Lab Exercise 1
Getting Started with iOS

Description

You are to construct a single view (scene) app for **Swift/iOS** and another for **Java/Android** to display **Place** information as described below. This assignment and some others this semester build on **Geocoding Services**, such as those provided by Texas A&M Geoservices (Links to an external site.). These are common services utilized by many mobile and web apps where mapping and geo-location are required. You will not be required to subscribe to any **Geocoding Services** to complete this or other assignments this semester. In this app, you will be defining a class for a **place description**. In future apps, we'll create an underlying model for apps that use places, and explore different approaches for representing this model. We will look at multi-view apps, navigation among views, databases, networking, mapping, and location services, for example.

Both of the apps you develop for this assignment should display the information of a **PlaceDescription** as will fit in a single view. Both should be single-view apps. For our purposes a place is a simplified holder for information about geographic places of interest, such as a waypoint, and includes at least the following information:

- **Name**. A simple string, which is unique among all names of places for this user. Usually one or two words.
- **Description**. Text providing descriptive information about the place.
- **Category**. Text describing the type of place this entry describes. Sample categories are **residence**, **travel**, and **hike**.
- **Address Title**. This field is equivalent to the first line that would appear in an address, sometimes called the **recipient line**. It indicates the individual or organization to which the address pertains. This field is generally not used in a geocoding lookup.
- **Address Street, City, State, Country, Zip Code**. The rest of an address. This includes the street address, city or town, the state, and optionally country, and/or zip code. Although the examples indicate otherwise, you may assume **USA** address format only.
- **elevation**. Feet mean sea level elevation of the place. This field is also generally not used in geocoding services but is useful in route planning.
- **latitude**. Degrees latitude, using a single double value to represent. Lines of equal latitude run parallel to the Equator. Values range from -90.0 to +90.0. Negative values refer to the southern hemisphere and positive values to the northern hemisphere.
- **longitude**. Degrees of longitude, using a single double value. Values range from -180.0 to +180.0. Lines of equal longitude run perpendicular to the Equator. Negative values increase west from the Prime Meridian, which is longitude 0.0 and is located in Greenwich England. Positive values increase east from the Prime Meridian. 180.0 (plus and minus) is the International Date Line.
- **Example**. Here is an example **Place Description** represented in **JSON**:

```
{
    "name" : "ASU-Poly",
```

```
    "description" : "Home of ASU's Software Engineering Programs",
    "category" : "School",
    "address-title" : "ASU Software Engineering",
    "address-street" : "7171 E Sonoran Arroyo Mall\nPeralta Hall 230\nMesa AZ 85212",
    "elevation" : 1384.0,
    "latitude" : 33.306388,
    "longitude" : -111.679121
}
```

You are to construct a **PlaceDescription** class for each app. **PlaceDescription** is a container for the information described above. Through properties, methods, or visibility controls, provide that users of the class may access and set values of each field.

In future assignments, your class should provide the ability to calculate the **great-circle spherical distance** between two **Place Description**s, as well as an **initial heading**. You can use the following references (or any others you find) to see how to calculate these values using available math packages. See Wikipedia Great Circle Distance (Links to an external site.) And, another reference which expresses the same in **javascript**: Java Script Calculations (Links to an external site.) You should understand the difference between **Great Circle** and **Rhumb-line** distances. One reference showing the difference is Wikipedia Rhumb Line (Links to an external site.).

These assignments also require your **Place Description** to serialize to and from a **Json** representation. You should build that into your class definition to aid in initializing a place description object for display in your app.

To do so, your class would include a constructor that builds a **PlaceDescription** object from a string of the **JSON** form shown above. You may also want to include a **toJsonString** method that returns a string of the **JSON** in the same format.

Your app should have a single view, and in that view display, the fields referenced above on a single screen.

Make the following additions to the app demonstrated.

Your UI should primarily utilize the layouts and simple widgets introduced in the class, and you may form it based on the example given in class (or Unit 1 first app videos). Your solutions should follow the documentation and packaging constraints outlined below.

