SER 321 A Session

SI Session

Monday September 11th 2023

6:00 - 7:00 pm MST

Agenda

- Serialization
- JSON Review
- Using JSON
- Protocol Buffers
- Using Protobufs

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
Serialization

What is it? 🤔

“Translating data structures or object states for storage or transmission”

Main forms:	Data Format
• XML	Text
• JSON	Text
• Java Serialization (Java Objects)	Binary
• Protocol Buffers	Binary

SER 321
Serialization

Why do I care about the transmission data type?

Changes how you handle the data!

Remember NetworkUtils and JsonUtils?

```
public class JsonUtils {
  public static JSONObject fromByteArray(byte[] bytes) {
    String jsonString = new String(bytes);
    return new JSONObject(jsonString);
  }

  public static byte[] toByteArray(JSONObject object) {
    return object.toString().getBytes();
  }
}
```

SER 321
Serialization

```
byte[] messageBytes = NetworkUtils.Receive(in);
JSONObject message = JsonUtils.fromByteArray(messageBytes);
```

Why do I care about the transmission data type?

Changes how you handle the data!

Remember NetworkUtils and JsonUtils?

Converted data types **for** us

```
public class JsonUtils {
  public static JSONObject fromByteArray(byte[] bytes) {
    String jsonString = new String(bytes);
    return new JSONObject(jsonString);
  }

  public static byte[] toByteArray(JSONObject object) {
    return object.toString().getBytes();
  }
}
```

What stream do I use??

- Buffered Streams
- Data Streams
- Object Streams

What stream do I use??

- Buffered Streams

Bytes

- Data Streams

- Object Streams

What stream do I use??

- Buffered Streams — Bytes
- Data Streams — Primitive Data Types
- Object Streams

What stream do I use??

● Buffered Streams	Bytes
● Data Streams	Primitive Data Types
● Object Streams	Objects

SER 321

Streams

What stream do I use??

- **B**uffered Streams — **B**ytes
- **Data** Streams — Primitive **Data** Types
- **Object** Streams — **Objects**

SER 321

Streams

NetworkUtils uses InputStream and OutputStream

Abstract superclasses of input/output **byte** streams

NetworkUtils allows for easy conversion of JSON!

SER 321

JSON Review

What do we know about JSON?

- JavaScript Object Notation
- Contains name-value pairs
- Restricted value types

Number - int or double
String
boolean
null
Object
Array

SER 321

JSON Review

Object starts and ends with a bracket { }

Contains either whitespace, or a **member**

A member is one “`name`” : “`value`” pair

This is an object:

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

SER 321
JSON Review

```
{ “nameOne” : “valueOne”, “nameTwo: “valueTwo” }
```

The value for a given member **can** contain another object.

```
{
    “nameOne” : “valueOne”,
    “nameTwo: “valueTwo”
}
```

```
{
    “nameOne”, : “valueOne”,
    “objName” : {
        “nestedName” : “nestedValue”,
        “nestedName2” : “nestedValue2”
    }
}
```

SER 321
JSON Review

```
{ “nameOne” : “valueOne”, “nameTwo: “valueTwo” }
```

The value for a given member **can** contain an array of objects.

```
{
    “nameOne” : “valueOne”,
    “nameTwo: “valueTwo”
}
```

```
{
    “nameOne”, : “valueOne”,
    “objName” : {
        “nestedName” : “nestedValue”,
        “nestedName2” : “nestedValue2”
    }
}
```

```
{
    “nameOne”, : “valueOne”,
    “arrName” : [
        {
            “obj1Member1” : “value”,
            “obj1Member2” : “value2”
        },
        {
            “obj2Member2” : “value”,
            “obj2Member2” : “value2”
        }
    ]
}
```

Sockets/SimpleProtocolWithSomeErrorHandling

SER 321
JSON Mini-Quiz

Which of the following is a valid add response?

A.
```
{
    “type”:“add”,
    “ok”:“no”,
    “message”:“error”
}
```

B.
```
{
    “type”:“add”,
    “ok”:“true”,
    “message”:“none”
}
```

C.
```
{
    “type”:“add”,
    “ok”:false,
    “result”:-1
}
```

D.
```
{
    “ok”:false,
    “message”:“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
}
```

Sockets/SimpleProtocolWithSomeErrorHandling

SER 321
JSON Mini-Quiz

Which of the following is a valid add response?

