SER 321 A Session

SI Session

Wednesday September 13th, 2023

6:00 - 7:00 pm MST

Agenda

- Gradle Review
- JSON Review
- Beefing up Client and Server
- Protobufs
- Protocol Organization Strategies
- Starter Code

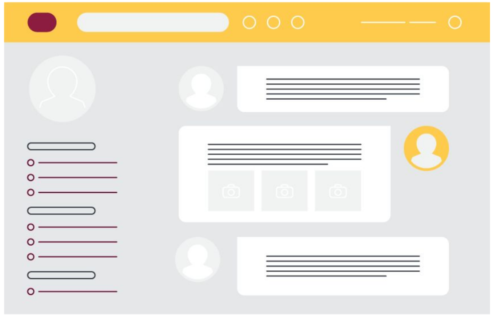
SI Session Expectations

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



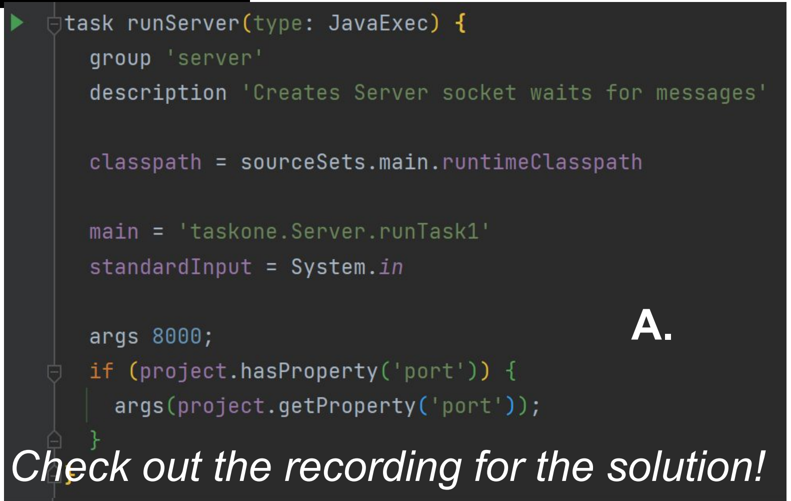
Zoom Chat

- Use the chat feature to interact with the presenter and respond to presenter's questions.
- Annotations are encouraged

SER 321

Gradle Review

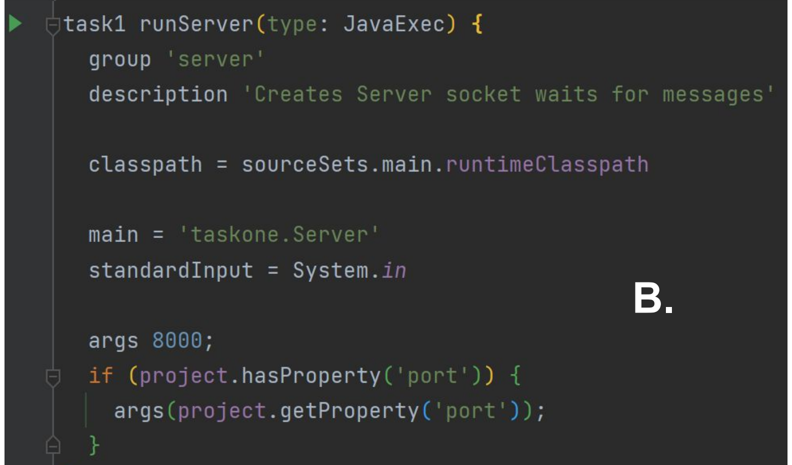
Which of the following will run the main method in /java/taskone/Server.java with `gradle runTask1`?



```
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'));
  }
}
```

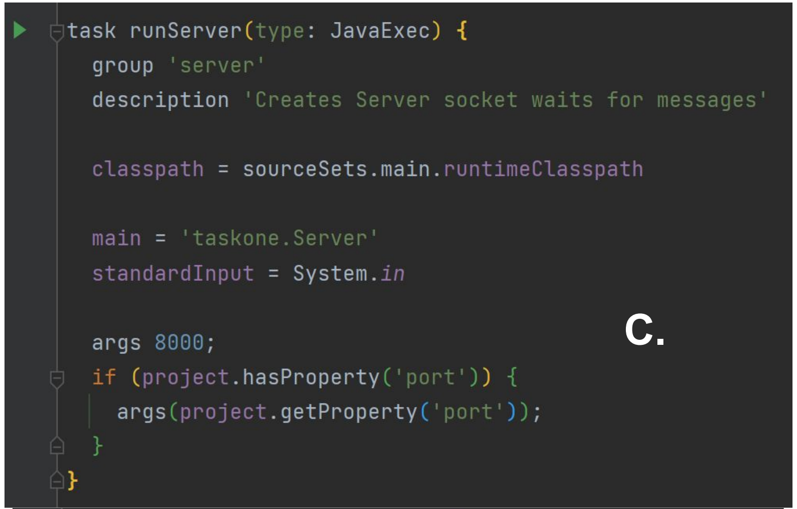
A.

Check out the recording for the solution!



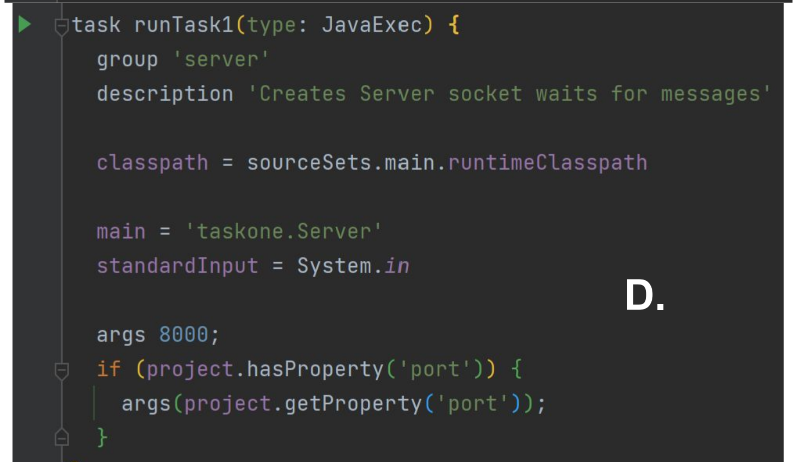
```
task1 runServer(type: JavaExec) {
  group 'server'
  description 'Creates Server socket waits for messages'

  classpath = sourceSets.main.runtimeClasspath

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

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

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.



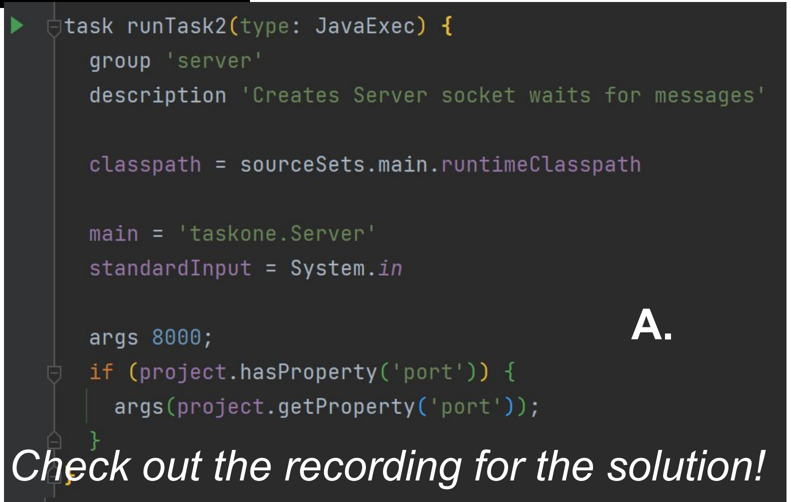
```
task runTask1(type: JavaExec) {
  group 'server'
  description 'Creates Server socket waits for messages'

  classpath = sourceSets.main.runtimeClasspath

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

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

D.

