# SER 321 A Session

## SI Session

**Sunday, February 9th 2025**

*7:00 pm - 8:00 pm MST*

# Agenda

- Gradle Review
- Threaded Pitfalls Review
- Concurrency Structures
- Threading the Server
  - Where and How
  - Thread Tracing

# 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 Language Correction!

The phrasing I used last session implied that protobufs have built-in error handling…

Protobufs generate the *methods* you will need

BUT you still have to call them!

# SER 321
## Protobufs

```
public void startGame() throws IOException {  1 usage
    try {
        while (true) {
            // read the proto object and put into new object
            Request op = Request.parseDelimitedFrom(in);
            System.out.println("Got request: " + op.toString());
            Response response;

                                    uit = false;

                                    n.getOperationType()) {
```

has the operation type

RequestProtos

```
optional .operation.Request.OperationType operationType = 1 [default = NAME];
```

```
public static buffers.RequestProtos.Request parseDelimitedFrom(java.io.InputStream input)
    throws java.io.IOException {
  return com.google.protobuf.GeneratedMessageV3
      .parseDelimitedWithIOException(PARSER, input);
}
```

Tries to parse, throws exception if it can't

```
optional .operation.Request.OperationType operationType = 1 [default = NAME];
```

Returns: The operationType.

```
@java.lang.Override  3 usages
public buffers.RequestProtos.Request.OperationType getOperationType() {
  /deprecation/
  buffers.RequestProtos.Request.OperationType result = buffers.RequestProtos.Request.OperationType.valueOf(operationType_);
  return result == null ? buffers.RequestProtos.Request.OperationType.NAME : result;
}
```

mplete!

tionType().name());

# SER 321
## Gradle Review

***Check out the recording for the solution!***

Which of the following will run the main method in */java/taskone/Server.java* with `gradle runTask1` ?

A.

```
task runServer(type: JavaExec) {
  group 'server'
  description 'Creates Server socket waits for messages'

  classpath = sourceSets.main.runtimeClasspath

  main = 'taskone.Server.runTask1'
  standardInput = System.in

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

B.

```
task runServer(type: JavaExec) {
  group 'server'
  description 'Creates Server socket waits for messages'

  classpath = sourceSets.main.runtimeClasspath

  main = 'taskone.Server'
  standardInput = System.in

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

C.

```
task1 runServer(type: JavaExec) {
  group 'server'
  description 'Creates Server socket waits for messages'

  classpath = sourceSets.main.runtimeClasspath

  main = 'taskone.Server'
  standardInput = System.in

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

D.

```
task runTask1(type: JavaExec) {
  group 'server'
  description 'Creates Server socket waits for messages'

  classpath = sourceSets.main.runtimeClasspath

  main = 'taskone.Server'
  standardInput = System.in

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

# SER 321
## Gradle Review

***Check out the recording for the solution!***

Which of the following will run the main method in */java/tasktwo/Server.java* with `gradle runTask2` ?

A.

```
task runTask2(type: JavaExec) {
  group 'server'
  description 'Creates Server socket waits for messages'

  classpath = sourceSets.main.runtimeClasspath

  main = 'taskone.Server'
  standardInput = System.in

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

B.

```
task runTask2(type: JavaExec) {
  group 'server'
  description 'Creates Server socket waits for messages'

  classpath = sourceSets.main.runtimeClasspath

  main = 'tasktwo.Server'
  standardInput = System.in

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

C.

```
task2 runServer(type: JavaExec) {
  group 'server'
  description 'Creates Server socket waits for messages'

  classpath = sourceSets.main.runtimeClasspath

  main = 'tasktwo.Server'
  standardInput = System.in

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

D.

```
task runServer(type: JavaExec) {
  group 'server'
  description 'Creates Server socket waits for messages'

  classpath = sourceSets.main.runtimeClasspath

  main = 'tasktwo.Server'
  standardInput = System.in

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

# SER 321
## Gradle Review

***Check out the recording for the solution!***

Which of the following will run the main method in */java/taskone/Client.java* with `gradle runClient` ?