A.
```
{
    “type”:“add”,
    “ok”:“no”,
    “message”:“error”
}
```

B.
```
{
    “type”:“add”,
    “ok”:“true”,
    “message”:“none”
}
```

C.
```
{
    “type”:“add”,
    “ok”:false,
    “result”:-1
}
```

D.
```
{
    “ok”:false,
    “message”:“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
}
```

Full code for remove here ->

```java
/**
 * Function JSONObject remove()
 */
1 usage
public static JSONObject remove() {
    int inNum = 0;
    JSONObject request = new JSONObject();
    try {
        System.out.print("Please input the integer for removal: ");
        inNum = stdin.read();
    } catch (IOException e) {
        e.printStackTrace();
    }
    request.put("selected", 2);
    request.put("data", inNum);
    return request;
}
```

SER 321
Using JSON

org.json

JSONObjects and JSONArrays make life much easier!

Looking at the starter code for Assignment 4 Activity 1 in Client.java remove():

1. Create JSONObject

```java
JSONObject request = new JSONObject();
```

2. Add Members

```java
request.put("selected", 2);
request.put("data", inNum);
```

3. Done!

JSONObject

SER 321
Using JSON

org.json

What about reading from JSONObjects or JSONArrays?

You need to know the *name* (or the *key*) for the member you want

1. From the JSONObject

```
byte[] responseBytes = NetworkUtils.receive(in);
JSONObject response = JsonUtils.fromByteArray(responseBytes);
```

2. **You should** check if the key exists

```
has(String key)
```

Determine if the JSONObject contains a specific key.

3. Extract the value

```
System.out.println("The response from the server: ");
System.out.println("datatype: " + response.getString( key: "type"));
System.out.println("data: " + response.getString( key: "data"));
System.out.println();
String typeStr = (String) response.getString( key: "type");
```

NOTE: this was the last slide covered in the 9.11.23 Session

JSONObject

SER 321

Using JSON

Note that the `get` methods are based on the data type of the *value*

```
("data: " + response.getString( key: "data"));
```

You must use the correct method for whatever data type you are fetching

Just like always right?

`getBoolean(String key)`
Get the boolean value associated with a key.

`getDouble(String key)`
Get the double value associated with a key.

`getEnum(Class<E> clazz, String key)`
Get the enum value associated with a key.

`getFloat(String key)`
Get the float value associated with a key.

`getInt(String key)`
Get the int value associated with a key.

`getJSONArray(String key)`
Get the JSONArray value associated with a key.

`getJSONObject(String key)`
Get the JSONObject value associated with a key.

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 is handled for you
  - Will 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 response 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?

```
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

