

SER 321 B Session

SI Session

Thursday, April 10th 2025

7:00 pm - 8:00 pm MST

Agenda

- 
- Protocol Buffers
 - Threading
 - Intro, Usage, and Pitfalls
 - Threading our System
 - Concurrency Structures

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

Quick PSA for Protobufs!

Make sure you watch all the lecture videos!

You must generate the Protobufs for use!

Sanity Check - what are Protocol Buffers?

4-2 Starter Code

SER 321 Protobufs

Option 1:
IDE

The screenshot shows a Java code editor with the following details:

- Project:** Assignment4.2given
- File:** SockBaseServer.java
- Code Content:** The code implements a server using sockets. It defines a static string `logFilename` and two static strings for menu and game options. It initializes a `ServerSocket`, `InputStream`, and `OutputStream`. It maintains a `clientSocket` and a `Game` object. It tracks the current state and grading status. It has a constructor that takes a `Socket`, a `Game`, and an `id`. It also has a `try` block that creates an `InputStream` from the `clientSocket`.
- IDE UI:** The interface includes a sidebar with project files like `.gradle`, `.idea`, and `gradle`. The code editor has tabs for `README.md`, `PROTOCOL.md`, `SockBaseServer.java`, and `SockBaseClient.java`. A status bar at the bottom shows the file path and system information.

4-2 Starter Code

SER 321
Protobufs

Option 1:
IDE

The screenshot shows an IDE interface with the following details:

- Project:** Assignment4.2given
- File:** SockBaseServer.java
- Code Editor:** Displays the Java code for the SockBaseServer class.
- Right Sidebar:**
 - Tasks:** A list of Gradle tasks, with "generateProto" highlighted by a yellow arrow.
 - Other:** A list of other Gradle tasks.
- Bottom Status Bar:** Shows the file path (Assignment4.2given > src > main > java > server > SockBaseServer), the current time (13:7), and other system information (LF, UTF-8, 4 spaces).

```
package server;
import ...;

class SockBaseServer {
    static String logFilename = "logs.txt"; 3 usages
    // Please use these as given so it works with our test cases
    static String menuOptions = "\nWhat would you like to do? \n 1 - to see";
    static String gameOptions = "\nChoose an action: \n (1-9) - Enter an int";

    ServerSocket serv = null; no usages
    InputStream in = null; 3 usages
    OutputStream out = null; 4 usages
    Socket clientSocket = null; 4 usages
    private final int id; // client id 2 usages

    Game game; // current game 3 usages

    private boolean inGame = false; // a game was started (you can decide if
    private String name; // player name 4 usages

    private int currentState = 1; // I used something like this to keep track

    private static boolean grading = true; // if the grading board should be

    public SockBaseServer(Socket sock, Game game, int id) { 1 usage
        this.clientSocket = sock;
        this.game = game;
        this.id = id;
    }
}
```