A.

```
task runClient(type: JavaExec) {
  group 'client'
  description 'Creates client socket sends a message to the server'

  classpath = sourceSets.main.runtimeClasspath
  standardInput = System.in

  main = 'taskone.Client'
  standardInput = System.in

  if (project.hasProperty("host") && project.hasProperty('port')) {
    args(project.getProperty('host'), project.getProperty('port'));
  }
}
```

B.

```
task runClient(type: JavaExec) {
  group 'client'
  description 'Creates client socket sends a message to the server'

  classpath = sourceSets.main.runtimeClasspath
  standardInput = System.in

  main = 'taskone.Client'
  standardInput = System.in

  args("localhost", 8000);
  if (project.hasProperty("host") && project.hasProperty('port')) {
    args(project.getProperty('host'), project.getProperty('port'));
  }
}
```

C.

```
task runClient(type: JavaExec) {
  group 'client'
  description 'Creates client socket sends a message to the server'

  classpath = sourceSets.main.runtimeClasspath
  standardInput = System.in

  main = 'taskone.Client'
  standardInput = System.in

  if (project.hasProperty("host") && project.hasProperty('port')) {
    args(project.getProperty('host'), project.getProperty('port'));
  } else if (project.hasProperty("host")) {
    args(project.getProperty('host'), 8000);
  } else if (project.hasProperty("port")) {
    args("localhost", project.getProperty('port'))
  } else {
    args("localhost", 8000);
  }
}
```