SER 321

Gradle Review

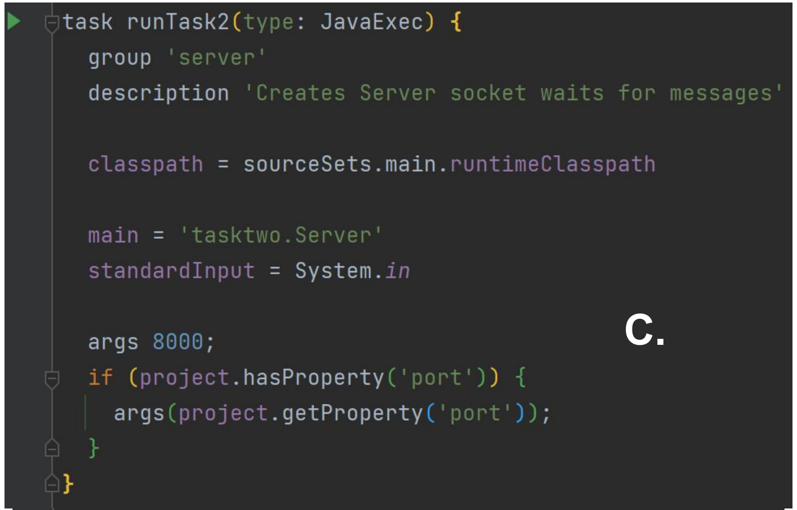
Which of the following will run the main method in /java/tasktwo/Server.java with `gradle runTask2`?



A.

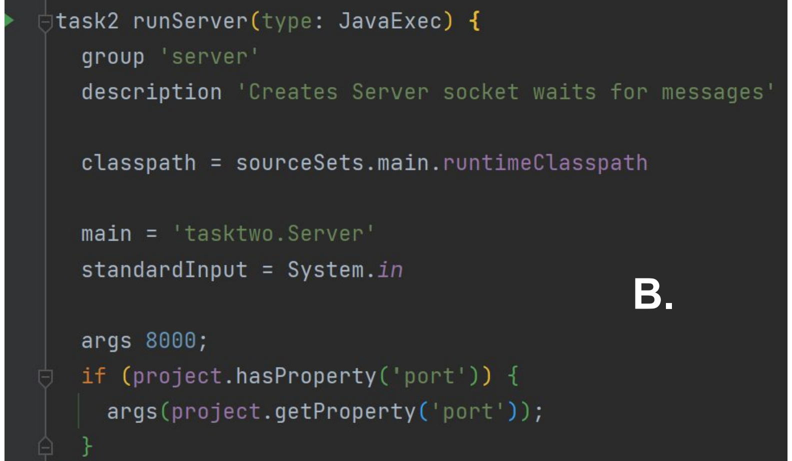
```
task runTask2(type: JavaExec) {
  group 'server'
  description 'Creates Server socket waits for messages'

  classpath = sourceSets.main.runtimeClasspath

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

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

Check out the recording for the solution!



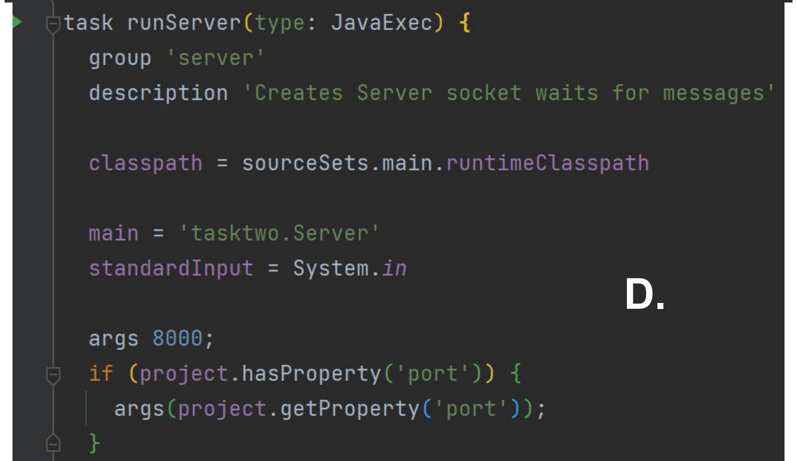
C.

```
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'));
  }
}
```



B.

```
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'));
  }
```

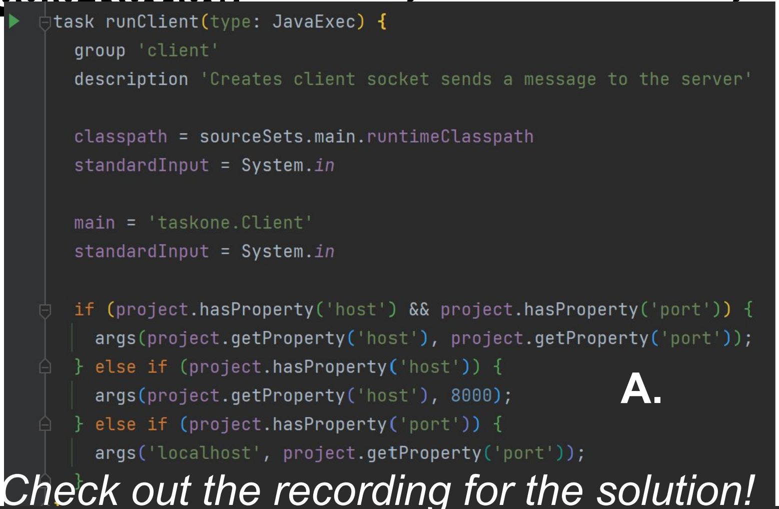


D.

```
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'));
  }
```

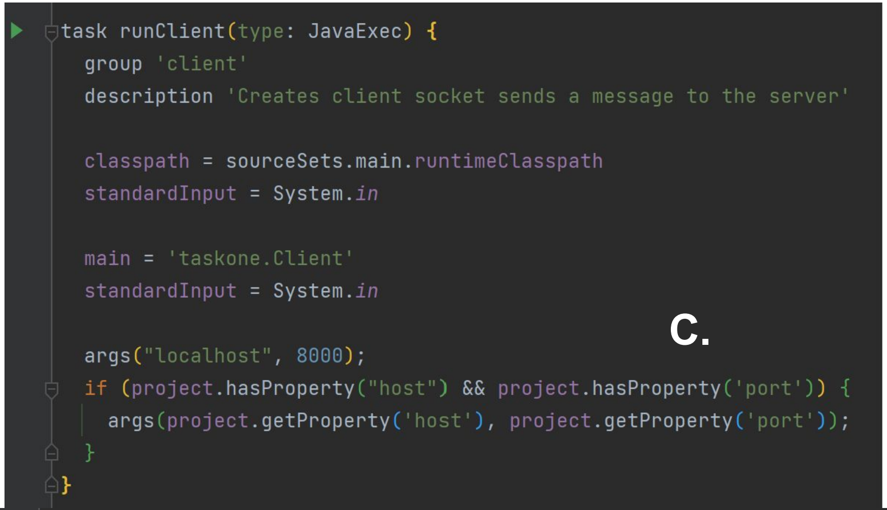
SER 321
Gradle Review

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



```
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'));
```

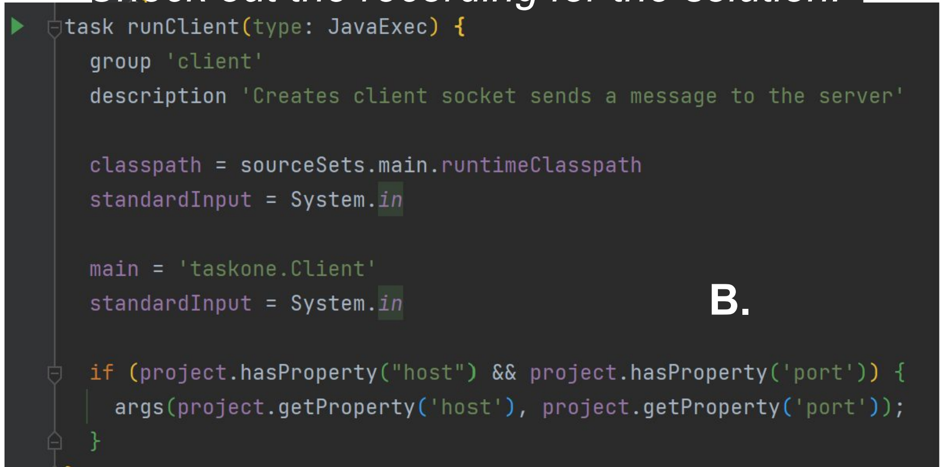
A.