4-2 Starter Code

SER 321 Protobufs

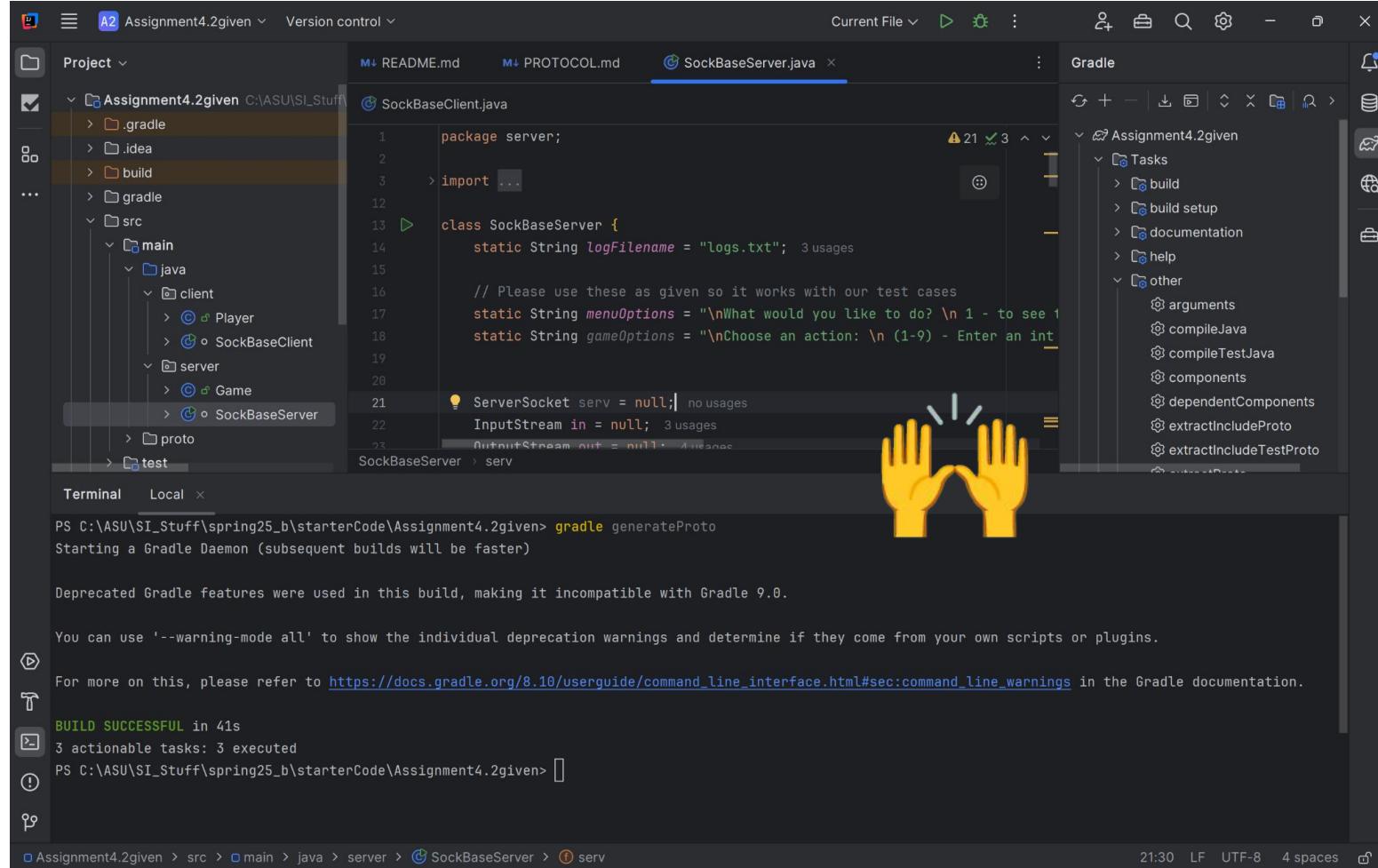
Option 2:
Command
Line

The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "Assignment4.2given". It contains a "src" directory with "main" and "proto" sub-directories. "main" further contains "client" and "server" sub-directories, each with "Player" and "Game" classes, and a "SockBaseClient" and "SockBaseServer" class respectively.
- Code Editor:** The "SockBaseServer.java" file is open. The code includes imports for "Player" and "Game". It defines static strings for logFilename ("logs.txt"), menuOptions ("What would you like to do? 1 - to see 1"), and gameOptions ("Choose an action: (1-9) - Enter an int"). It also declares variables for ServerSocket, InputStream, OutputStream, and Socket, and a private final int id.
- Terminal:** The terminal window shows the command "PS C:\ASU\SI_Stuff\spring25_b\starterCode\Assignment4.2given: gradle generateProto" entered by the user. This command is highlighted with a yellow box.
- Gradle Task List:** The right-hand sidebar lists various Gradle tasks under the "Tasks" section, including build, build setup, documentation, help, and other options like arguments, compileJava, and generateProto.

4-2 Starter Code

SER 321
Protobufs



Project A2 Assignment4.2given Version control Current File M: README.md M: PROTOCOL.md SockBaseServer.java Gradle

Assignment4.2given C:\ASU\SI_Stuff

src main java client Player SockBaseClient server Game SockBaseServer proto test

package server;

> import ...

class SockBaseServer {

static String logFilename = "logs.txt"; 3 usages

// Please use these as given so it works with our test cases

static String menuOptions = "\nWhat would you like to do? \n 1 - to see 1

static String gameOptions = "\nChoose an action: \n (1-9) - Enter an int

ServerSocket serv = null; | no usages

InputStream in = null; 3 usages

OutputStream out = null; 4 usages

SockBaseServer > serv

Tasks build build setup documentation help other arguments compileJava compileTestJava components dependentComponents extractIncludeProto extractIncludeTestProto

Terminal Local

```
PS C:\ASU\SI_Stuff\spring25_b\starterCode\Assignment4.2given> gradle generateProto
Starting a Gradle Daemon (subsequent builds will be faster)

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.10/userguide/command\_line\_interface.html#sec:command\_line\_warnings in the Gradle documentation.

BUILD SUCCESSFUL in 41s
3 actionable tasks: 3 executed
PS C:\ASU\SI_Stuff\spring25_b\starterCode\Assignment4.2given> 
```

Assignment4.2given > src > main > java > server > SockBaseServer > serv

21:30 LF UTF-8 4 spaces

SER 321**Protobufs**

Options for Message Creation

```
Response.newBuilder()  
    .setResponseType(Response.ResponseType.GREETING)  
    .setMessage("Hello " + name + " and welcome to a simple game of Sudoku.")  
    .setMenuoptions(menuOptions)  
    .setNext(currentState)  
    .build();
```

SockBaseServer

Create the message in
a single statement

Create the message in
increments

```
Request.Builder req = Request.newBuilder();  
  
switch (response.getResponseType()) {  
    case GREETING:  
        System.out.println(response.getMessage());  
        req = chooseMenu(req, response);  
        break;
```

SockBaseClient - main

SER 321**Protobufs**

Options for Message Creation

```

static Request.Builder chooseMenu(Request.Builder req, Response response) throws IOException {
    while (true) {
        System.out.println(response.getMenuoptions());
        System.out.print("Enter a number 1-3: ");
        BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
        String menu_select = stdin.readLine();
        System.out.println(menu_select);
        switch (menu_select) {
            // needs to include the other requests
            case "2":
                // this is not a complete START request!! Just as example
                req.setOperationType(Request.OperationType.START);
                return req;
            default:
                System.out.println("\nNot a valid choice, please try again");
                break;
        }
    }
}

```

Need one
more step...

```

Request.Builder req = Request.newBuilder();

switch (response.getResponse_type()) {
    case GREETING:
        System.out.println(response.getMessage());
        req = chooseMenu(req, response);
        break;
}

```

SockBaseClient - main

req.build().writeDelimitedTo(out);