D.

```
task runClient(type: JavaExec) {
  group 'client'
  description 'Creates client socket sends a message to the server'

  classpath = sourceSets.main.runtimeClasspath
  standardInput = System.in

  main = 'taskone.Client'
  standardInput = System.in

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

***Check out the recording for the solution!***

# 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 |

Dining Philosophers

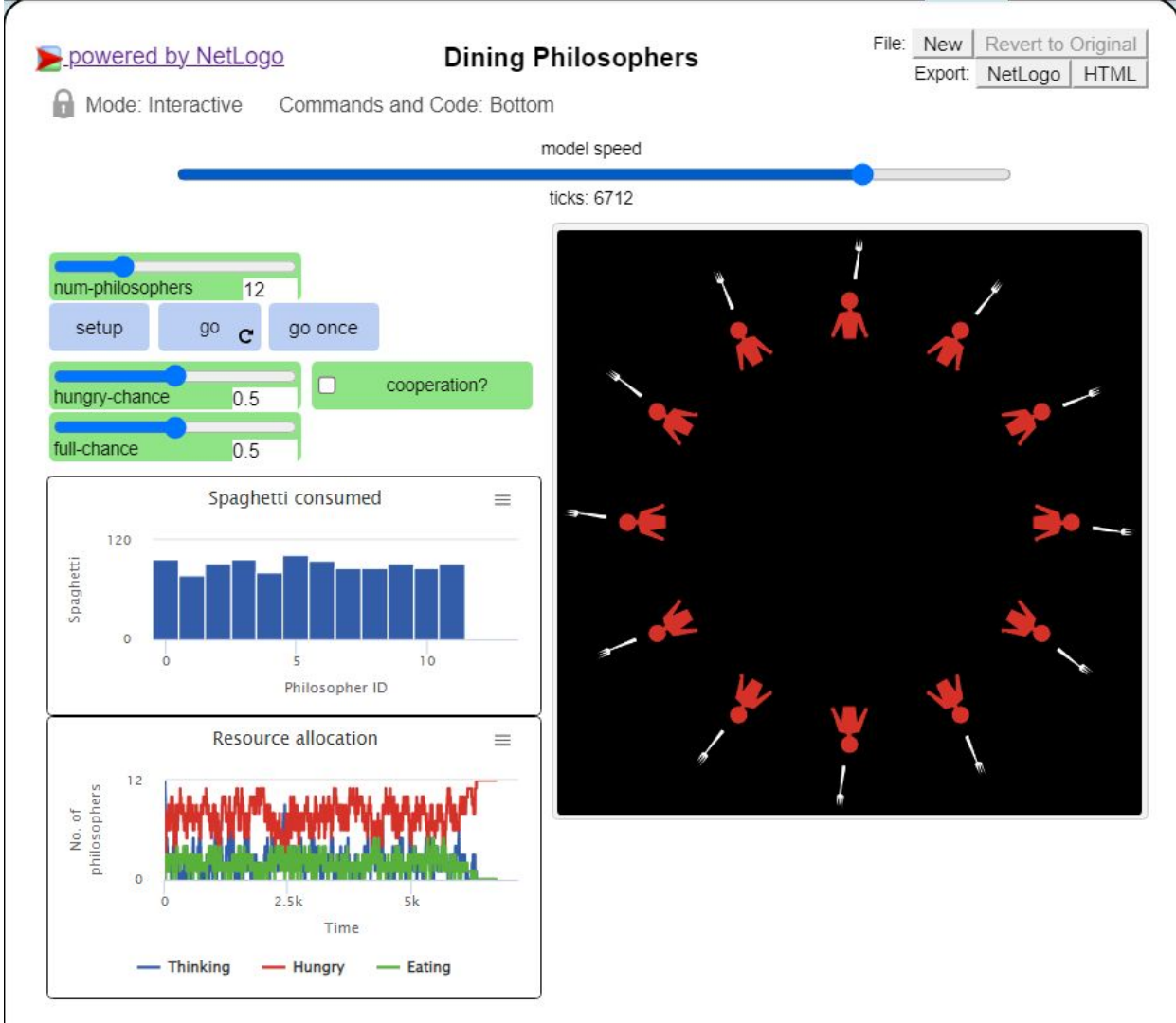
# SER 321
## Threading Pitfalls

What does *Spaghetti Consumed* represent?

What does *Thinking* represent?

What does *Hungry* represent?

***Check out the recording for the solution and discussion!***



Austin Walter's Traffic Comparison

# SER 321

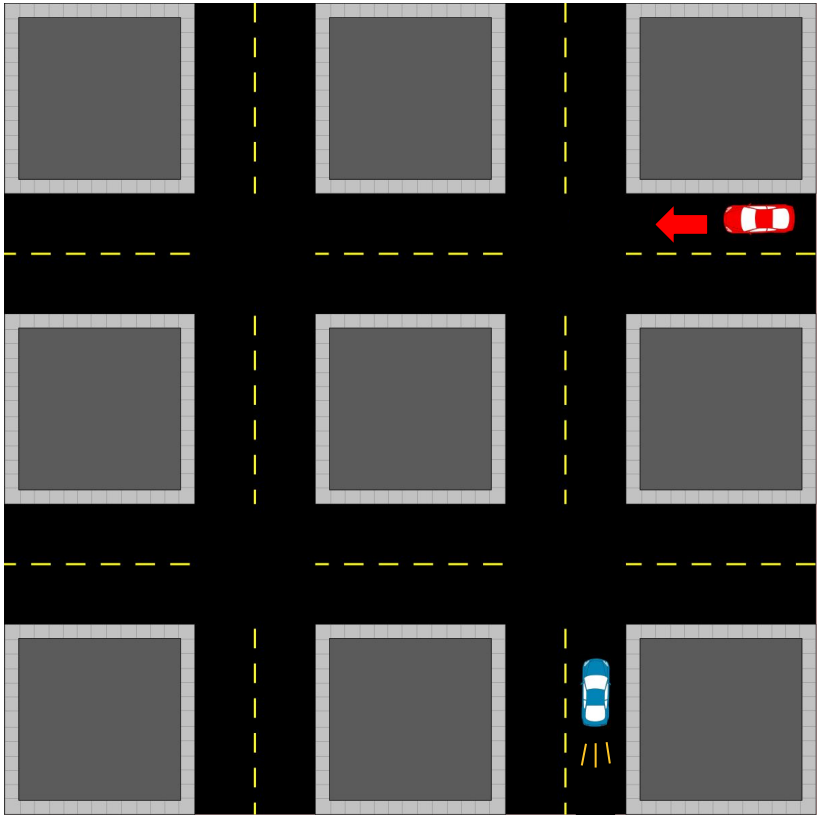
## Threading Pitfalls

***Check out the recording for the discussion!***

Race Condition

Crash

More than one thread accesses a single resource at once



Austin Walter's Traffic Comparison

# SER 321

## Threading Pitfalls

***Check out the recording for the discussion!***

Race Condition

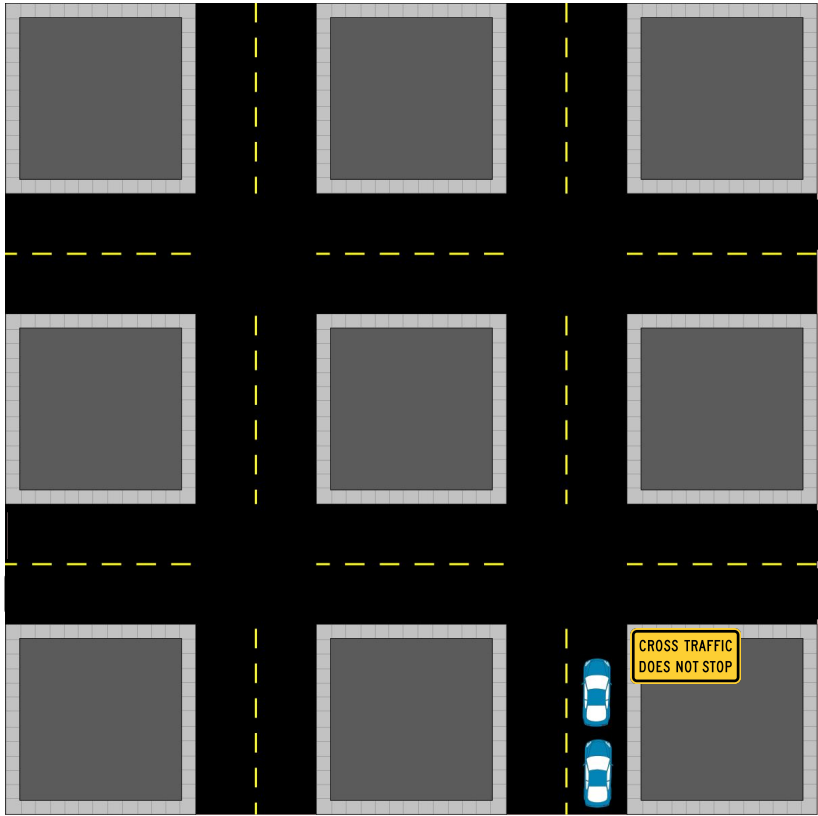
Crash

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

***Check out the recording for the discussion!***

**Race Condition** — **Crash**

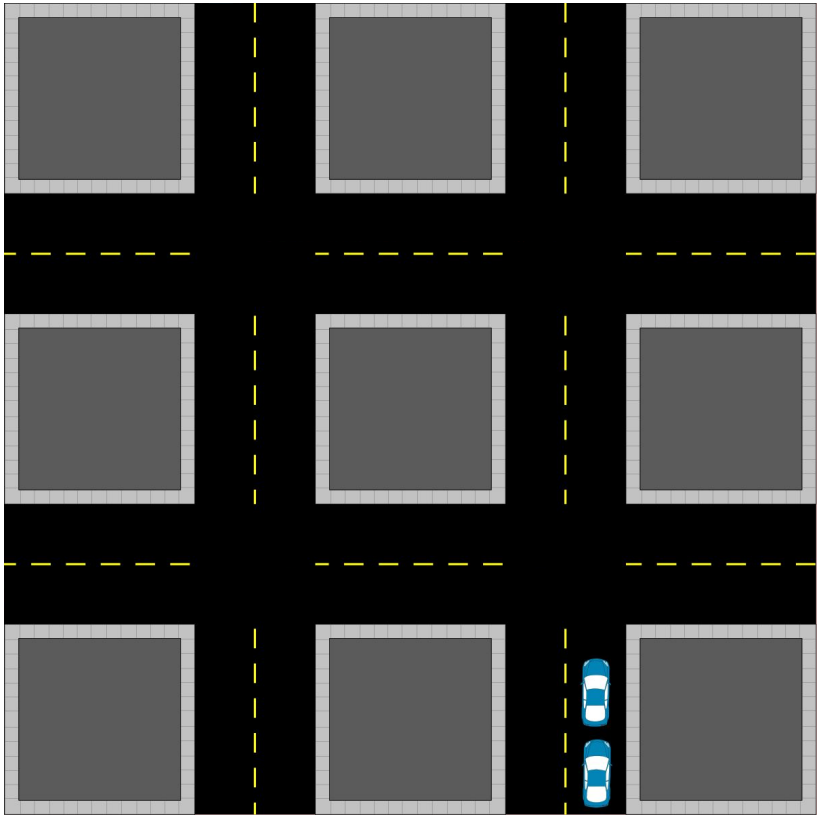
More than one thread accesses a single resource at once

**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



***Check out the recording for the discussion!***

Can we name some concurrency structures?

***Check out the recording for the discussion!***

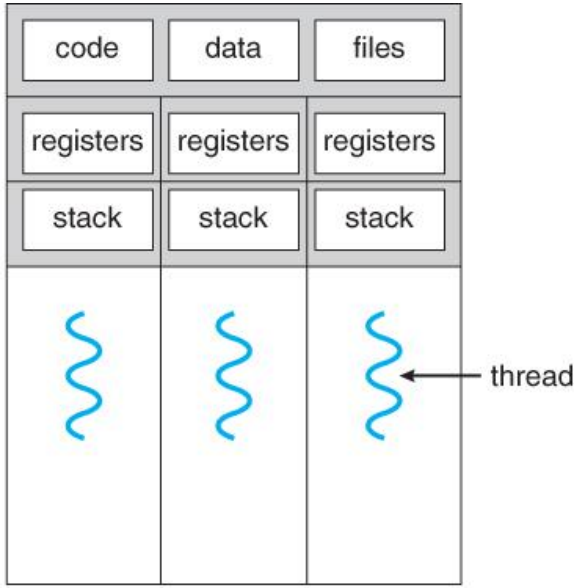
**SER 321**

**Concurrency Structures**

## Atomic Operations & Variables

Recall *registers*…

Ensures updates are immediately visible for the local copy in *each thread*