```
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

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

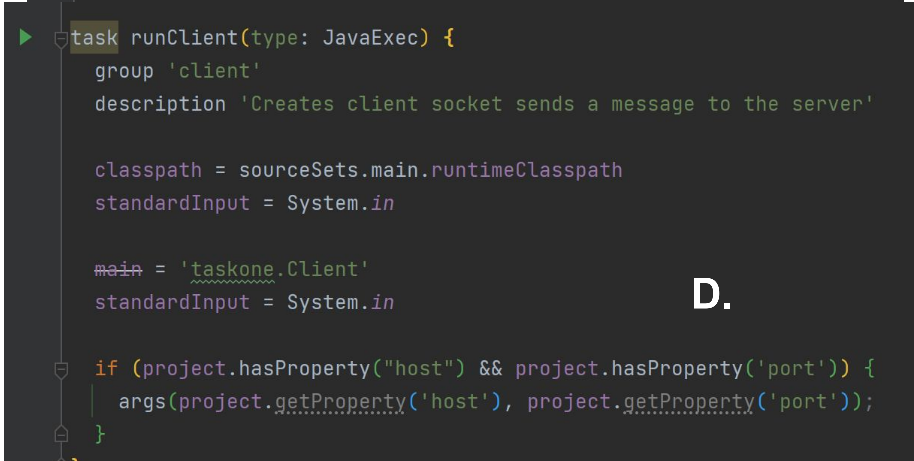
C.

Check out the recording for the solution!



```
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.

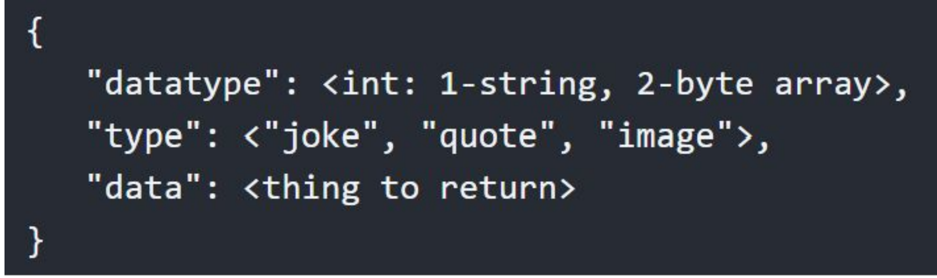


```
task runClient(type: JavaExec) {
  group 'client'
  description 'Creates client socket sends a message to the server'

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

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

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

D.

SER 321

JSON Review



```
{
    "datatype": <int: 1-string, 2-byte array>,
    "type": <"joke", "quote", "image">,
    "data": <thing to return>
}
```

Given the protocol above, which is a valid response?

A.

```
{
        “datatype”:3,
        “type”:“joke”,
        “data”:<joke>
}
```

B.

```
{
        “datatype”:1,
        “type”:“quote”,
        “data”:<quote>
}
```

C.

```
{
        “datatype”:2,
        “type”:“joke”,
        “joke”:“data”
}
```

D.

```
{
        “datatype”:2,
        “img”:“type”,
        “data”:<img>
}
```

Check out the recording for the solution!

Sockets/SimpleProtocolWithSomeErrorHandling

SER 321
JSON Review

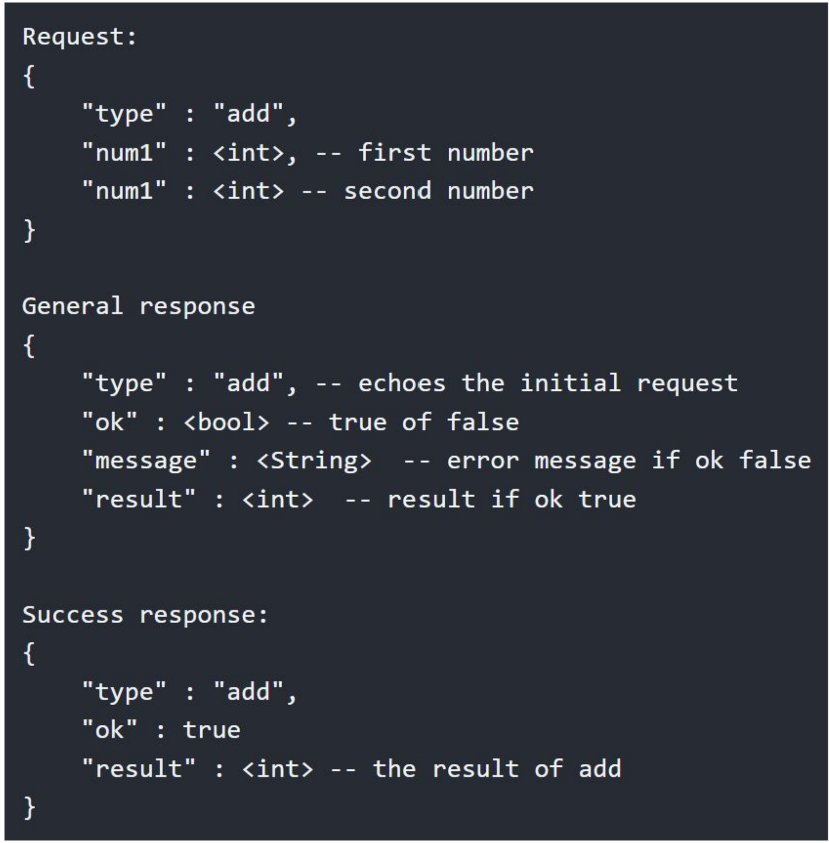
Which of the following is a invalid response?

A.
```
{
      “ok”:“false”,
      “message”:“error”
}
```

B.
```
{
      “type”:“add”,
      “ok”:“true”,
      “result”:5
}
```

C.
```
{
      “type”:“add”,
      “ok”:true,
      “result”:10
}
```

D.
```
{
      “ok”:true,
      “result”:“error”
}
```


```
Request:
{
    "type" : "add",
    "num1" : <int>, -- first number
    "num1" : <int> -- second number
}

General response
{
    "type" : "add", -- echoes the initial request
    "ok" : <bool> -- true of false
    "message" : <String>  -- error message if ok false
    "result" : <int>  -- result if ok true
}

Success response:
{
    "type" : "add",
    "ok" : true
    "result" : <int> -- the result of add
}
```

Check out the recording for the solution!

SER 321

Making the Client and Server Robust

We have some protobuff content to get through as well so we won't spend too much time on it, but let's talk about beefing up our client and server to prevent crashes.