SockBaseClient - chooseMenu

SER 321

Protobufs

Parsing Messages

GETTERS!

```
System.out.println("Got a response: " + response.toString());
```

```
switch (response.getResponseType()) {  
    case GREETING:  
        System.out.println(response.getMessage());  
        req = chooseMenu(req, response);  
        break;  
}
```

Fetch a single value

Fetch a *repeated* value

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

SER 321

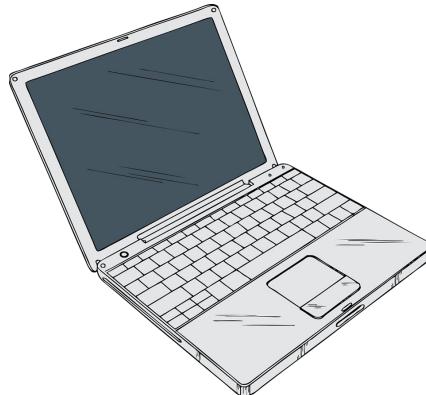
System Layout

You have two systems...

How can we test our server with multiple clients?



?

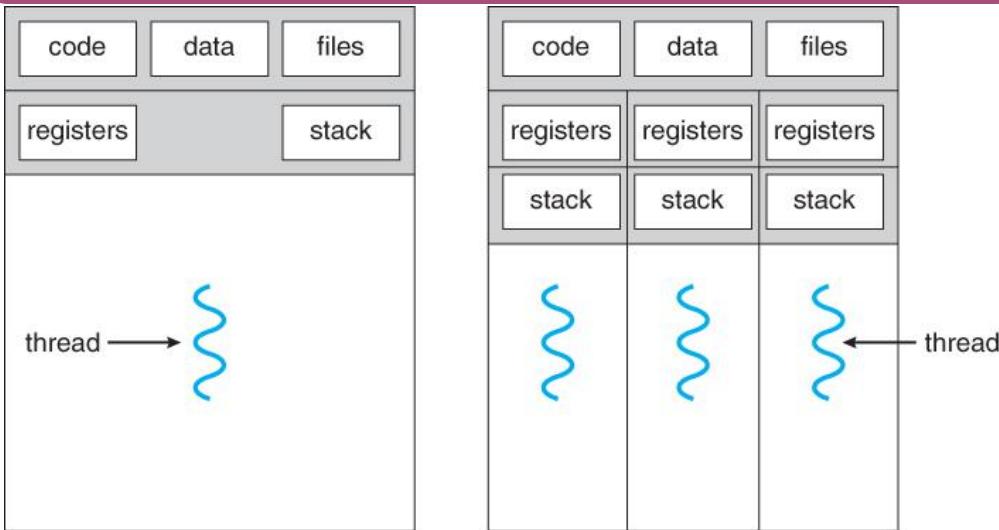


SER 321

Threads

What does that imply?

Remember that they exist
within the parent process



Process 1

data

Thread
Thread
Thread

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

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

SER 321

Threading Pitfalls

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

```
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();  
    }  
}
```

Client

But what happened?

```
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();  
    }  
}
```