```
main:
    pushq   %rbp
    movq    %rsp, %rbp
    subq    $48, %rsp
    call    __main
    movl    $5, -4(%rbp)
    movl    $12, -8(%rbp)
    movl    -4(%rbp), %eax
    addl    $7, %eax
    movl    %eax, -12(%rbp)
    movl    -8(%rbp), %edx
    movl    -12(%rbp), %eax
    addl    %edx, %eax
    movl    %eax, -16(%rbp)
    movl    -16(%rbp), %eax
    movl    %eax, %edx
    leaq    .LC0(%rip), %rax
    movq    %rax, %rcx
    call    printf
    movl    $0, %eax
    addq    $48, %rsp
    popq    %rbp
    ret
```

*Check out the recording for the discussion!*

**SER 321**

**Concurrency Structures**

Pros and Cons?

Locks


Acquire the Lock

- Open & Enter
- Close & Lock

Release the Lock → Unlock & Exit

***Check out the recording for the discussion!***

**SER 321**

**Concurrency Structures**

How am I different from a lock?

## Semaphores

More than one stall!

Acquire Lock ⇨ Open & Enter; Close & Lock

Release Lock ⇨ Unlock & Exit

Semaphores support ***more than one*** acquirer

When would that be beneficial?

***Check out the recording for the discussion!***

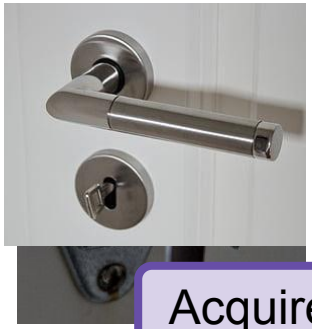
**SER 321**

**Concurrency Structures**

Pros and Cons?

Monitors

You lock the main door instead!



Acquire Lock ⇨ Open & Enter

Close & Lock

Covers the ***entire object***

Release Lock ⇨ Unlock & Exit

***Check out the recording for the discussion!***

**SER 321**

**Concurrency Structures**

# RECAP

Atomic Operations & Variables

***YOU*** control the locks directly

Locks

***YOU*** control the locks directly

Semaphores

***YOU*** control the locks directly

Monitors

Locks managed for you

# SER 321
## Concurrency Structures

Monitors

Both *bow()* and *bowBack()* are synchronized → are we good?

***Check out the recording for the discussion!***