```
// 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?

```
// 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!!

Questions?

Survey:

https://bit.ly/asn_survey

33

Upcoming Events

SI Sessions:

- Wednesday, September 13th 2023 at 6:00 pm MST
  - JSON Organization
  - Protobuf Organization
  - Assignment Specifics

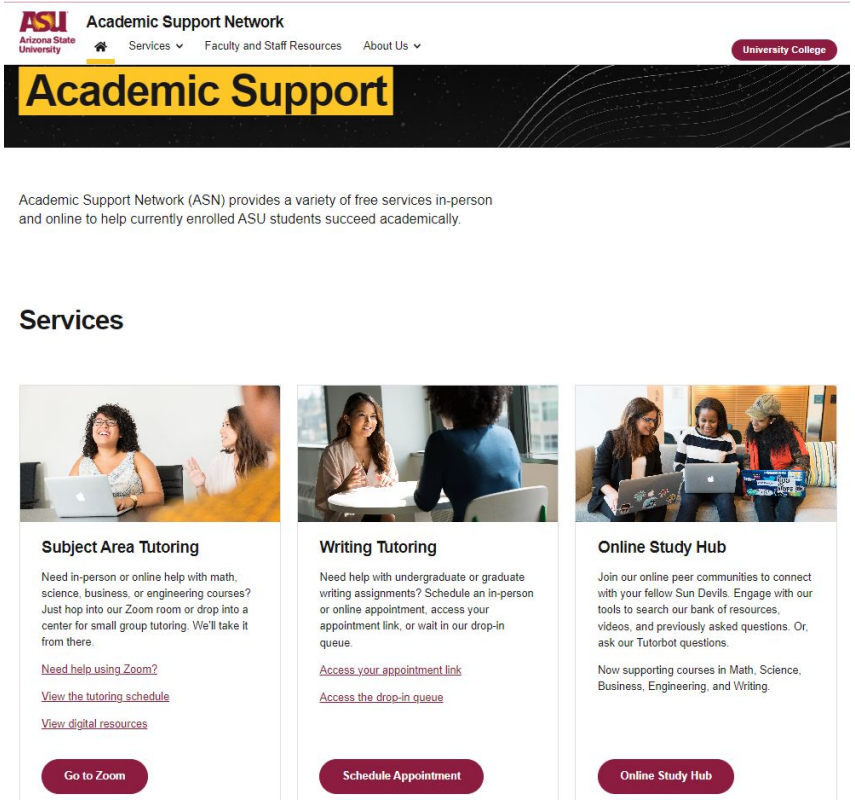
Review Sessions:

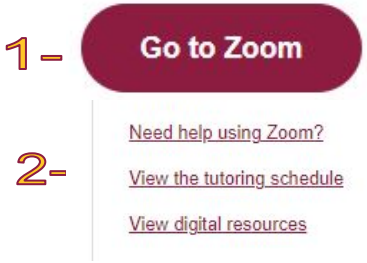
- TBD

More Questions?

Check out our other resources!

tutoring.asu.edu



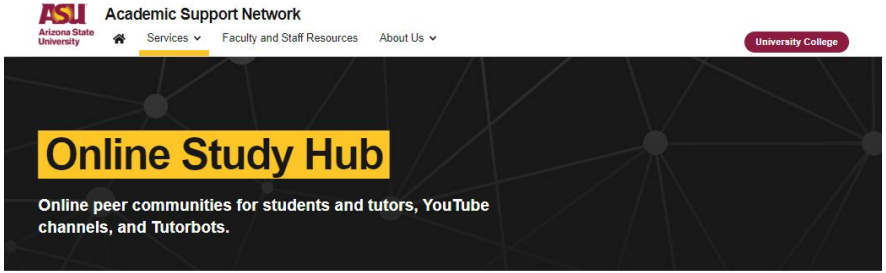


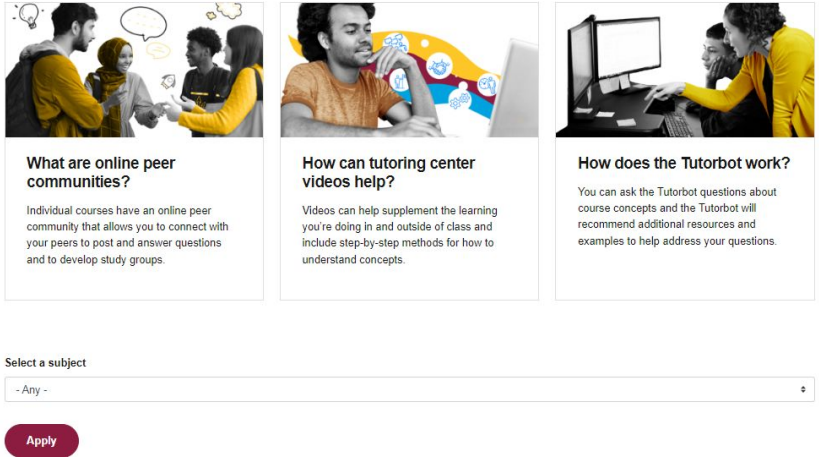
1. **Click on 'Go to Zoom' to log onto our Online Tutoring Center.**
2. **Click on 'View the tutoring schedule' to see when tutors are available for specific courses.**

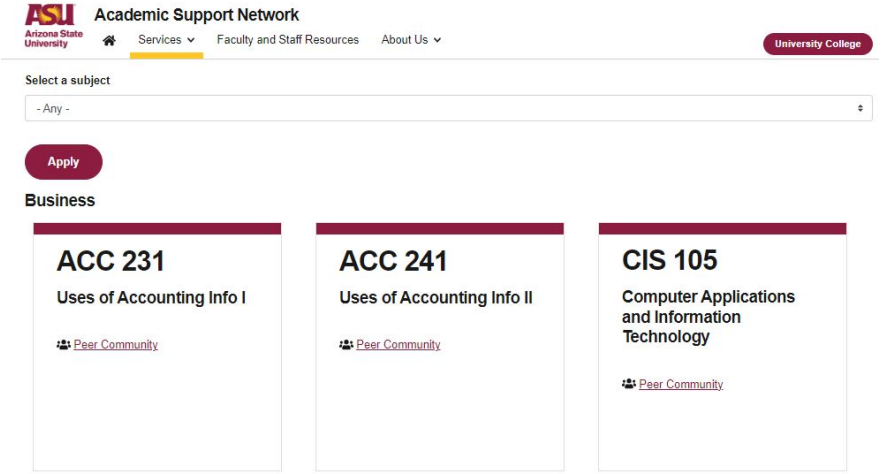
More Questions?

Check out our other resources!

tutoring.asu.edu/online-study-hub







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

Additional Resources

JSON Reference

JSON Specification