Server

```
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

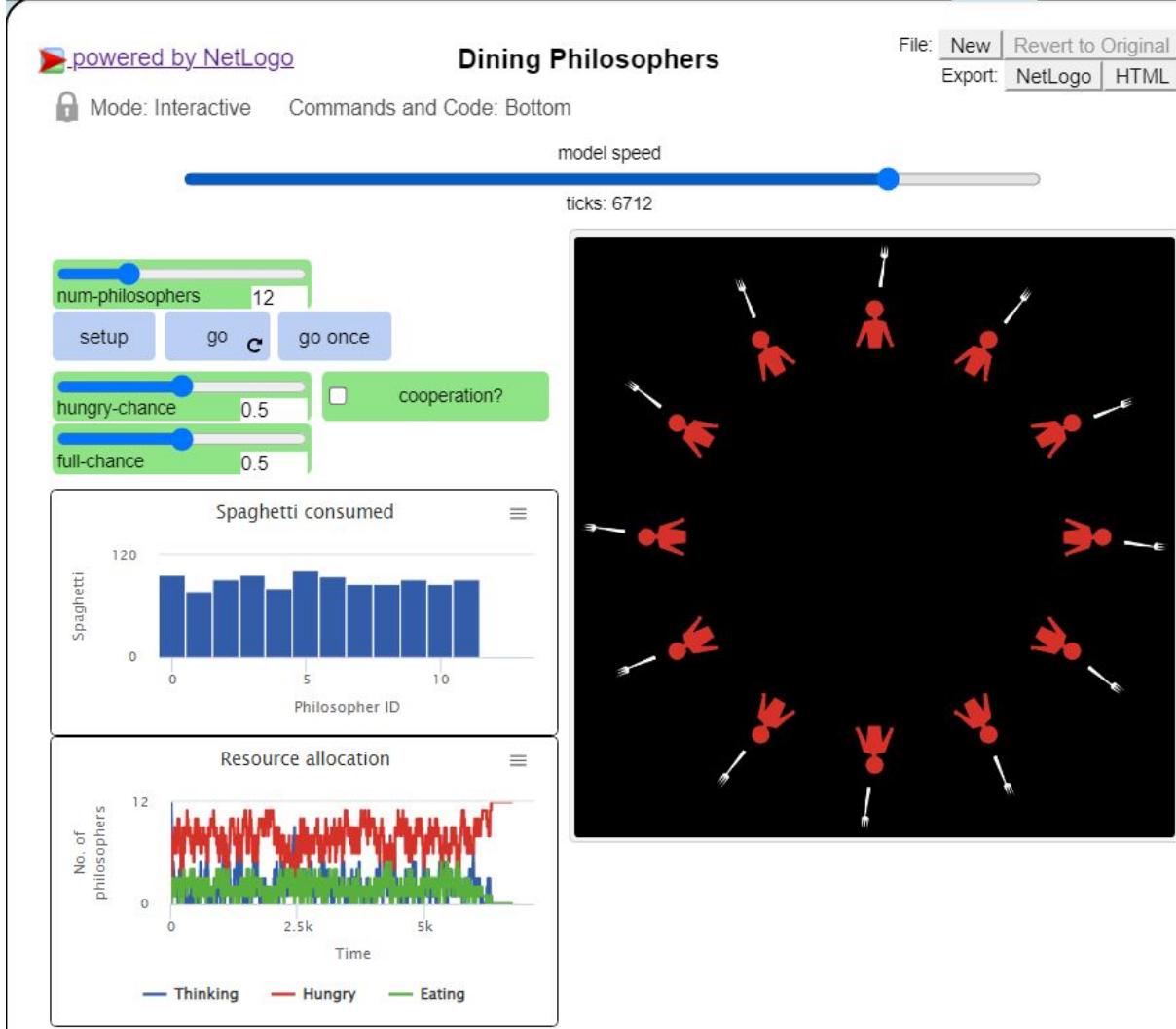
SER 321

Threading Pitfalls

What does *Spaghetti Consumed* represent?

What does *Thinking* represent?

What does *Hungry* represent?



SER 321

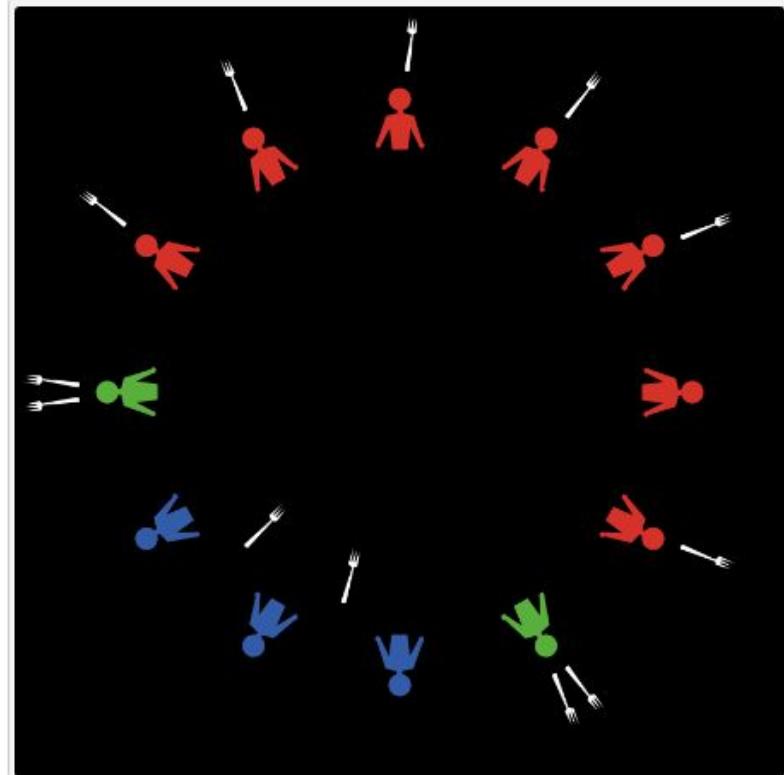
Dining Philosophers

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?



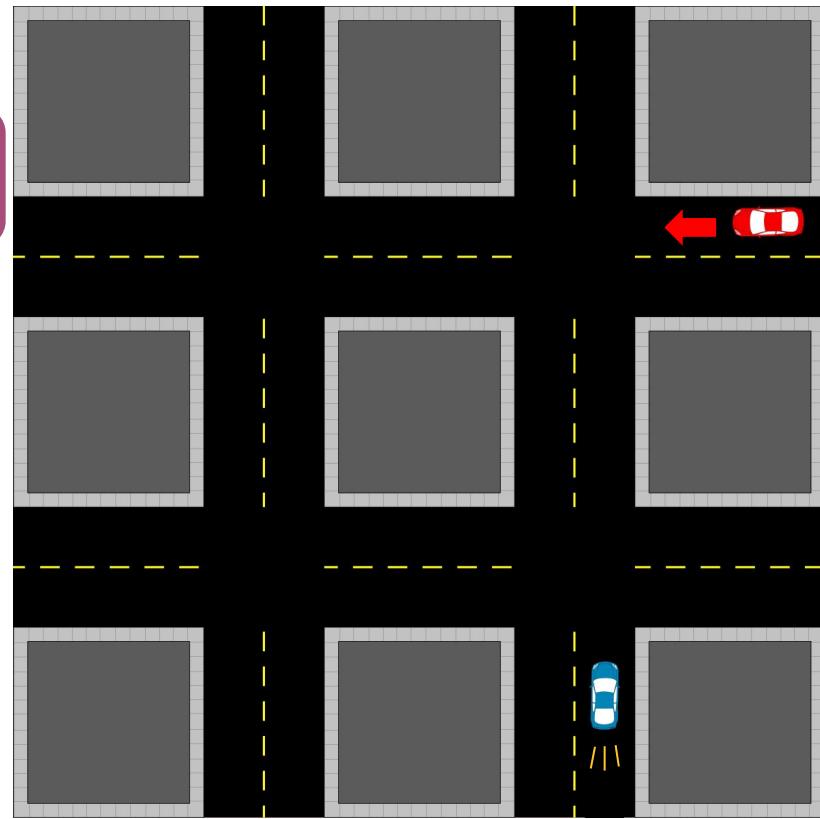
SER 321

Threading Pitfalls

Race Condition

Crash

More than one thread accesses a single resource at once



SER 321

Threading Pitfalls