1. Place all of your application source code into the package named: **edu.asu.bsse.asuriteid.appname** where **bsse** designates your academic program: **bscse**, **msse**, **bsse**, **bscs**, which stand for **BS Computer Systems Engineering**, **MS Software Engineering**, **BS Software Engineering**, or **BS Computer Science** respectively. **asuriteid** is your asurite id.

2. As part of all class header comments, that you create and turn-in this semester, include a copyright notice, such as **Copyright 2020 Your Name**.
3. As part of the class header comments, include a **right to use** statement. The examples presented in class use the Apache License Version 2, but you should put whatever rights you prefer. At the very least, you must provide the instructor and the University with the right to build and evaluate the software package for the purpose of determining your grade and program assessment.
4. As part of the class header comments, include a reference to the software's author, such as: **@author FirstName LastName mailto:FirstName.LastName@asu.edu**.
5. As part of the class header comments, include an indication of the software version, such as: **@version Oct 18, 2020**

These comments are required on all code (classes) that you generate this semester. If you hand in an assignment that does not include them, it will not be graded. Here are **StudentJson** classes that demonstrated **JSON** serialization in each language and platform **Android** and **iOS**. See studentJsonClasses.jar Download studentJsonClasses.jar . Download . Use java's **jar** command to extract this file. It will create a folder containing a **playground** that can be opened in **Xcode** and a **Java** class in the file **StudentJson.java**.

Lab Exercise 2
Life-cycles

Description

You are again to construct both an Android and iOS app. These apps should demonstrate lifecycles on these platforms. Here is a description of the two apps.

Android

In the Android app, create a two activity app. One activity is the **MainActivity** and the other is a **manually** created alert Activity (**AlertActivity**, as demonstrated in class, start the second activity upon a button click in the first view. Both activities should override the following activity lifecycle methods: **onRestart, onStart, onResume, onPause, onStop,** and **onDestroy**. The only actions in each of these methods are to call the parent's corresponding (overridden) method and to log the name of the class and the name of the method being called. That is, call **android.util.Log.d** passing the string name of the class (**this.getClass().getSimpleName()**) and the string name of the method.

Get each life-cycle method to be called. Using comments before each method, explain what you had to do to get the method to be called.

The **MainActivity** and its corresponding view (**layout/activity_main.xml** should include a **button** and a simple message. When the button is clicked, create a new **Intent** for the **AlertActivity** and start the activity associated with the intent. The **AlertActivity** should only take up a portion of the screen (about a third) and include a message, such as "Hello Android Developer", and a **Button** labeled **OK**. Clicking the **OK** button should cause the **AlertActivity** to **finish**.

iOS

In the iOS app, do not implement the same functionality as you did for Android. Instead, log all of the **AppDelegate** lifecycle methods. Use the **NSLog** or the **print** method. These provide the opportunity for the app developer to react to any state changes in the app from a single place in the app's code. All of these methods have empty implementations in the **Single View Application** template. You should provide a log that names the method that has been called. Explore with the App and the Simulator to discover how to get all of the six methods to be called and generate a log that displays in **Xcode**. Add the scenario for getting the method to be called in the source code comments above the method.

In this app, implement two simple **views** that demonstrate each of the optional lifecycle methods in **UIViewController**:

- **override func viewWillAppear(_ animated: Bool)**
- **override func viewDidAppear(_ animated: Bool)**

- **override func viewWillDisappear(_ animated: Bool)**
- **override func viewDidDisappear(_ animated: Bool)**

For both **Android** and **iOS**, implement these methods to create a log entry (**NSLog**, **print**, and/or **android.util.Log.d** method) that identifies the view controller and method that's been called. In comments before the method implementation, indicate what you had to do to get the system to call the method, thus generating a log entry.

Same as with the Android app, get each life-cycle (state transition) method to be called. Using comments before each method, explain what you had to do to get the method to be called. Augment the template comments in the app delegate methods with a description of what you had to do on the **Simulator/Xcode** to get the message to display in the **Xcode** log.