```
PS C:\ASU\SER321\examples_repo\ser321examples\Threads\Deadlock> gradle run
Starting a Gradle Daemon (subsequent builds will be faster)

> Task :run
Alphonse: Gaston  has bowed to me!
Gaston: waiting to bow back
Gaston: Alphonse  has bowed to me!
Alphonse: waiting to bow back
<==========----> 75% EXECUTING [17s]
> :run
```

Deadlock!

```
public class Deadlock {
    static class Friend {  6 usages
        private final String name;  5 usages
        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) {  2 usages
            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( bower: this);
        }
        public synchronized void bowBack(Friend bower) {  1 usage
            System.out.format("%s: waiting", 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( name: "Alphonse");
        final Friend gaston =
                new Friend( name: "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
## Concurrency Structures

Monitors manage locks for us by ***locking the entire object***

```
public class Deadlock {
```

This program demonstrate how a deadlock can be created with synchronized methods:

- https://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html
- https://docs.oracle.com/javase/tutorial/essential/concurrency/locksync.html

The key to why it locks can be found in this bullet point from the Tutorial:

- "When a thread invokes a synchronized method, it automatically acquires the intrinsic lock for that method's object and releases it when the method returns. The lock release occurs even if the return was caused by an uncaught exception."

Since both the `bow()` and `bowback()` method are synchronized methods, they cannot both be called on the same object at the same time, whichever is called first must complete prior to the other executing.

The key to solving this is to use a synchronized statement rather than a synchronized method. With this approach a separate lock object can be shared and keep a deadlock from occurring by not allowing the second bower to start before the first has finished.

A more sophisticated locking scheme can be accomplished with explicit Lock objects and is described here:

- https://docs.oracle.com/javase/tutorial/essential/concurrency/newlocks.html

***Check out the recording for the discussion!***

```
> Task :run
Alphonse: Gaston  has bowed to me!
Gaston: waiting to bow back
Gaston: Alphonse  has bowed to me!
Alphonse: waiting to bow back
<==========----> 75% EXECUTING [17s]
> :run
```

```
}
```