Let's look at Activity 1 starter code together

SER 321
Protocol Buffers

Require a few steps before use - listed in the README

1. Run the following:

   ```
   gradle generateProto
   ```

2. IntelliJ users have an extra step - insert the following into build.gradle

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

SER 321
Protocol Buffers

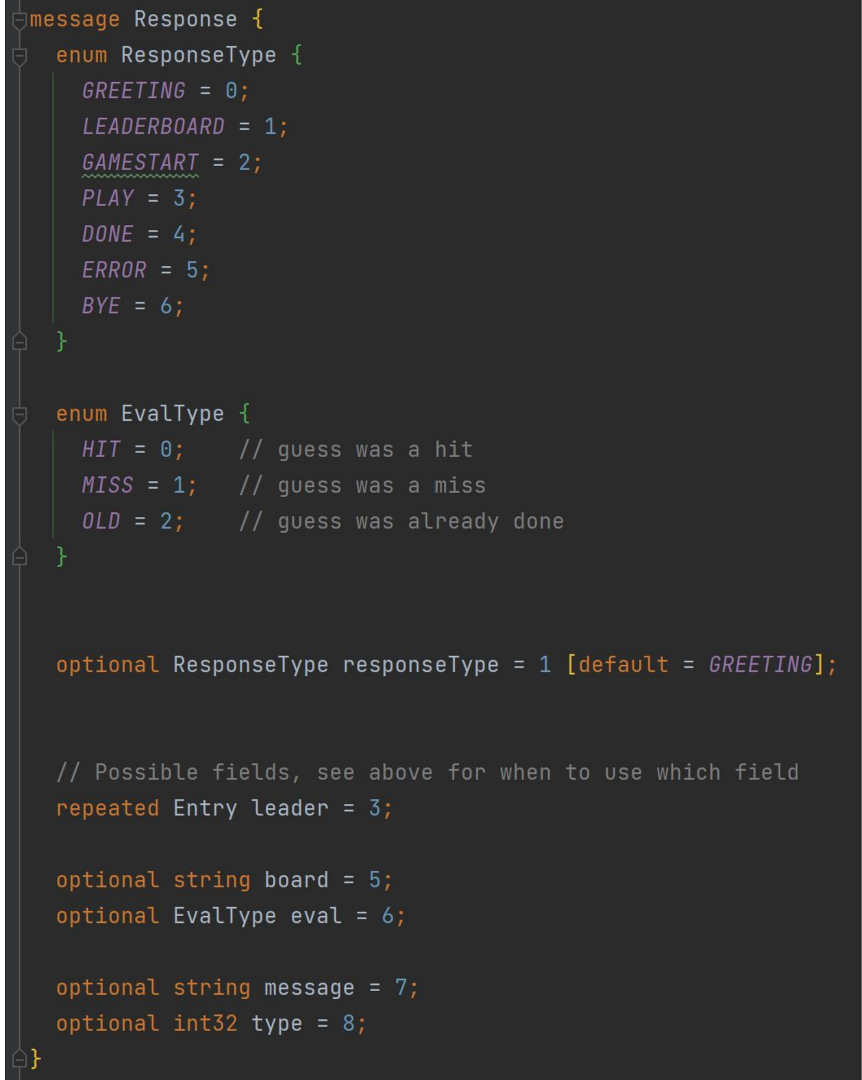
Little bit different:

- .proto files provide the language interface
- Message is the standard data structure
- Serialization and Deserialization are both handled for you
    - Can use different methods based on the input/output stream data type
    - writeTo(OutputStream) and parseFrom(InputStream)
- Will use a **Builder** to create each object

SER 321
Protocol Buffers

Defining types for use below

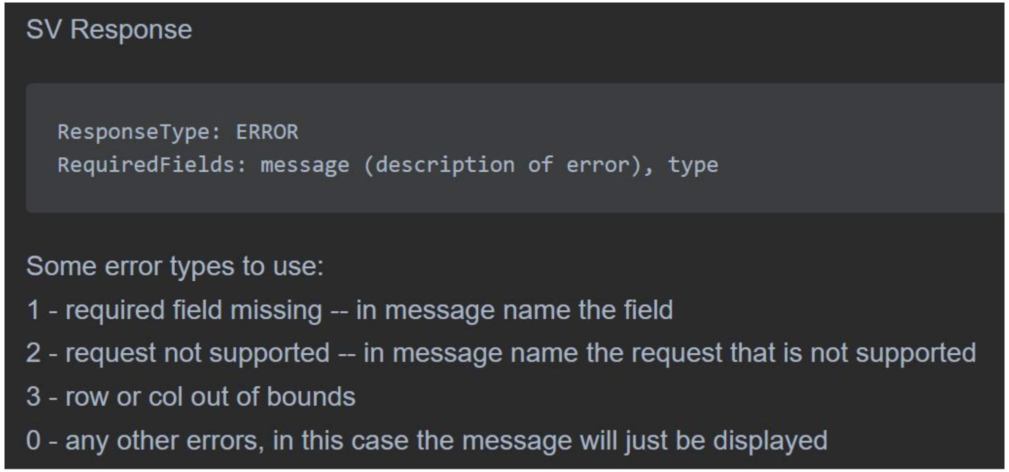
The actual response structure



```
message Response {
  enum ResponseType {
    GREETING = 0;
    LEADERBOARD = 1;
    GAMESTART = 2;
    PLAY = 3;
    DONE = 4;
    ERROR = 5;
    BYE = 6;
  }

  enum EvalType {
    HIT = 0;     // guess was a hit
    MISS = 1;    // guess was a miss
    OLD = 2;     // guess was already done
  }


  optional ResponseType responseType = 1 [default = GREETING];


  // Possible fields, see above for when to use which field
  repeated Entry leader = 3;

  optional string board = 5;
  optional EvalType eval = 6;

  optional string message = 7;
  optional int32 type = 8;
}
```

SER 321
Protocol Buffers

What would creating a Response look like?



SV Response

```
ResponseType: ERROR
RequiredFields: message (description of error), type
```

Some error types to use:

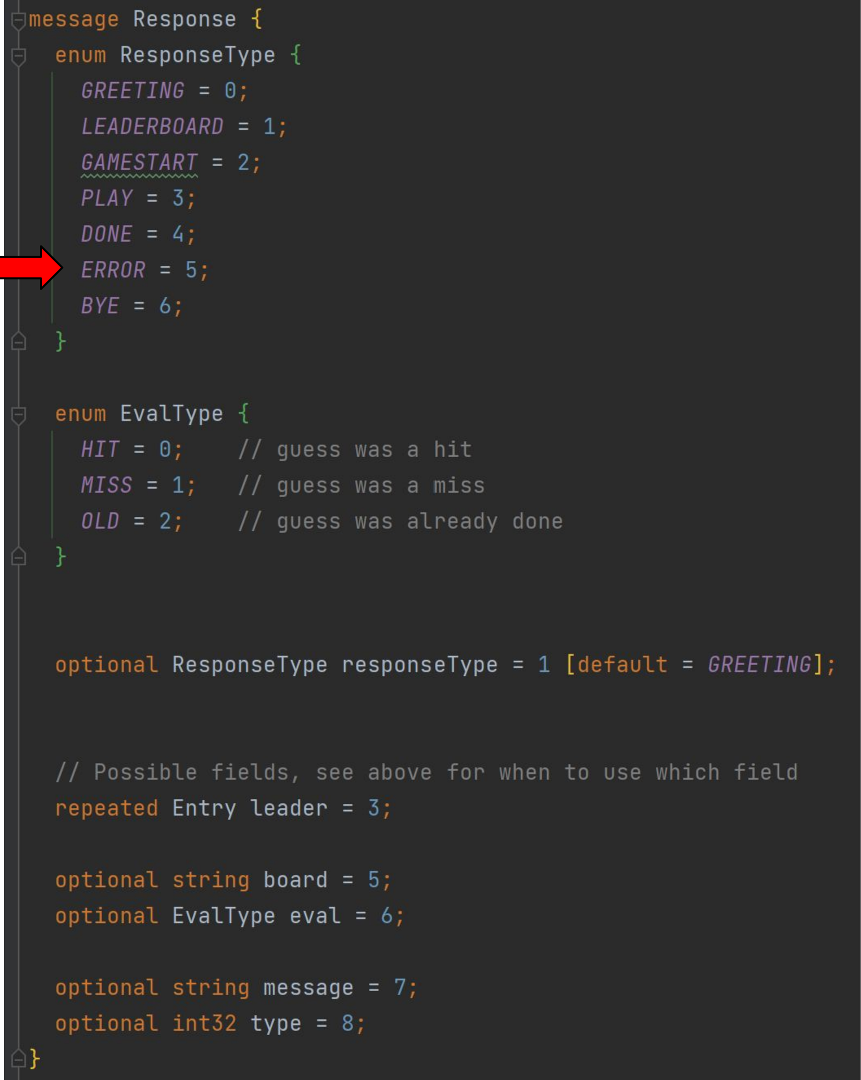
1 - required field missing -- in message name the field