1. Place all of your application source code into the package named: **edu.asu.bsse.asuriteid.appname** where **bsse** designates your academic program: **bscse**, **msse**, **bsse**, **bscs**, which stand for **BS Computer Systems Engineering**, **MS Software Engineering**, **BS Software Engineering**, or **BS Computer Science** respectively. **asuriteid** is your asurite id.
2. As part of all class header comments, that you create and turn-in this semester, include a copyright notice, such as **Copyright 2020 Your Name**.
3. As part of the class header comments, include a **right to use** statement. The examples presented in class use the Apache License Version 2, but you should put whatever rights you prefer. At the very least, you must provide the instructor and the University with the right to build and evaluate the software package for the purpose of determining your grade and program assessment.
4. As part of the class header comments, include a reference to the software's author, such as: **@author FirstName LastName mailto:FirstName.LastName@asu.edu**.
5. As part of the class header comments, include an indication of the software version, such as: **@version Oct 25, 2020**

These comments are required on all code that you generate this semester. If you hand in an assignment that does not include them, it will not be graded. Here are the **StudentJson** classes that were constructed during the first class session as part of the example **Android** and **iOS** apps. See studentJsonClasses.jar Download studentJsonClasses.jar . Use java's **jar** command to extract this file. It will create a folder containing a **playground** that can be opened in **Xcode** and a **Java** class in the file **StudentJson.java**.

Lab Exercise 4
iOS App to Manage Place Descriptions

Description

You are to construct a multiple view iOS app for managing **PlaceDescriptions**. The app should be based on the Android app that you developed for Lab3 and your first lab submission in which you created a **Swift**. **PlaceDescription** class. This app is similar to that one but uses more complex controls and multiple views to display and manage a collection of **PlaceDescription** objects. Add a **PlaceLibrary** class which retains place objects. This app does not need to use the network or access a **Place** database, although a future revision of the app will develop these aspects. Your app should provide a list of all place names contained in the collection. It should allow the user to select one title and see all of the information about that title, and it should provide the ability to add or remove place description entries. New descriptions should be added to the collection by the user manually entering the appropriate information needed to create a **PlaceDescription** object. Include at least the same properties as were implemented in the first lab. Here Download Here is the instructors' JSON initialization file that may be useful in creating your solution.

Constraints

In the iOS app adhere to the following constraints:

- Your app should not be a **split-screen**, and it should use a **Navigation Controller** to control the navigation bar.
- Use a **UITableViewController** and your storyboard should include a **Navigation Controller**.
- Use at least the following basic controls: **Button**, **TextField**, and a **Picker**.
- Provides separate classes for **PlaceDescription** and **Place Library**

The same packaging, documentation, and solution formatting constraints as with the prior assignments apply to this assignment as well:

1. Place all of your application source code into the package named: **edu.asu.bsse.asuriteid.appname** where **bsse** designates your academic program: **bscse**, **msse**, **bsse**, **bscs**, which stand for **BS Computer Systems Engineering**, **MS Software Engineering**, **BS Software Engineering**, or **BS Computer Science** respectively. **asuriteid** is your asurite id.
2. As part of all class header comments, that you create and turn-in this semester, include a copyright notice, such as **Copyright 2020 Your Name**.
3. As part of the class header comments, include a **right to use** statement. The examples presented in class use the Apache License Version 2, but you should put whatever rights you prefer. At the very least, you must provide the instructor and the University with the right to build and evaluate the software package for the purpose of determining your grade and program assessment.

4. As part of the class header comments, include a reference to the software's author, such as:
   @author FirstName LastName mailto:FirstName.LastName@asu.edu.
5. As part of the class header comments, include an indication of the software version, such as:
   @version Nov 1, 2021

These comments are required on all code that you generate this semester. If you hand in an assignment that does not include them, it will not be graded.

Lab Exercise 6
JsonRPC Enabled iOS App

Description

Create an **iOS** app to browse and modify a collection of **Place Descriptions**, as you did in prior assignments. But, for this app, make network connections to a JsonRPC server to provide the model for your app, instead of storing the model on the device itself. Your app should provide the ability to browse, add, and remove places. Any changes to places or the library that are made by the app user should be pushed to the server using the JsonRPC methods defined by the server.