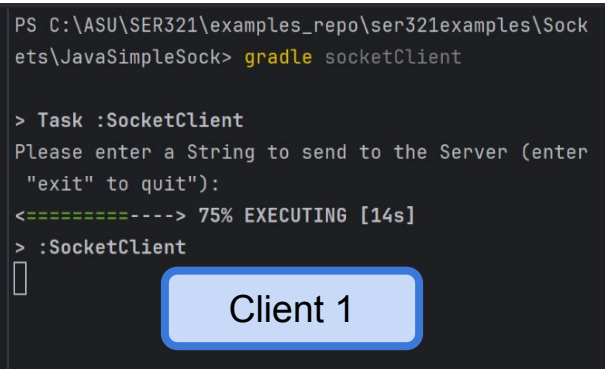
JavaSimpleSock

***Check out the recording for the discussion!***

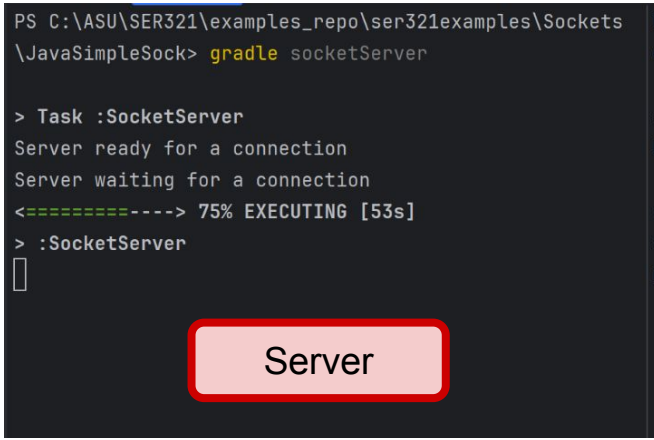
# SER 321

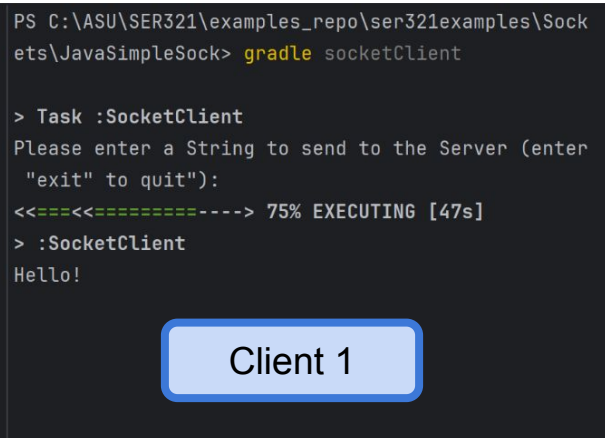
## Single Threaded Server

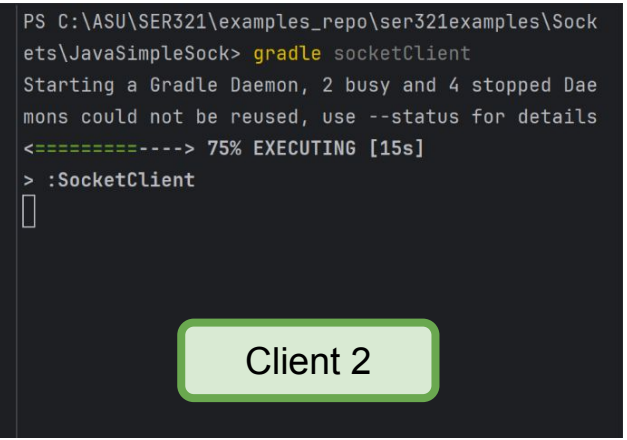
What will happen if there are two clients?





