(alfonse); }  
        }).start();  
    }  
}
```

# SER 321

## Synchronization and Deadlocks

This is Deadlock from the examples repo

Synchronizing **methods** is not always the  
best solution

How do we fix it?

Can remove the synchronized keyword, but  
that would allow a race condition...

```
public class Deadlock {  
    static class Friend {  
        private final String name;  
        public Friend(String name) {  
            this.name = name;  
        }  
        public String getName() {  
            return this.name;  
        }  
        /* See the README.md for a reference on 'synchronized' methods */  
        public synchronized void bow(Friend bower) {  
            System.out.format("%s: %s"  
                + " has bowed to me!\n",  
                    this.name, bower.getName());  
            System.out.format("%s: waiting to bow back\n", bower.getName());  
            bower.bowBack(this);  
        }  
        public synchronized void bowBack(Friend bower) {  
            System.out.format("%s: waiting\n", this.name);  
            System.out.format("%s: %s"  
                + " has bowed back to me!\n",  
                    this.name, bower.getName());  
        }  
    }  
  
    public static void main(String[] args) {  
        final Friend alphonse =  
            new Friend("Alphonse");  
        final Friend gaston =  
            new Friend("Gaston");  
        /* start two threads - both operating on the same objects */  
        new Thread(new Runnable() {  
            public void run() { alphonse.bow(gaston); }  
        }).start();  
        new Thread(new Runnable() {  
            public void run() { gaston.bow(alfonse); }  
        }).start();  
    }  
}
```

# SER 321

## Synchronization and Deadlocks

This is Deadlock from the examples repo

Synchronizing **methods** is not always the  
best solution

How do we fix it?

Synchronized *statement* allows us to

synchronize without attempting to call

two synchronized methods on the same

object

```
public class Deadlock {
    static class Friend {
        private final String name;
        public Friend(String name) {
            this.name = name;
        }
        public String getName() {
            return this.name;
        }
        /* See the README.md for a reference on 'synchronized' methods */
        public synchronized void bow(Friend bower) {
            System.out.format("%s: %s"
                + " has bowed to me!\n",
                this.name, bower.getName());
            System.out.format("%s: waiting to bow back\n", bower.getName());
            bower.bowBack(this);
        }
        public synchronized void bowBack(Friend bower) {
            System.out.format("%s: waiting\n", this.name);
            System.out.format("%s: %s"
                + " has bowed back to me!\n",
                this.name, bower.getName());
        }
    }

    public static void main(String[] args) {
        final Friend alphonse =
            new Friend("Alphonse");
        final Friend gaston =
            new Friend("Gaston");
        /* start two threads - both operating on the same objects */
        new Thread(new Runnable() {
            public void run() { alphonse.bow(gaston); }
        }).start();
        new Thread(new Runnable() {
            public void run() { gaston.bow(alfonse); }
        }).start();
    }
}
```



# SER 321

## Synchronization and Deadlocks

```
> Task :Deadlock.main()
Alphonse: Gaston has bowed to me!
Gaston: waiting to bow back
Gaston: waiting
Gaston: Alphonse has bowed back to me!
Gaston: Alphonse has bowed to me!
Alphonse: waiting to bow back
Alphonse: waiting
Alphonse: Gaston has bowed back to me!
```