2 - request not supported -- in message name the request that is not supported

3 - row or col out of bounds

0 - any other errors, in this case the message will just be displayed

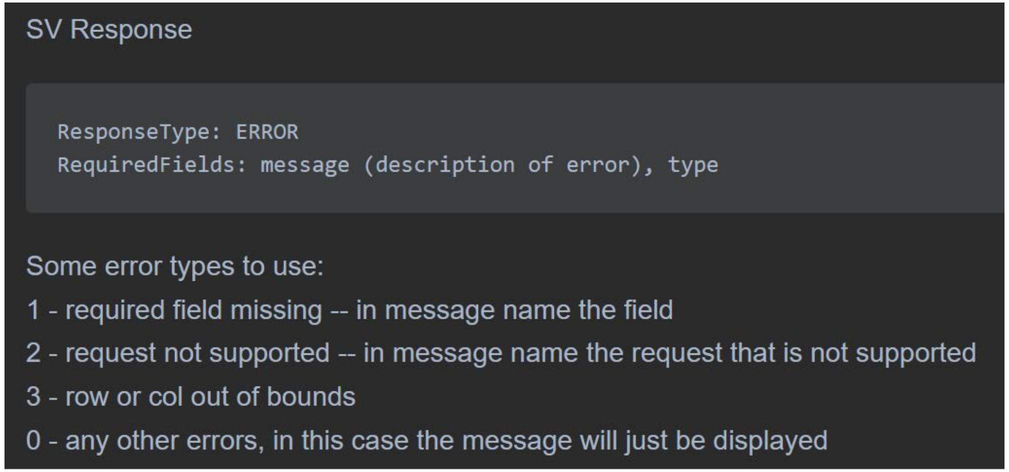
PROTOCOL.md contains the definitions



```
message Response {
  enum ResponseType {
    GREETING = 0;
    LEADERBOARD = 1;
    GAMESTART = 2;
    PLAY = 3;
    DONE = 4;
    ERROR = 5;
    BYE = 6;
  }

  enum EvalType {
    HIT = 0;    // guess was a hit
    MISS = 1;   // guess was a miss
    OLD = 2;    // guess was already done
  }


  optional ResponseType responseType = 1 [default = GREETING];


  // Possible fields, see above for when to use which field
  repeated Entry leader = 3;

  optional string board = 5;
  optional EvalType eval = 6;

  optional string message = 7;
  optional int32 type = 8;
}
```

SER 321
Protocol Buffers

What would creating a Response look like?



SV Response

```
ResponseType: ERROR
RequiredFields: message (description of error), type
```

Some error types to use:

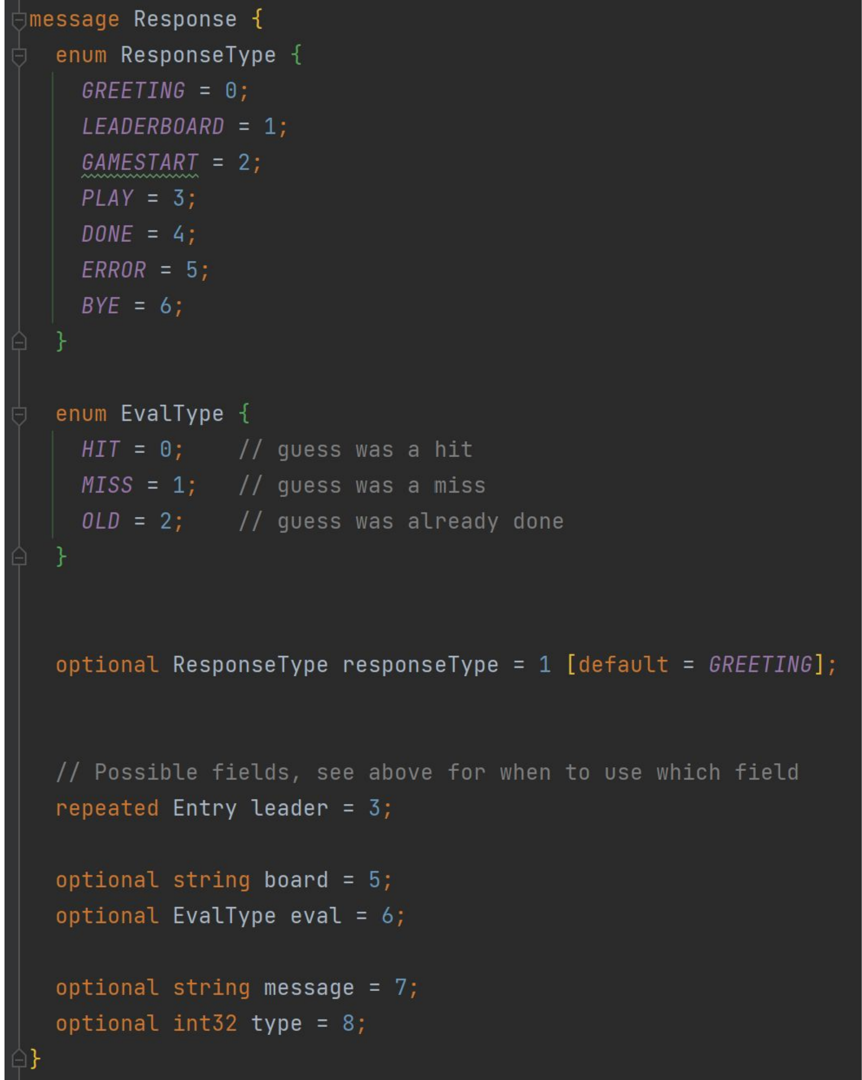
1 - required field missing -- in message name the field

2 - request not supported -- in message name the request that is not supported

3 - row or col out of bounds

0 - any other errors, in this case the message will just be displayed

```
Response resp = Response.newBuilder()
     .setResponseType(Response.ResponseType.ERROR)
     .setMessage("Error Example!")
     .setType(0)
     .build();
```



```
message Response {
  enum ResponseType {
    GREETING = 0;
    LEADERBOARD = 1;
    GAMESTART = 2;
    PLAY = 3;
    DONE = 4;
    ERROR = 5;
    BYE = 6;
  }

  enum EvalType {
    HIT = 0;    // guess was a hit
    MISS = 1;   // guess was a miss
    OLD = 2;    // guess was already done
  }


  optional ResponseType responseType = 1 [default = GREETING];


  // Possible fields, see above for when to use which field
  repeated Entry leader = 3;

  optional string board = 5;
  optional EvalType eval = 6;

  optional string message = 7;
  optional int32 type = 8;
}
```

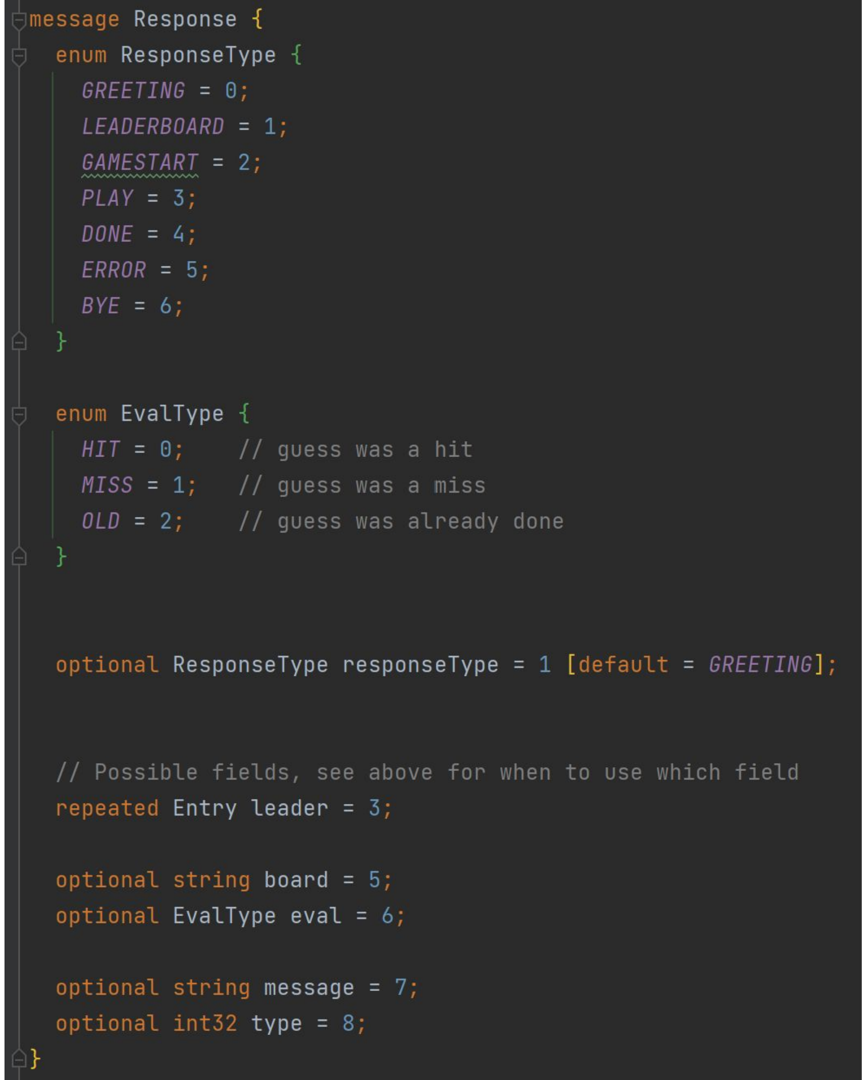
SER 321
Protocol Buffers

What if I don’t have all the information right now?

```
ResponseBuilder respBuild = Response.newBuilder()
     .setResponseType(Response.ResponseType.ERROR)
     .setMessage(“Error Example!”)
     .setType(0);
```