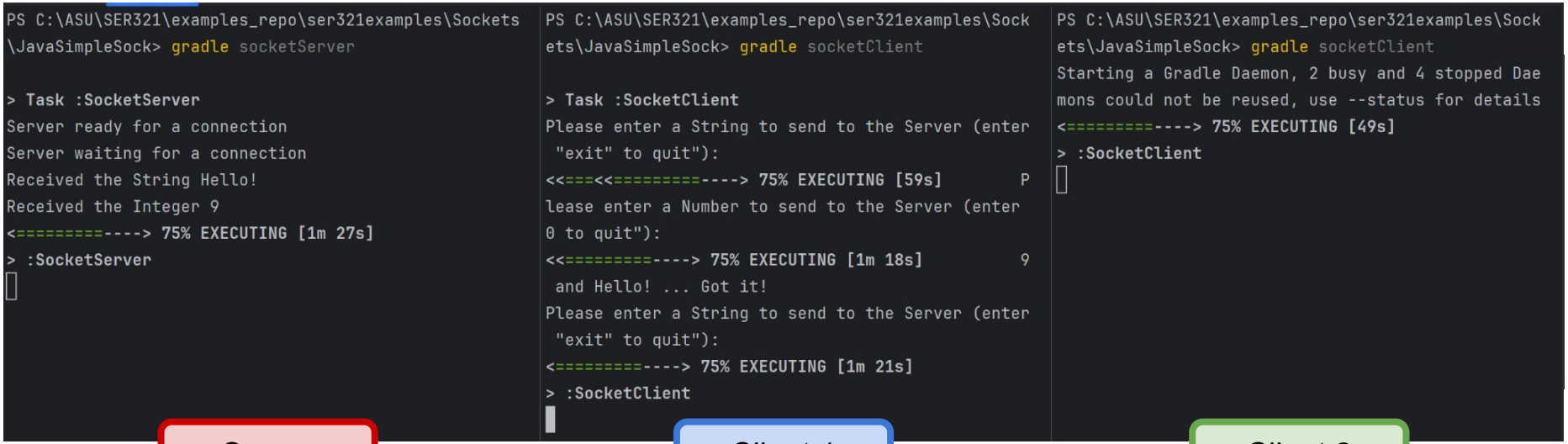



JavaSimpleSock

***Check out the recording for the discussion!***

# SER 321

## Single Threaded Server



JavaSimpleSock

***Check out the recording for the discussion!***

# 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

Why?



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

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

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

```
ets\JavaSimpleSock>
```

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

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

Server

Client 1

Client 2

***Check out the recording for the discussion!***

# SER 321

## Scratch Space

## Upcoming Events

# SI Sessions:

- Tuesday, February 11th at 11:00 am MST
- Thursday, February 13th at 7:00 pm MST
- Sunday, February 16th at 7:00 pm MST

# Review Sessions:

- Tuesday, February 25th at 11:00 am MST - **Q&A Session**
- Thursday, February 27th at 7:00 pm MST - **Exam Review Session (2hrs)**