```
public static void main(String[] args) {
    final Friend alphonse =
        new Friend("Alphonse");
    final Friend gaston =
        new Friend("Gaston");
    /* start two threads - both operating on the same objects */
    new Thread(new Runnable() {
        public synchronized void run() { alphonse.bow(gaston); }
    }).start();
    new Thread(new Runnable() {
        public synchronized void run() { gaston.bow(alphonse); }
    }).start();
}
```

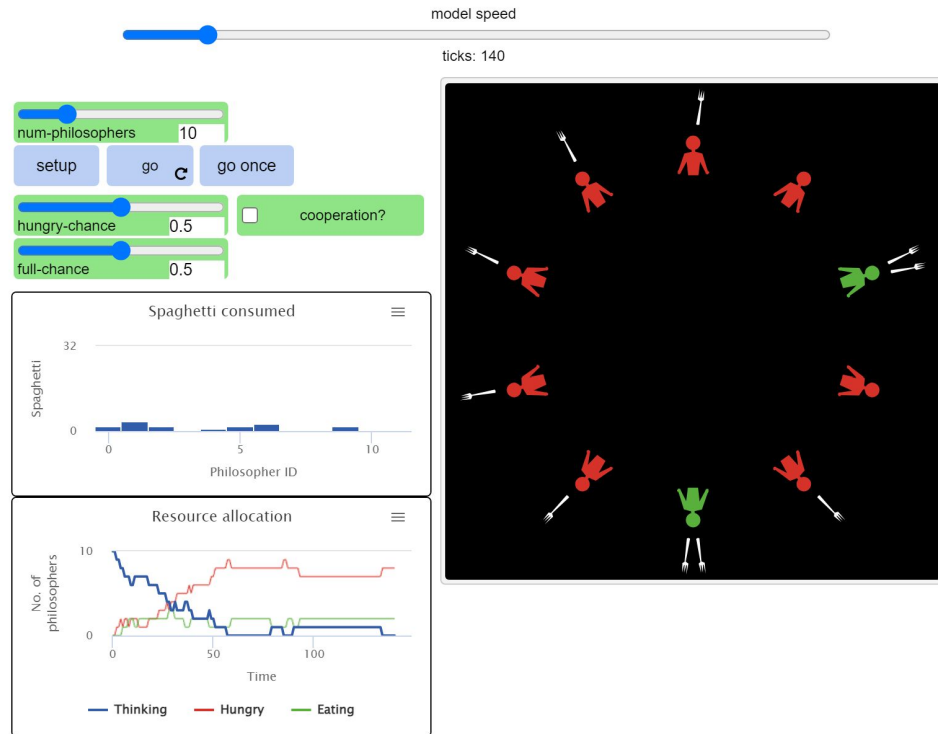
```
public void bow(Friend bower) {
    System.out.format("%s: %s"
        + " has bowed to me!\n",
        this.name, bower.getName());
    System.out.format("%s: waiting to bow back\n", bower.getName());
    synchronized(this) {
        bower.bowBack(this);
    }
}
```

# SER 321

## Remember Dining Philosophers?



## Interactive





# SER 321

## Tomorrow's Agenda

- More Thread Review
  - Anything we didn't cover today
  - More on Race Conditions?
- Review Serialization
- Review Protobufs
- Review Assignment 4 Activity 1

**Have a request? Post here or in the si-channel on slack and I'll be sure to spend some extra time on it tomorrow!**

# Questions?

## Survey:

[https://bit.ly/asn\\_survey](https://bit.ly/asn_survey)



## Upcoming Events

### SI Sessions:

- Tomorrow Monday September 11th, 6:00 pm MST

### Review Sessions:

- TBD

# More Questions?

Check out our other resources!

tutoring.asu.edu



## Academic Support

Academic Support Network (ASN) provides a variety of free services in-person and online to help currently enrolled ASU students succeed academically.

### Services



#### Subject Area Tutoring

Need in-person or online help with math, science, business, or engineering courses? Just hop into our Zoom room or drop into a center for small group tutoring. We'll take it from there.

[Need help using Zoom?](#)

[View the tutoring schedule](#)

[View digital resources](#)

Go to Zoom



#### Writing Tutoring

Need help with undergraduate or graduate writing assignments? Schedule an in-person or online appointment, access your appointment link, or wait in our drop-in queue.

[Access your appointment link](#)

[Access the drop-in queue](#)

Schedule Appointment



#### Online Study Hub

Join our online peer communities to connect with your fellow Sun Devils. Engage with our tools to search our bank of resources, videos, and previously asked questions. Or, ask our Tutorbot questions.

Now supporting courses in Math, Science, Business, Engineering, and Writing.

Online Study Hub

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](https://tutoring.asu.edu/online-study-hub)

 **Academic Support Network**

 [Services](#)  [Faculty and Staff Resources](#) [About Us](#) 

[University College](#)

## Online Study Hub

Online peer communities for students and tutors, YouTube channels, and Tutorbots.



### What are online peer communities?

Individual courses have an online peer community that allows you to connect with your peers to post and answer questions and to develop study groups.



### How can tutoring center videos help?

Videos can help supplement the learning you're doing in and outside of class and include step-by-step methods for how to understand concepts.



### How does the Tutorbot work?

You can ask the Tutorbot questions about course concepts and the Tutorbot will recommend additional resources and examples to help address your questions.

Select a subject

- Any -

[Apply](#)



Academic Support Network



[Services](#) 

[Faculty and Staff Resources](#)

[About Us](#) 

[University College](#)

Select a subject

- Any -

[Apply](#)

Business

### ACC 231

Uses of Accounting Info I

 [Peer Community](#)

### ACC 241

Uses of Accounting Info II

 [Peer Community](#)

### CIS 105

Computer Applications and Information Technology

 [Peer Community](#)

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

## Additional Resources

<https://ccl.northwestern.edu/netlogo/models/DiningPhilosophers>

[Examples Repo](#)