Then when you are ready use:

```
Response resp = respBuild.build();
```



```
message Response {
  enum ResponseType {
    GREETING = 0;
    LEADERBOARD = 1;
    GAMESTART = 2;
    PLAY = 3;
    DONE = 4;
    ERROR = 5;
    BYE = 6;
  }

  enum EvalType {
    HIT = 0;     // guess was a hit
    MISS = 1;    // guess was a miss
    OLD = 2;     // guess was already done
  }


  optional ResponseType responseType = 1 [default = GREETING];


  // Possible fields, see above for when to use which field
  repeated Entry leader = 3;

  optional string board = 5;
  optional EvalType eval = 6;

  optional string message = 7;
  optional int32 type = 8;
}
```

SER 321

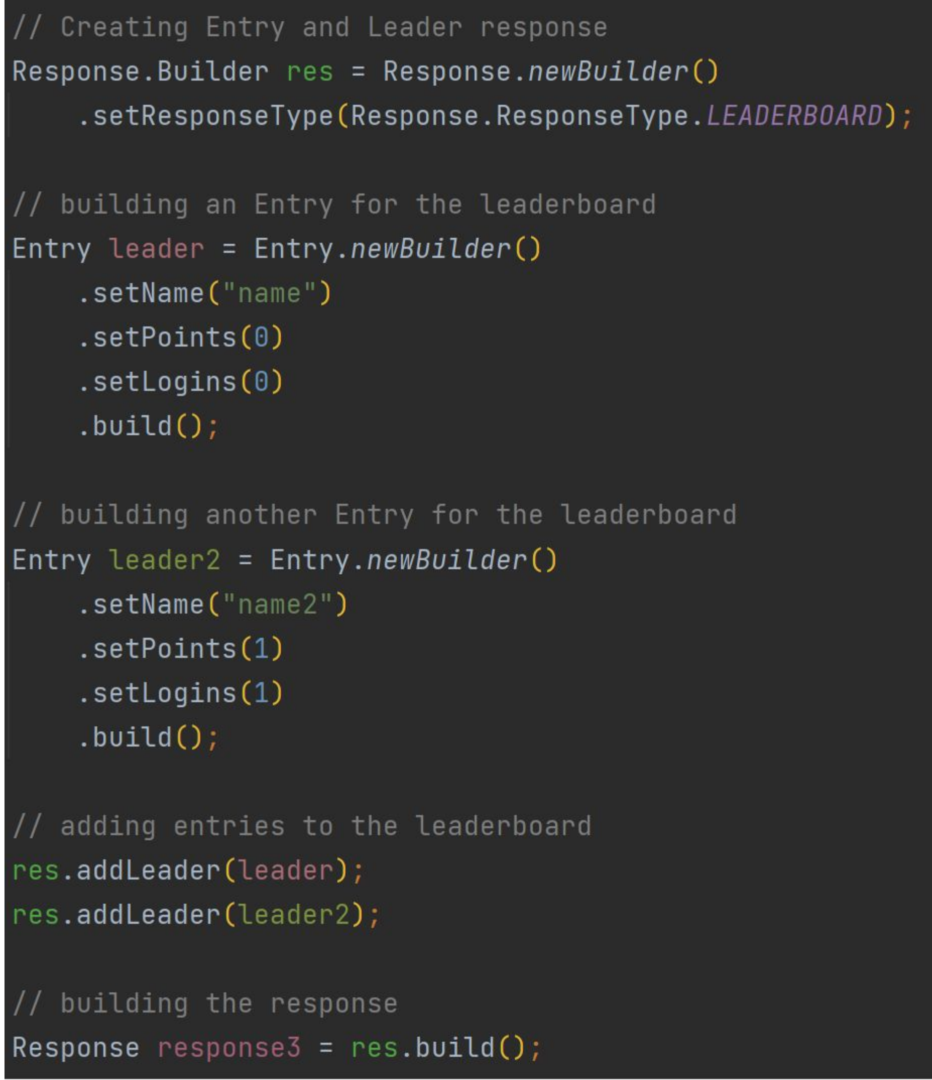
Protocol Buffers

What about repeated fields?

First, create the object

Then just add them to the object!

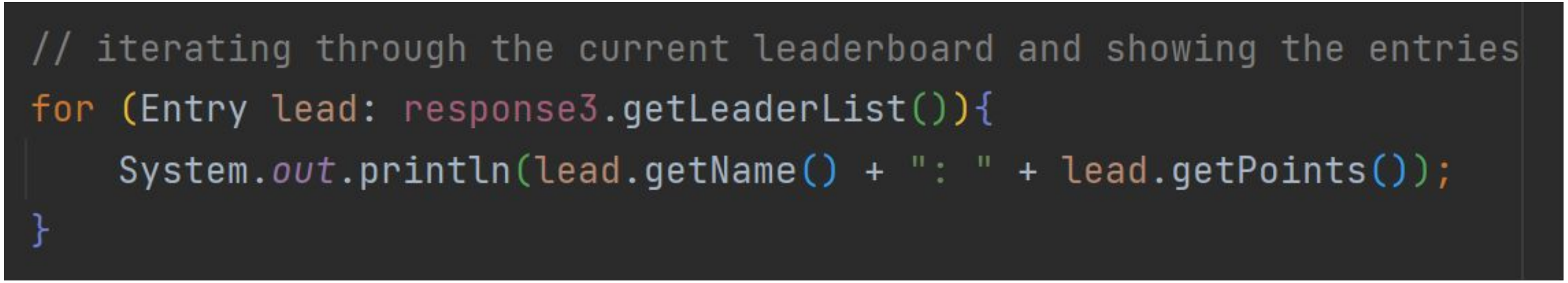
No need to worry about structure



```java
// Creating Entry and Leader response
Response.Builder res = Response.newBuilder()
    .setResponseType(Response.ResponseType.LEADERBOARD);

// building an Entry for the leaderboard
Entry leader = Entry.newBuilder()
    .setName("name")
    .setPoints(0)
    .setLogins(0)
    .build();

// building another Entry for the leaderboard
Entry leader2 = Entry.newBuilder()
    .setName("name2")
    .setPoints(1)
    .setLogins(1)
    .build();

// adding entries to the leaderboard
res.addLeader(leader);
res.addLeader(leader2);

// building the response
Response response3 = res.build();
```

SER 321
Protocol Buffers

What about **READING** repeated fields?



```java
// iterating through the current leaderboard and showing the entries
for (Entry lead: response3.getLeaderList()){
    System.out.println(lead.getName() + ": " + lead.getPoints());
}
```

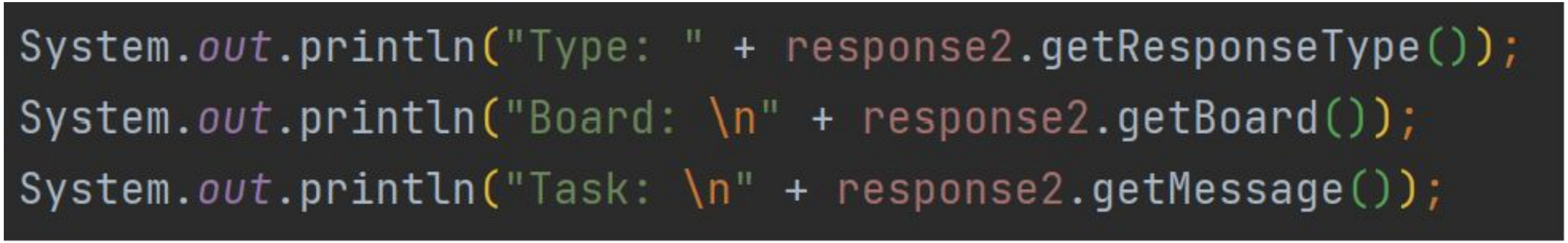
Your **only** option is an enhanced for loop

You will use a getter to obtain a List containing the repeated data

SER 321
Protocol Buffers

What about reading regular fields?