Race Condition

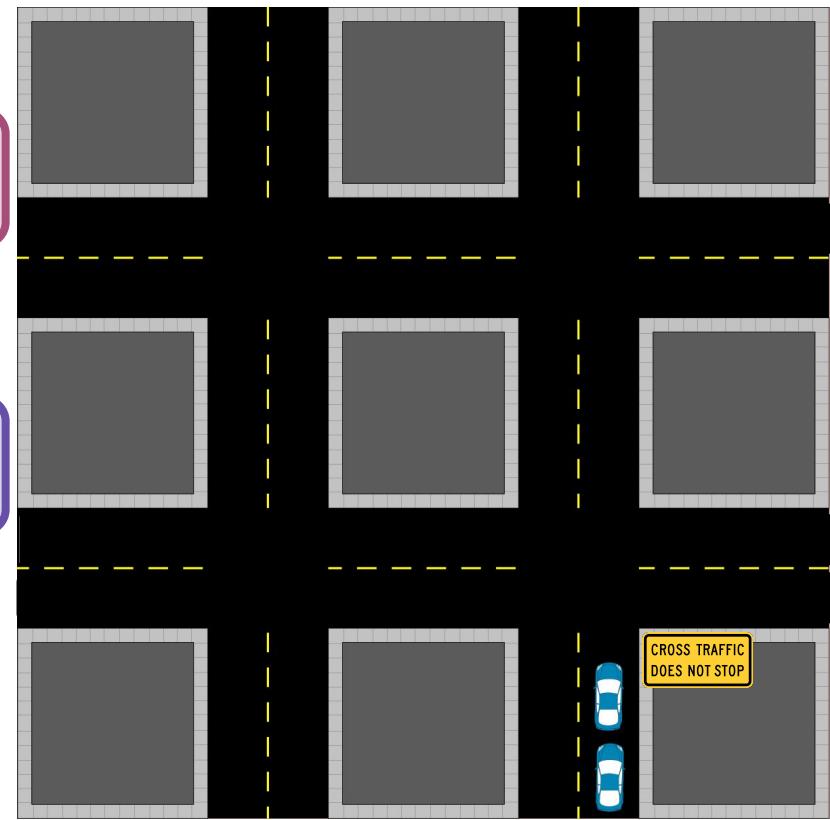
Crash

More than one thread accesses a single resource at once

Starvation

Cross Traffic

A thread never gains access to the resource it needs



SER 321

Threading Pitfalls

Race Condition

Crash

More than one thread accesses a single resource at once

Starvation

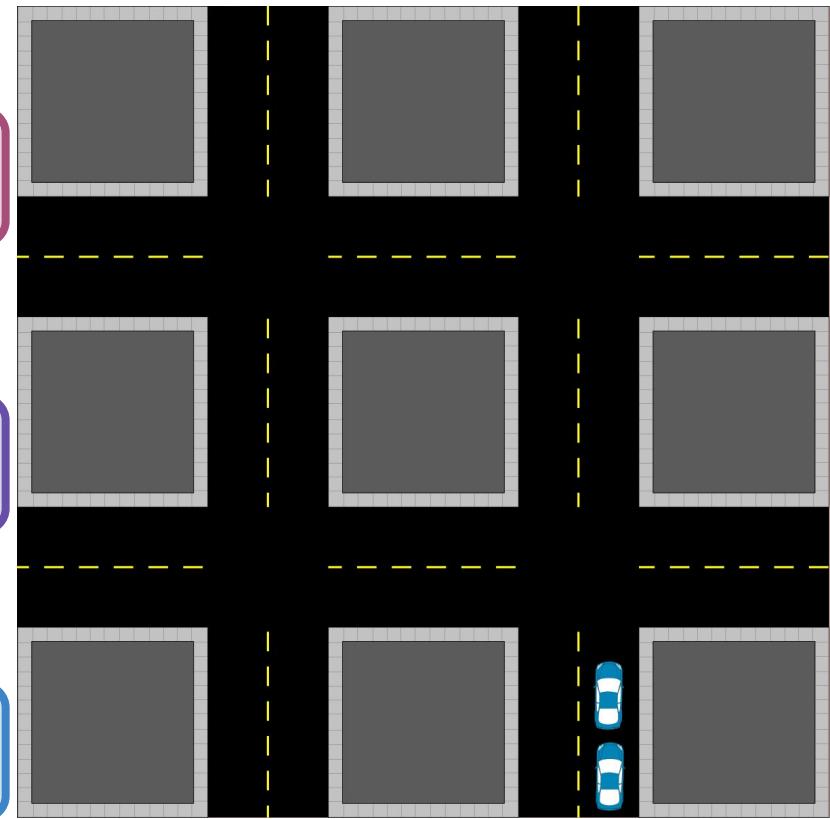
Cross Traffic

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

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
<=====--> 75% EXECUTING [20s]
> :SocketServer
```