The **iOS** app you create to complete this problem should be compatible with the Android App from the last assignment. That is, your iOS app running on the simulator and your Android app running on the emulator should be able to access and jointly utilize the JsonRPC place server. A change made by one app should become visible to the other (the next time that app references the server -- no push to clients/mobile apps is required). Both of your apps should use the server as its only underlying model for obtaining place information. You should not retain a complete local mirror of the collection within the app, and instead, use the **JsonRPC** methods defined by the collection to obtain model information and only store minimal information within the app.

Your app should not perform network connections on the UI or Main Thread but instead should perform these operations asynchronously. Your app should also not access view objects except on the UI or Main Thread. You are free to use either the **iOS**

AsyncTask (asyncTaskiOS.playground.jar Download asyncTaskiOS.playground.jar)
discussed in class or the approach used in the **StudentCollectioniOS** sample app which is linked below. Both approaches when properly applied will adhere to these constraints.
You may use the instructor provided **JsonRPC** server for places to implement your iOS

app: placeServerWeb.jar Download placeServerWeb.jar or you may build your own server based on the student collection example discussed in class. Here are links to examples that may be useful in constructing your solution to this lab exercise:

- studentCollectionJsonRPCServer.jar Download studentCollectionJsonRPCServer.jar JsonRPC student collection server.
- studentCollectionPlayground.jar Download studentCollectionPlayground.jar Swift JsonRPC student collection stub.
- studentCollectioniOSJsonRPC.jar Download studentCollectioniOSJsonRPC.jar Student collection JsonRPC client.

Similar to the last lab exercise, store and access the **URL** of the server following good practice (and ease of grading). In a future unit, we'll discuss app settings, but in this exercise, your **iOS** app should define a key, value pair in either the **info.plist** file, or in a separate property list file that you add to your project. The property should provide the server URL. To run with the server on the same machine, the address should be **localhost**. So, your default URL for the iOS app may be:

http://localhost:8080/

The simulator runs as a normal process on OSX. Access the value of this property from the source code with code, assuming the key is **ServerURLString**, as follows:

```
if let infoPlist = NSBundle.mainBundle().infoDictionary {
   self.urlString = ((infoPlist["ServerURLString"]) as?  String!)!
   NSLog("The default urlString from info.plist is \(self.urlString)")
}else{
   NSLog("error getting urlString from info.plist")
}
```

What To Hand-In

The same packaging, documentation and solution formatting constraints as with the prior assignments apply to this assignment as well.

1. Place all of your application source code into the package named: **edu.asu.bsse.asuriteid.appname**
   where **bsse** designates your academic program: **bscse**, **msse**, **bsse**, **bscs**, which stand for **BS Computer Systems Engineering**, **MS Software Engineering**, **BS Software Engineering**, or **BS Computer Science** respectively.
   asuriteid is your asurite id.
2. As part of all class header comments, that you create and turn-in this semester, include a copyright notice, such as **Copyright 2020 Your Name**.
3. As part of the class header comments, include a **right to use** statement. The examples presented in class use the Apache License Version 2, but you should put whatever rights you prefer. At the very least, you must provide the instructor and the University with the right to build and evaluate the software package for the purpose of determining your grade and program assessment.
4. As part of the class header comments, include a reference to the software's author, such as:
   @author FirstName LastName mailto:FirstName.LastName@asu.edu.
5. As part of the class header comments, include an indication of the software version, such as:
   @version NOV 8, 2020

These comments are required on all code that you generate this semester. If you hand in an assignment that does not include them, it will not be graded.

Submission

Structure your project as sub-directories of the folder named **Assign6_7MyASURITEID**. Two sub-directories should separately contain each of the following:

- **The Android app** (Lab Exercise 7)
- **The iOS app** (Lab Exercise 6)

You will submit this project, by first cleaning each of its component apps, to remove all generated files. Then create a **jar** or **zip** archive of the project (Assign6*) directory. You can create a **Java Archive (jar)** by executing the following command from a terminal in the directory which is parent to the project directory:

jar -cvf Assign6_7MyASURITEID.jar Assign6_7MyASURITEID/