Mobile Application Development
Week 15: Google Maps

Android and Google Maps

Google Maps can be used on any platform through their API.
The Maps SDK for Android offers four types of maps, as well as an option to have no map at all:

- Normal
    - Typical road map. Shows roads, some features built by humans, and important natural features like rivers. Road and feature labels are also visible.
- Hybrid
    - Satellite photograph data with road maps added. Road and feature labels are also visible.
- Satellite
    - Satellite photograph data. Road and feature labels are not visible.
- Terrain
    - Topographic data. The map includes colors, contour lines and labels, and perspective shading. Some roads and labels are also visible.
- None
    - No tiles. The map will be rendered as an empty grid with no tiles loaded.

You can also add markers to a map. Markers indicate a single location on the map.
Create a marker using GoogleMap.addMarker(markerOptions).
https://developers.google.com/maps/documentation/android-sdk/marker
MarkerOptions include:

- Position is a LatLng() object
- Title is a String

Markers can also be customized and draggable.
Maps Android API Getting Started https://developers.google.com/maps/documentation/android-sdk/start

Google Play Services

https://developers.google.com/android/guides/overview
Google Play Services is a client library that contains the interfaces to the individual Google services and allows you to obtain authorization from users to gain access to these services with their credentials. It also contains APIs that allow you to resolve any issues at runtime, such as a missing, disabled, or out-of-date Google Play services APK.
Google Play Services are distributed as an APK (Android Package Kit) from the Google Play Store. This makes it independent of the carrier, OS, or OEM so users don't have to wait for them to send out an update for users to upgrade Google Play Services with improvements and bug fixes on their devices.
You must have these installed on your device or emulator to run an app using Google Maps.

Map
Create a new Android Studio project
Phone and Tablet: Google Maps Activity
Name: Map
Language: Kotlin
Minimum SDK: API 21: Android 5.0 Lollipop (or the level of your device if it's lower)
Make sure to choose the API version of your device or lower (Settings | More | About)
Leave legacy libraries unchecked (using these restricts usage of some newer capabilities)
Finish

Google Play Services

Tools | SDK manager
SDK Tools tab
Check Google Play Services to install

Files
activity_maps.xml is the layout for the map. Notice it contains only a fragment element and that has the id "map".
Fragments allow you to break your activities up into smaller modular components which can easily be reused and adapted for different device sizes, orientation, or other criteria.
You can have one or more fragments embedded in an activity.
For this app we're just going to leave the one fragment created for us.

MapsActivity.kt
Notice all the imports at the top in order to use maps.
When you go into MapsActivity.kt you'll see that view binding is used and in onCreate its used to inflate the layout. We'll leave this as it's already set up.
GoogleMap is the main class of the Maps SDK for Android.
A GoogleMap called mMap is defined and uses the keyword lateinit.
https://developers.google.com/android/reference/com/google/android/gms/maps/GoogleMap

The class also implements the OnMapReadyCallback interface which is the callback interface for when the map is ready to be used.

```
class MapsActivity : AppCompatActivity(), OnMapReadyCallback
```

When this interface is implemented it's required that the onMapReady(GoogleMap) method be impelemented.
onMapReady(GoogleMap) is called when a map is ready to be used.

- A GoogleMap instance associated with the MapFragment is automatically sent to onMapReady() and then assigned to mMap.
- A LatLng object is defined with the coordinates for Sydney, Australia https://developers.google.com/android/reference/com/google/android/gms/maps/model/LatLng
- A marker is added to those coordinates.
- The map view is modeled as a camera looking down on a flat plane. The position of the camera (and hence the rendering of the map) is specified by the following properties https://developers.google.com/maps/documentation/android-api/views#the_camera_position
  - target (latitude/longitude location)
  - bearing (orientation)
  - tilt (viewing angle)
  - zoom
- moveCamera() moves the map to be centered at the location coordinates.

In onCreate() we get access to the map fragment through the supportFragmentManager object using the map id through view binding.
The GoogleMap object mMap provides access to the map data and view.
The getMapAsync() method sets a callback object which will be triggered when the GoogleMap instance is ready to be used.

- If google play services is not installed the callback won't be triggered
- The parameter is "this" because it's the current class that implements the OnMapReadyCallback interface and the onMapReady() method

res/values/google_maps_api.xml contains instructions on getting a Google Maps API key before you try to run the application.

Google Maps API key
In order to use Google Maps from any form of application (not just Android ones), you need to source and use a Google API key and configure your Google account for the Maps API.

Copy and paste the link in the google_maps_api.xml file into a browser.

https://console.developers.google.com/flows/enableapi?apiid=maps_android_backend&keyType=CLIENT_SIDE_ANDROID&r=07:8D:07:3C:C1:53:34:E7:F6:11:2F:96:91:AB:6A:72:CC:94:5E:3D%3Bcom.example.aileen.map
Create a project (takes a few minutes)
Create API key
Copy your API key
https://console.developers.google.com/apis/credentials?project=driven-photon-150723&authuser=1
In the google_maps_api.xml file replace "YOUR_KEY_HERE" with your API key (no quotes)

The AndroidManifest.xml file has entries that we haven't seen before.

```
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Allows the API to use available location providers including cellular (tower) data, WiFi signals, and GPS data to determine as precise a location as possible.

You'll notice that you have an error saying that coarse location is required as well. This won't give you an error until we get to location access but if you want to get rid of it go ahead and add coarse location access as well.

```
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

The `"android.permission.ACCESS_COARSE_LOCATION"` permission allows the API to use WiFi or cell data to determine the devices location but does not use GPS. It returns a location with an accuracy approximately equivalent to a city block.

Android 10 (API level 29) or higher includes the `ACCESS_BACKGROUND_LOCATION` permission if you need to access the device location while your app is in the background, although this is not recommended. On Android 8.0 (API level 26) and higher, if an app is running in the background when it requests the current location, the device calculates the location only a few times each hour.

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="@string/google_maps_key" />
```

Gives your app access to the Google Maps API using your API key

The template also adds a Google Play Services dependency to build.gradle(Module: app). This dependency exposes the Google Maps and Location Services APIs to the application.

```
implementation 'com.google.android.gms:play-services-maps:17.0.0'
```

Run the app on your device. You should see the marker on Sydney, Australia.

(update Google Play services on your device if needed)

To run the app in the emulator Google Play Services must be installed. This can be a lengthy process but it ran on the Pixel 3 emulator with no additional steps needed (did not seem to work on a Pixel 5).

Troubleshooting:

You might get this error on older phones:

Error:Execution failed for task ':app:transformClassesWithDexForDebug'.

On your device go into Settings | Apps | Google Play Services OR

Settings | More | Application Manager | All (Note: You might need to scroll to the right to see this) | Google Play Services.

Update if available. Make sure it's enabled. Get the version number.

In your app go into Gradle Scripts and open build.gradle(Module: app) and under dependencies change this line to the version of Google Play Services on your device (only use 1 significant digit, so not .01):

```
implementation 'com.google.android.gms:play-services-maps:17.0.0'
(ex: change to implementation 'com.google.android.gms:play-services-
maps:8.3.0')
```

Sync after the change and try to run it on your device.