More getters!



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

SER 321
Protocol Buffers

Where did it all come from?

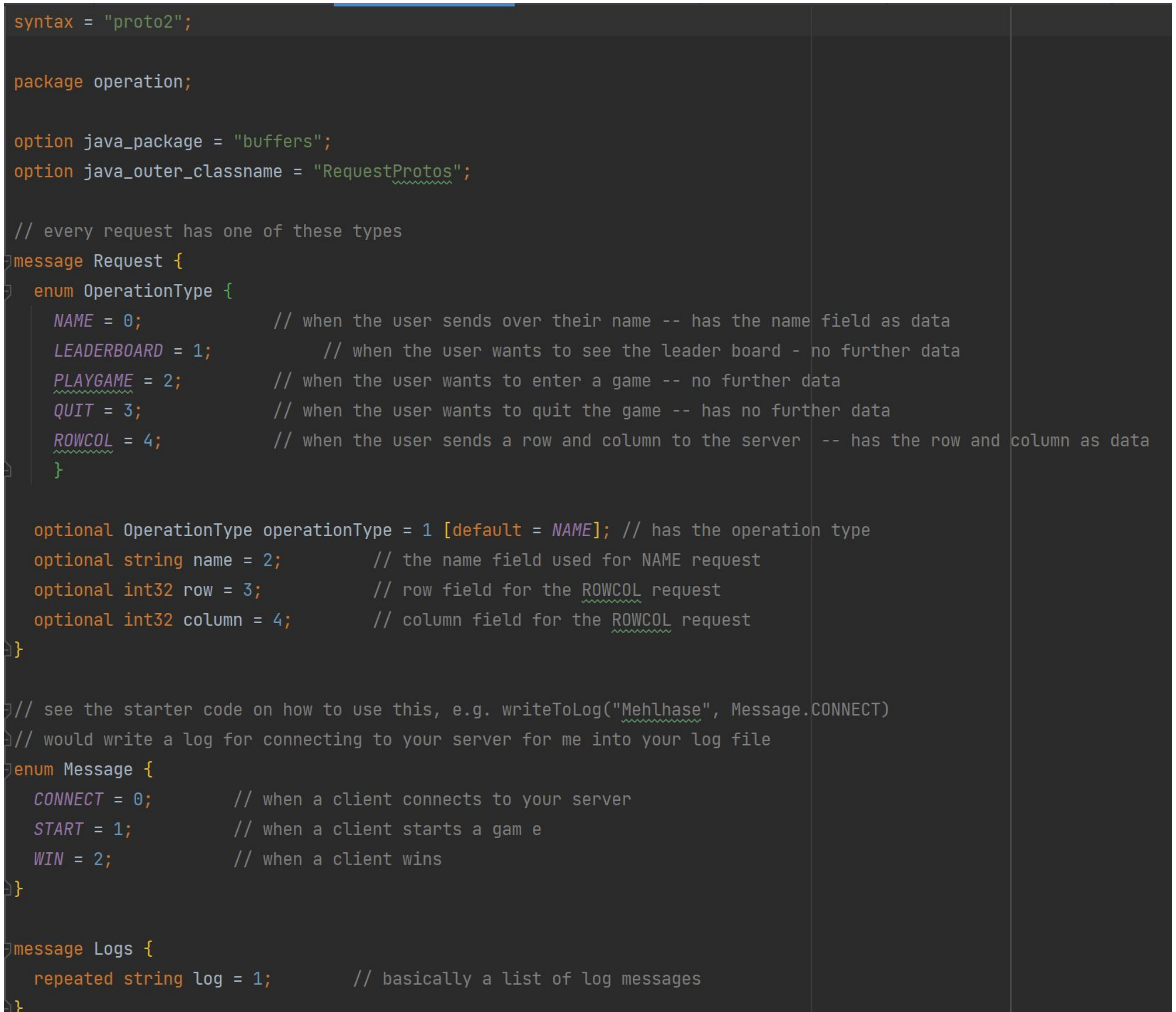
When you ran `gradle generateProto` all the code was created according to the .proto file!

Future changes to the structure (.proto) would be much easier!

NOT ALLOWED FOR THIS COURSE!!

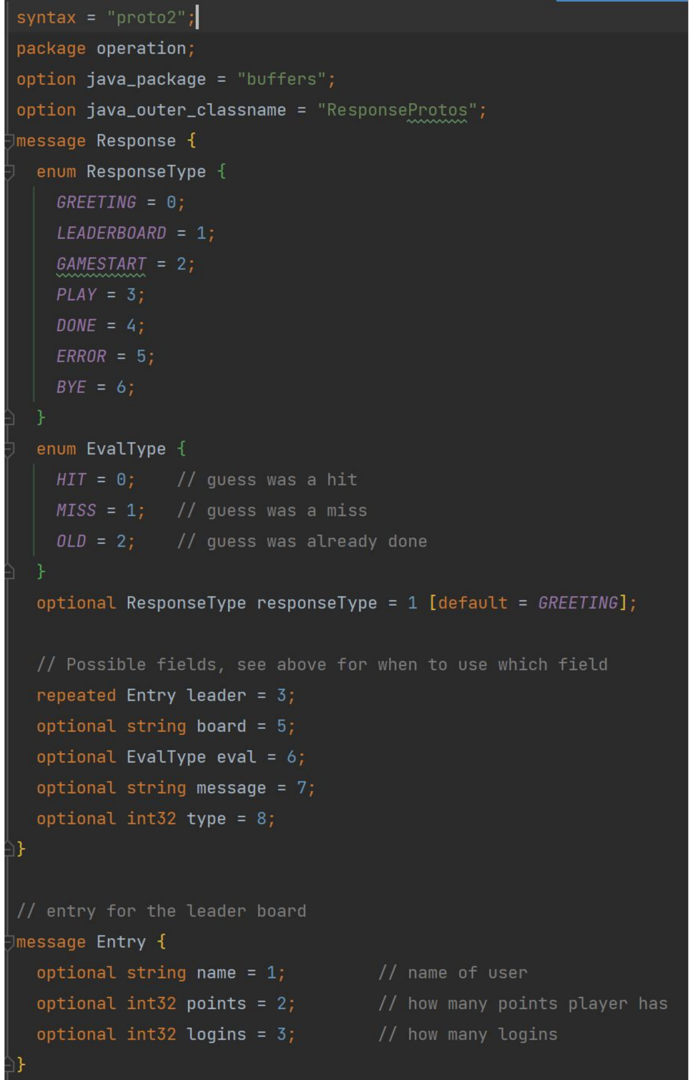
SER 321

Request Proto



```
syntax = "proto2";

package operation;

option java_package = "buffers";
option java_outer_classname = "RequestProtos";

// every request has one of these types
message Request {
  enum OperationType {
    NAME = 0;                  // when the user sends over their name -- has the name field as data
    LEADERBOARD = 1;               // when the user wants to see the leader board - no further data
    PLAYGAME = 2;              // when the user wants to enter a game -- no further data
    QUIT = 3;                  // when the user wants to quit the game -- has no further data
    ROWCOL = 4;                // when the user sends a row and column to the server  -- has the row and column as data
    }

  optional OperationType operationType = 1 [default = NAME]; // has the operation type
  optional string name = 2;          // the name field used for NAME request
  optional int32 row = 3;            // row field for the ROWCOL request
  optional int32 column = 4;         // column field for the ROWCOL request
}

// see the starter code on how to use this, e.g. writeToLog("Mehlhase", Message.CONNECT)
// would write a log for connecting to your server for me into your log file
enum Message {
  CONNECT = 0;        // when a client connects to your server
  START = 1;          // when a client starts a gam e
  WIN = 2;            // when a client wins
}

message Logs {
  repeated string log = 1;        // basically a list of log messages
}
```