# Questions?

# Survey:

https://asuasn.info/ASNSurvey

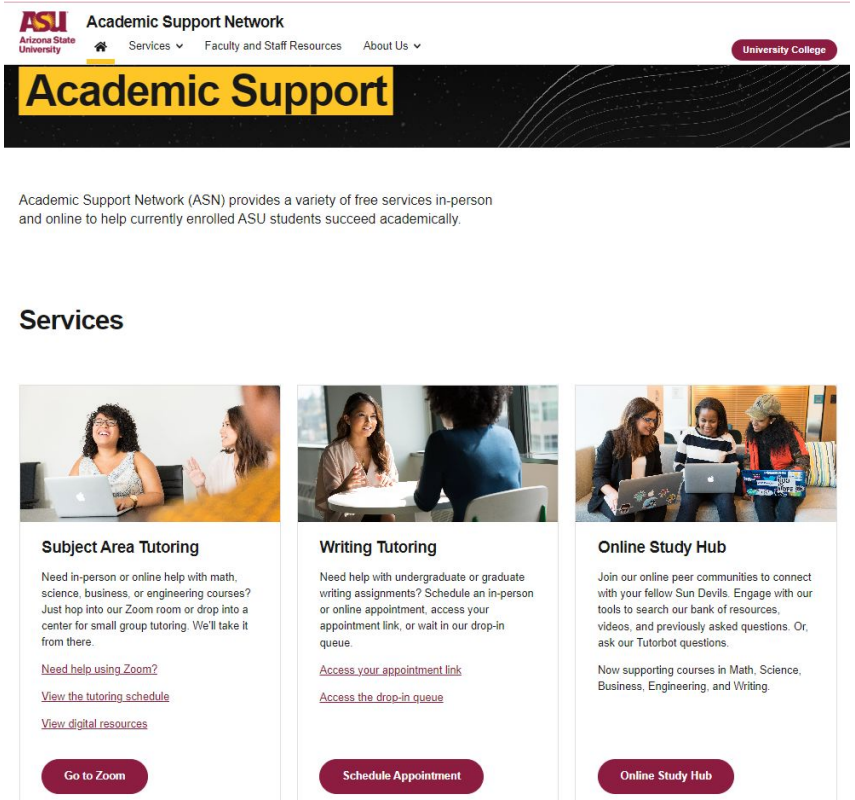


29

# More Questions?

## Check out our other resources!

**tutoring.asu.edu**



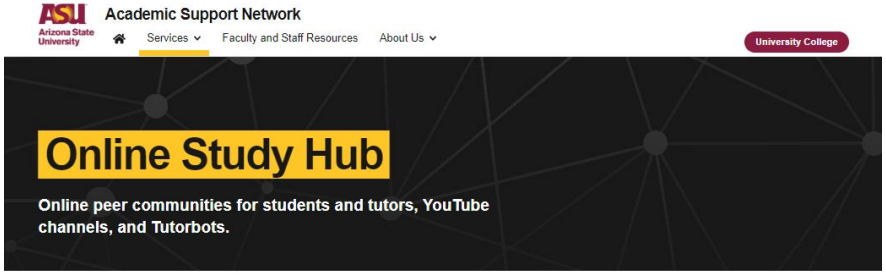


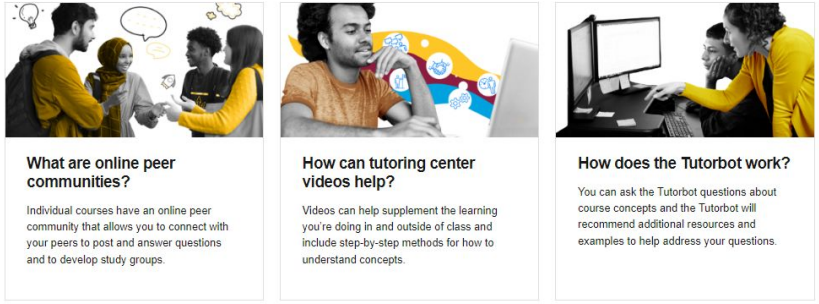
1. **Click on ‘Go to Zoom’ to log onto our Online Tutoring Center.**
2. **Click on ‘View the tutoring schedule’ to see when tutors are available for specific courses.**

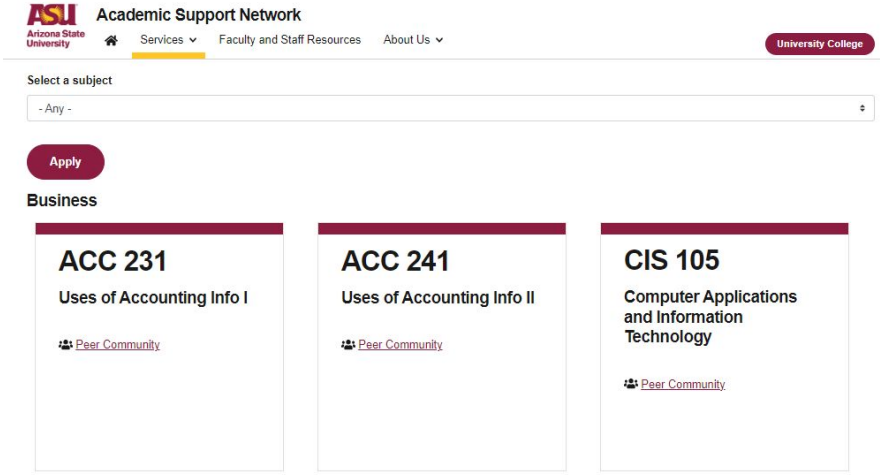
# More Questions?

## Check out our other resources!

**tutoring.asu.edu/online-study-hub**



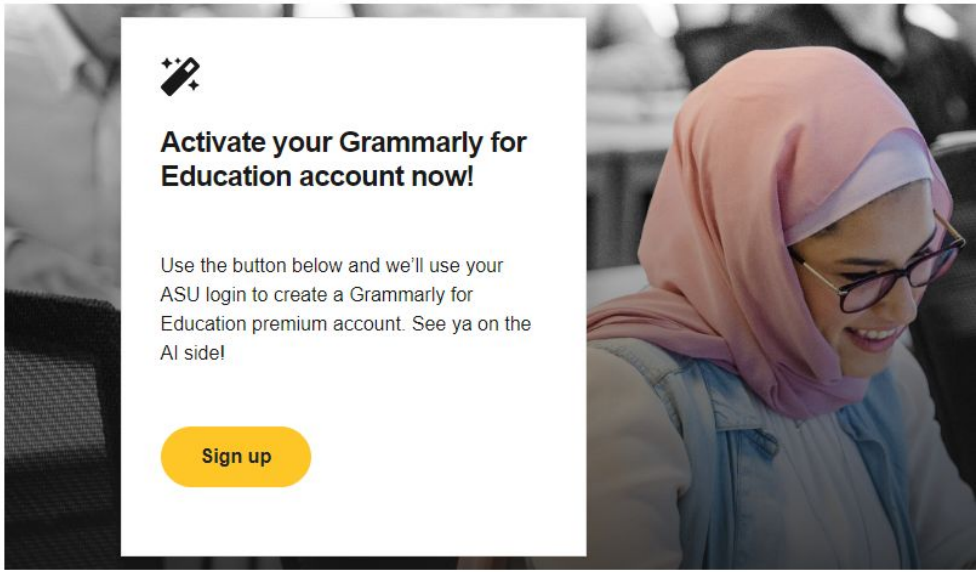




Don't forget to check out the Online Study Hub for additional resources!

# 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