Server

```
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

What will happen if there are two clients?

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

Client 2

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets
\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\Sockets
\JavaSimpleSock> gradle socketClient
Starting a Gradle Daemon, 2 busy and 4 stopped Daemons
could not be reused, use --status for details
<=====--> 75% EXECUTING [15s]
> :SocketClient
```

Client 2

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\Sockets\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\Sockets\JavaSimpleSock> gradle socketClient
Starting a Gradle Daemon, 2 busy and 4 stopped Daemons
could not be reused, use --status for details
<=====--> 75% EXECUTING [49s]
> :SocketClient
[]
```

Client 2

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\Sockets\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\Sockets\JavaSimpleSock> gradle socketClient  
Starting a Gradle Daemon, 2 busy and 4 stopped Daemons could not be reused, use --status for details  
<=====--> 75% EXECUTING [1m 18s]  
> :SocketClient  
[]
```

Client 2

What do we think will happen?

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  
Received the String exit  
Received the Integer 0  
Server waiting for a connection  
<=====--> 75% EXECUTING [2m 15s]  
> :SocketServer  
[]
```

```
and Hello! ... Got it!  
Please enter a String to send to the Server (enter  
"exit" to quit):  
<=====--> 75% EXECUTING [2m 3s]      e  
xitingketClient  
  
Deprecated Gradle features were used in this build  
, making it incompatible with Gradle 8.0.  
  
You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.  
  
See https://docs.gradle.org/7.4.2/userguide/command\_line\_interface.html#sec:command\_line\_warnings  
  
BUILD SUCCESSFUL in 2m 5s  
2 actionable tasks: 1 executed, 1 up-to-date  
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\JavaSimpleSock>
```

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\JavaSimpleSock> gradle socketClient  
Starting a Gradle Daemon, 2 busy and 4 stopped Daemons 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 [1m 37s]  
> :SocketClient  
[]
```

Server

Client 1

Client 2

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

```
and Hello! ... Got it!
Please enter a String to send to the Server (enter
"exit" to quit):
<=====--> 75% EXECUTING [2m 3s]      e
xitingketClient

Deprecated Gradle features were used in this build
, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.4.2/userguide/command\_line\_interface.html#sec:command\_line\_warnings

BUILD SUCCESSFUL in 2m 5s
2 actionable tasks: 1 executed, 1 up-to-date
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\JavaSimpleSock> []
```

Server

Client 1

Client 2

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\JavaSimpleSock> gradle socketClient
Starting a Gradle Daemon, 2 busy and 4 stopped Daemons 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[]
```

SER 321

Single Threaded Server

Why?

Client 1

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  
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.

```
PS C:\ASU\SER321\examples_repo\ser321examples\Sockets\JavaSimpleSock> gradle socketClient  
Starting a Gradle Daemon, 2 busy and 4 stopped Daemons 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

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

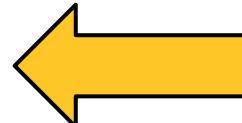
6. Wait for Connection

7. Handle Client Connection

8. Close Client Connection

9. Continue Listening

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



7. Send Client Socket to thread

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



1

2 & 3-5

9

6

7

8

```
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

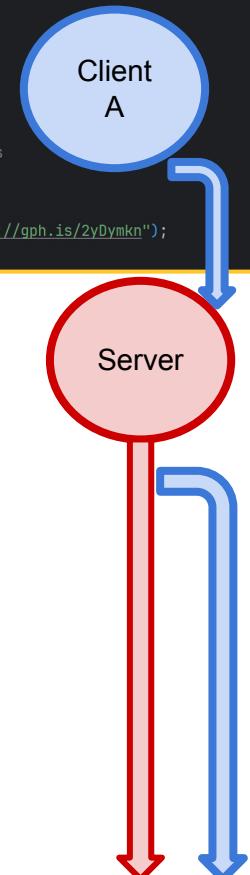
SER 321 Threads

```
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("^[\\d]+$")) {
                validInput = false;
                out.writeObject("Not a number: https://gph.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();
    }
}
```



```
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

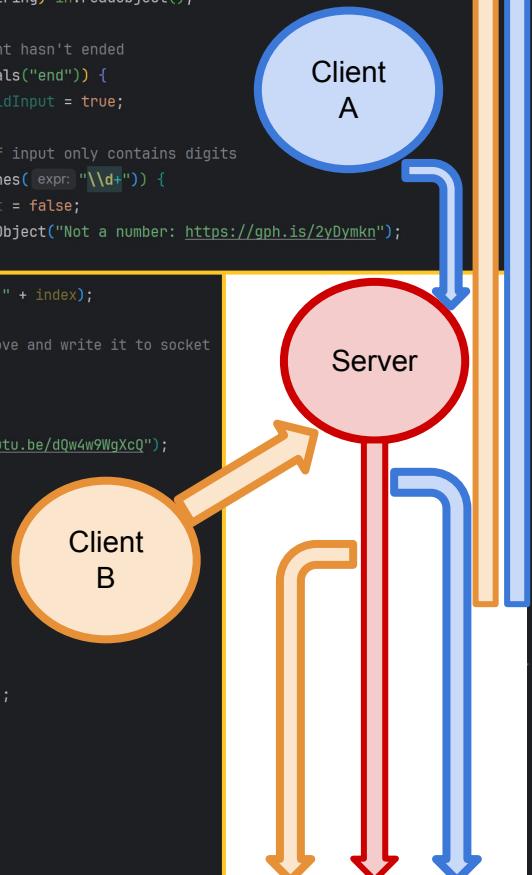
SER 321 Threads

```
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("^[\\d]+$")) {
                validInput = false;
                out.writeObject("Not a number: https://gph.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();
    }
}
```