SER 321
Response Proto



```
syntax = "proto2";
package operation;
option java_package = "buffers";
option java_outer_classname = "ResponseProtos";
message Response {
  enum ResponseType {
    GREETING = 0;
    LEADERBOARD = 1;
    GAMESTART = 2;
    PLAY = 3;
    DONE = 4;
    ERROR = 5;
    BYE = 6;
  }
  enum EvalType {
    HIT = 0;    // guess was a hit
    MISS = 1;   // guess was a miss
    OLD = 2;    // guess was already done
  }
  optional ResponseType responseType = 1 [default = GREETING];

  // Possible fields, see above for when to use which field
  repeated Entry leader = 3;
  optional string board = 5;
  optional EvalType eval = 6;
  optional string message = 7;
  optional int32 type = 8;
}

// entry for the leader board
message Entry {
  optional string name = 1;          // name of user
  optional int32 points = 2;         // how many points player has
  optional int32 logins = 3;         // how many logins
}
```

SER 321

Starter Code

Let’s look at some of the starter code for Activity 2 together

Questions?

Survey:

https://bit.ly/asn_survey

Upcoming Events

SI Sessions:

- Sunday September 17th 6:00 pm MST

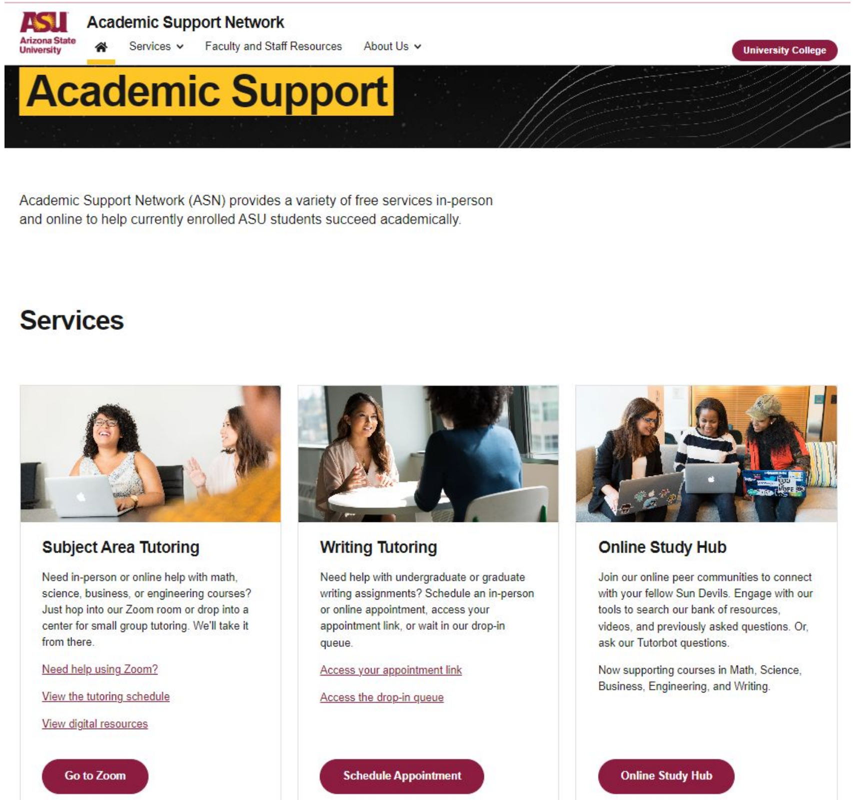
Review Sessions:

- 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

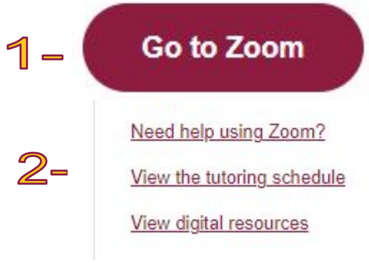
Schedule Appointment

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

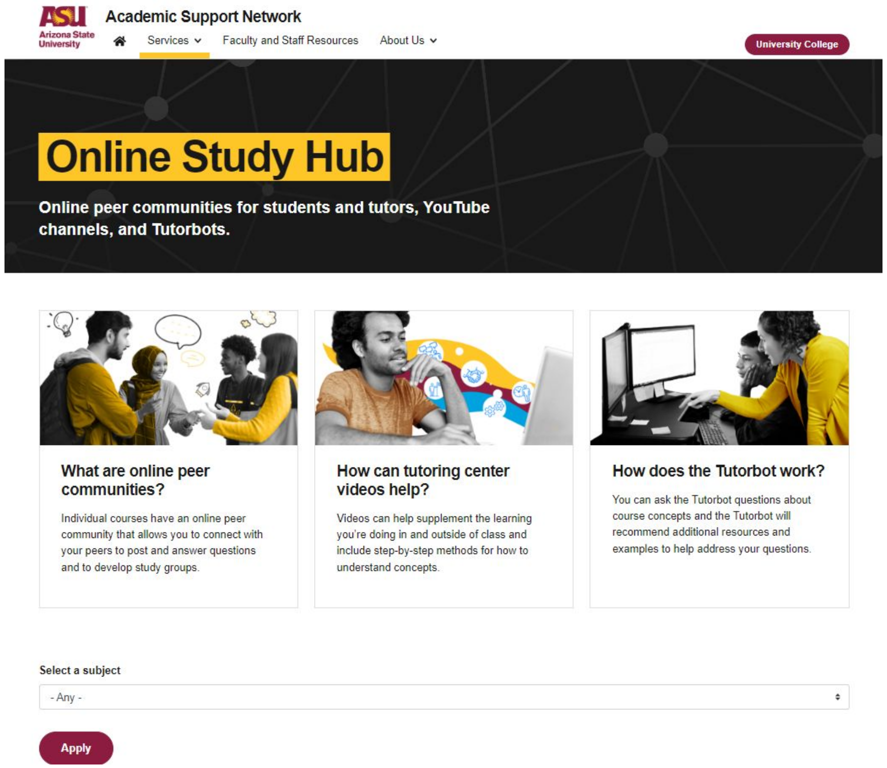


1. **Click on 'Go to Zoom' to log onto our Online Tutoring Center.**
2. **Click on 'View the tutoring schedule' to see when tutors are available for specific courses.**

More Questions?

Check out our other resources!

tutoring.asu.edu/online-study-hub


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.

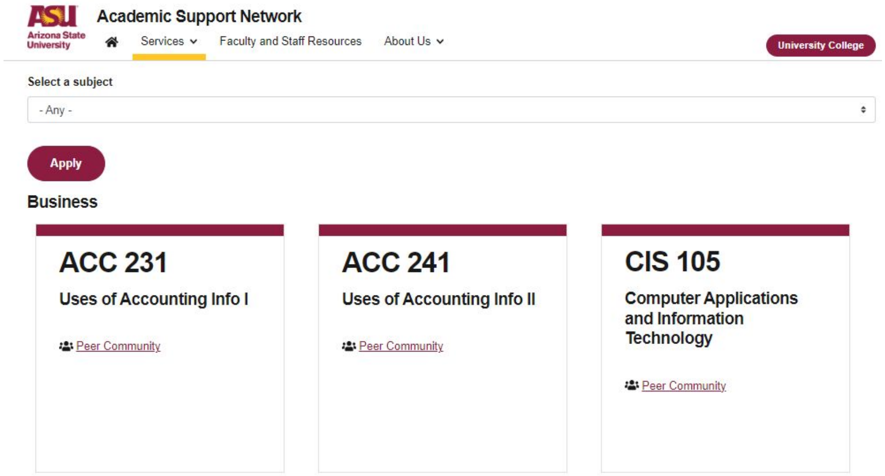
How does the Tutorbot work?

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	ACC 241	CIS 105
Uses of Accounting Info I	Uses of Accounting Info II	Computer Applications and Information Technology
Peer Community	Peer Community	Peer Community

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

Additional Resources