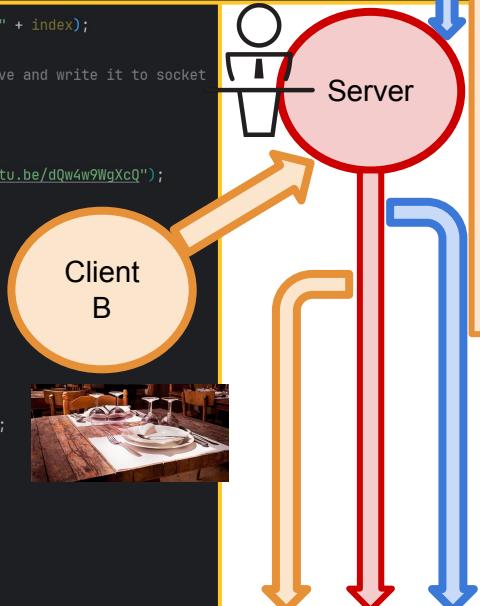
```
public static void main(String args[]) throws IOException {
    Socket sock = null;
    int id = 0;
    try {
        if (args.length != 1) {
            System.out.println(
                "Usage: gradle ThreadedSockServer --args=<port num>");
            System.exit( code: 0);
        }
      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

```
// 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();
}
```

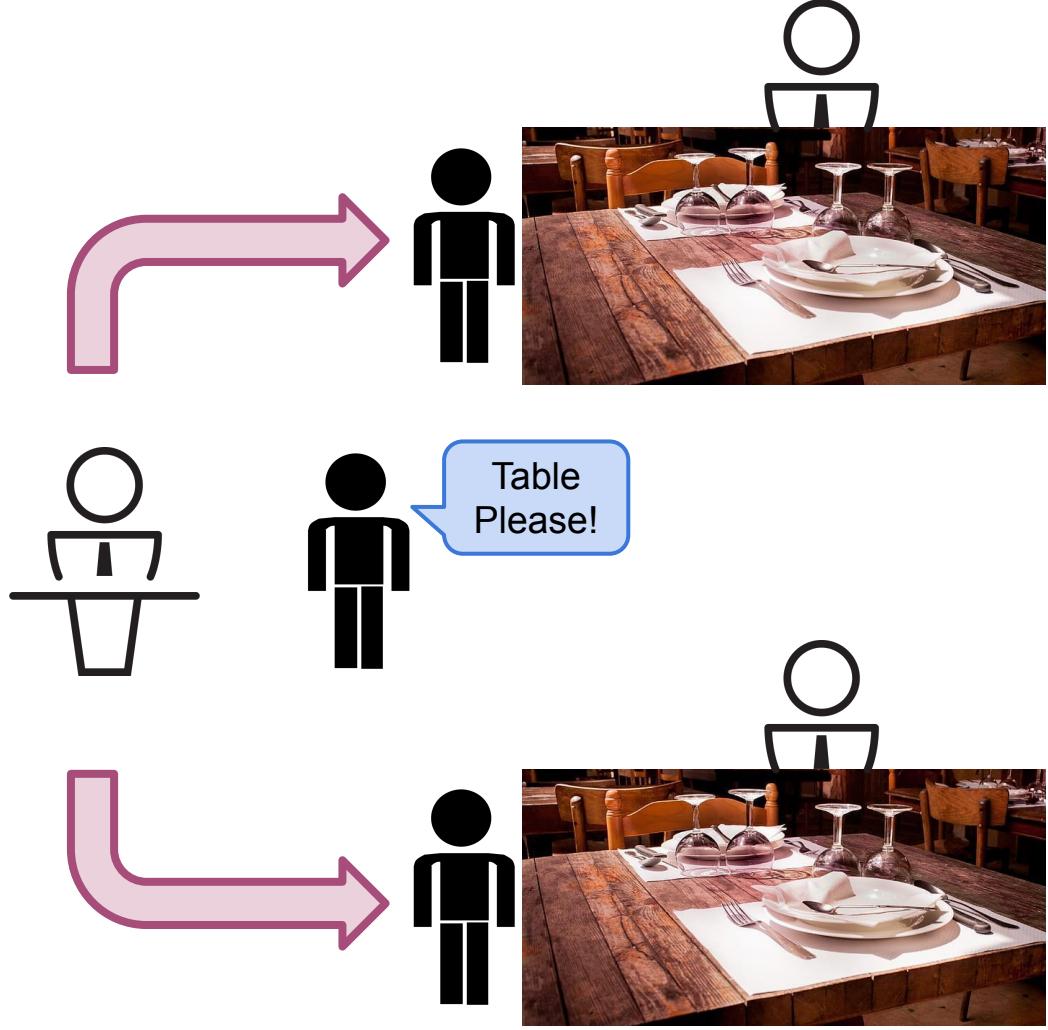
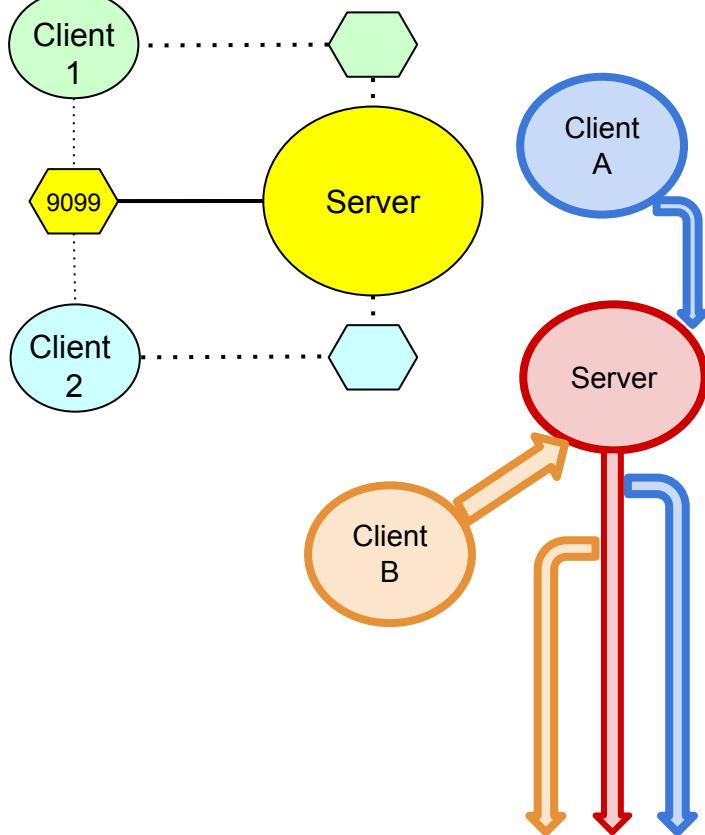


```
public static void main(String args[]) throws IOException {
    Socket sock = null;
    int id = 0;
    try {
        if (args.length != 1) {
            System.out.println(
                "Usage: gradle ThreadedSockServer --args=<port num>");
            System.exit( code: 0);
        }
      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

Port Examination



SER 321

Concurrency Structures

Can we name some concurrency structures?

Atomic Operations &
Variables

Locks

Semaphores

Monitors

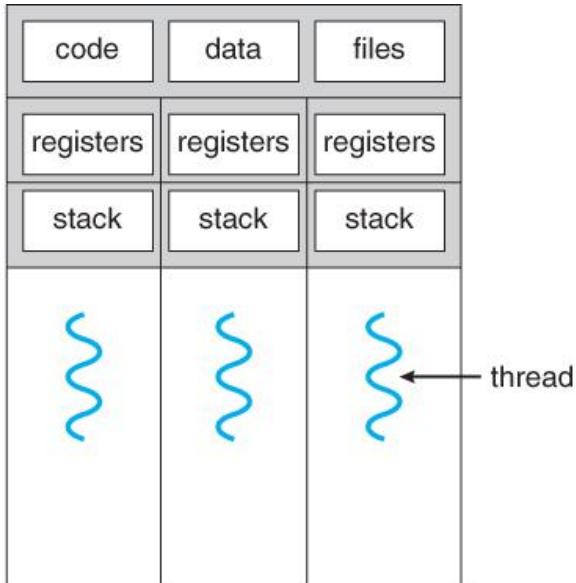
SER 321

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
```

SER 321

Concurrency Structures

Pros and Cons?

Locks

Acquire the Lock

Open & Enter

Close & Lock

Release the Lock

Unlock & Exit



SER 321

Concurrency Structures



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?

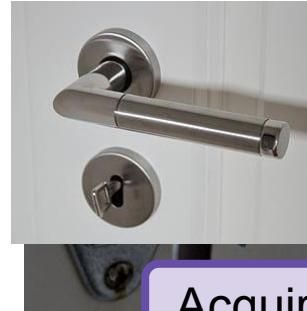
How am I
different from a
lock?

Pros and Cons?

Monitors



You lock
the main
door
instead!



Acquire Lock



Open & Enter

Close & Lock

Covers the
entire object

Release Lock



Unlock & Exit

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?

```
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***

```
> 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
```

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/locksSync.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>

SER 321

Scratch Space

Upcoming Events

SI Sessions:

- Sunday, April 13th at 7:00 pm MST
- Tuesday, April 15th at 10:00 am MST
- Thursday, April 17th at 7:00 pm 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>



More Questions?

Check out our other resources!

tutoring.asu.edu

The screenshot shows the ASU Academic Support Network homepage. At the top, there's a yellow header with the ASU logo and the text "Academic Support Network". Below the header, there's a navigation bar with links for "Services", "Faculty and Staff Resources", and "About Us". A red button labeled "University College" is visible. The main section features a large yellow banner with the text "Academic Support". Below the banner, there's a brief description of the service: "Academic Support Network (ASN) provides a variety of free services in-person and online to help currently enrolled ASU students succeed academically." There are also sections for "Services" and "Online Study Hub" with images and descriptions.

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](#)



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](#)

[View digital resources](#)



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

[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

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.

ASU Academic Support Network

Arizona State University Services Faculty and Staff Resources About Us

Select a subject

- Any -

Apply

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!

Expanded Writing Support Available

Including Grammarly for Education, at no cost!



 Activate your Grammarly for Education account now!

Use the button below and we'll use your ASU login to create a Grammarly for Education premium account. See ya on the AI side!

[Sign up](#)

*Available slots for this pilot are limited



tutoring.asu.edu/expanded-writing-support

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](#)