LabW09 – Location Access

Objectives:

1. Understand how to use Google Maps API
2. Understand how to use Google Location Services API

Tasks:

1. Use Google Maps API
2. Access to current location

Location Services allows apps and websites (including Maps, Camera, Weather, and other apps) to use information from cellular, Wi-Fi, Global Positioning System (GPS) networks, and Bluetooth to determine your approximate location.

Using location-based information in your app is a great way to keep the user connected to the surrounding world. Whether you use this information for practical purposes such as navigation or for entertainment, location-based information can offer users a more contextual experience and enhance the overall user experience.

In this tutorial, we will explore how to capture this location information by using Google Maps API and Google Location Services API in your mobile device.

Task 1: Use Google Maps API

This guide is a quick start to adding a map to an Android app. Android Studio is the recommended development environment for building an app with the Maps SDK for Android.

With the Maps SDK for Android, you can add maps based on Google Maps data to your application. The API automatically handles access to Google Maps servers, data downloading, map display, and response to map gestures. You can also use API calls to add markers, polygons, and overlays to a basic map, and to change the user's view of a particular map area. These objects provide additional information for map locations, and allow user interaction with the map. The API allows you to add these graphics to a map:

- Icons anchored to specific positions on the map (Markers).
- Sets of line segments (Polylines).
- Enclosed segments (Polygons).
- Bitmap graphics anchored to specific positions on the map (Ground Overlays).
- Sets of images which are displayed on top of the base map tiles (Tile Overlays).

This task is based on the following tutorial links:
https://developers.google.com/maps/documentation/android-sdk/start
https://developers.google.com/maps/documentation/android-sdk/map-with-marker

1. Start Android Studio and pick a “Google Maps Activity” project under “Phone and Tablet” tab for choosing your project. Enter the following information as prompted:

    - Name: “GoogleMaps”
    - Package name: “comp5216.sydney.edu.au.googlemaps”
    - Choose your preferred Save location
    - Language: Java
    - Minimum API level: API 23 Android 6.0 (Marshmallow)
    - Click Finish

2. Your app needs a Google Map API key in order to access the Google Maps servers, so in this step, you should obtain a Google Maps API key so that you can add the API key to your app. Use the link provided in the google_maps_api.xml file that Android Studio created for you by copying the link provided in the google_maps_api.xml file and paste it into your browser e.g.

```
https://console.developers.google.com/flows/enableapi?apiid=maps_android_backend&keyType=CLIENT_SIDE_ANDROID&r=62:7C:68:A8:A1:17:C2:91:7C:3C:00:BF:42:9F:FE:1E:EB:EC:85:65%3Bcomp5216.sydney.edu.au.googlemaps
```

3. Follow the instructions to create a new project on the Google Cloud Platform Console or select an existing project.

4. Create an Android-restricted API key for your project.

5. Copy the resulting API key, go back to Android Studio, and paste the API key into the <string> element in the google_maps_api.xml file.

```
<string name="google_maps_key" templateMergeStrategy="preserve"
translatable="false">YOUR_KEY_HERE</string>
```

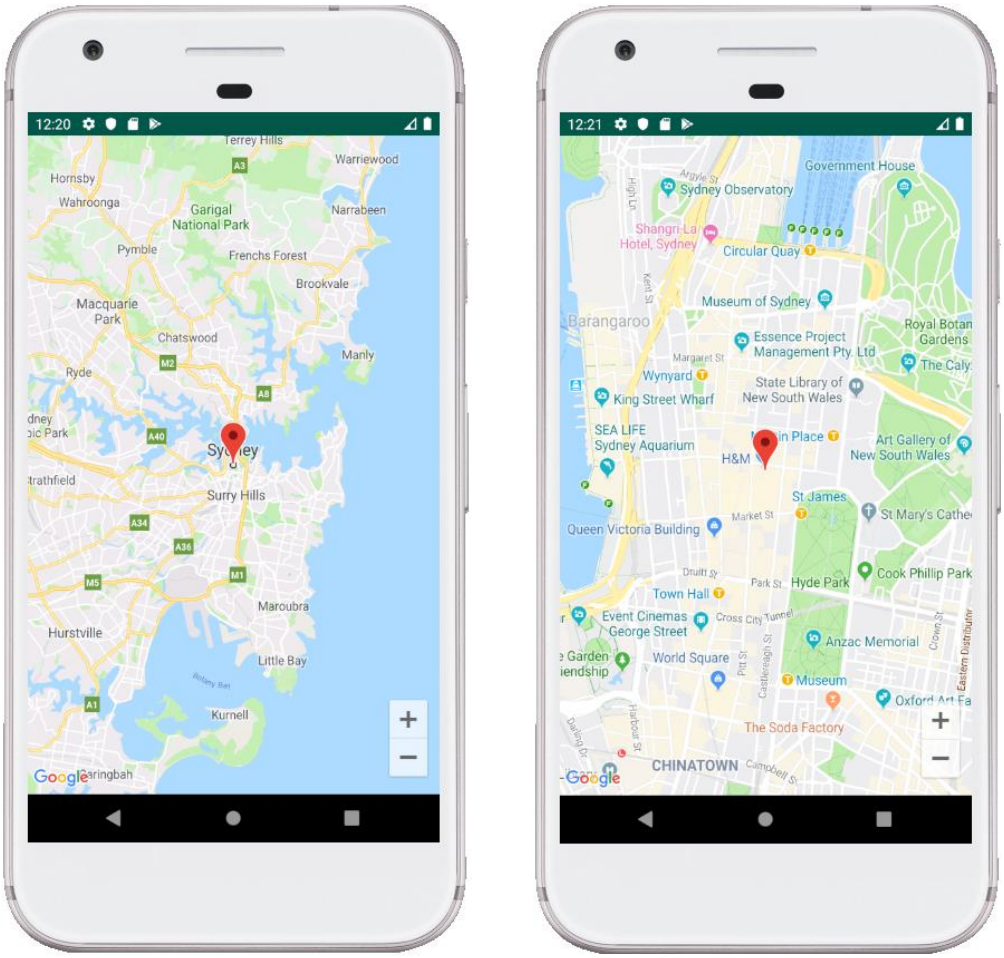
6. Review the code generated by Android Studio. Have a look at the app's layout XML file activity_maps.xml, and the Java file that defines the maps activity MapsActivity.java.

7. Enable zoom controls on your map and animate the camera to zoom into the map as soon as the map is ready to be used. Refer to the code snippet below.

The final code should look the same as per MapsActivity.java file given on the lab files for task 1.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_maps);
    setUpMapIfNeeded();
}

private void setUpMapIfNeeded() {
    // Do a null check to confirm that we have not already instantiated the map.
    if (mMap == null) {
        // Obtain the SupportMapFragment and get notified when the map is ready to be
used.
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }
}

@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;

    if (mMap != null) {
        // Map is ready
        Toast.makeText(this, "Map is ready to be used!!", Toast.LENGTH_SHORT).show();

        // Add a marker in Sydney and move the camera
        LatLng sydney = new LatLng(-33.8692, 151.2089);
        mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));

        // Enable zoom controls
        mMap.getUiSettings().setZoomControlsEnabled(true);
        // Animate the camera to zoom into the map
        mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(sydney, 11));
    } else {
        Toast.makeText(this, "Error - Map was null!!", Toast.LENGTH_SHORT).show();
    }
}
```

8. Build and run your app.
   A Google map, a marker anchored to a specific Sydney location and zoom controls should be displayed as per the following screenshots. Use the zoom controls on the map to zoom in or out of your map.

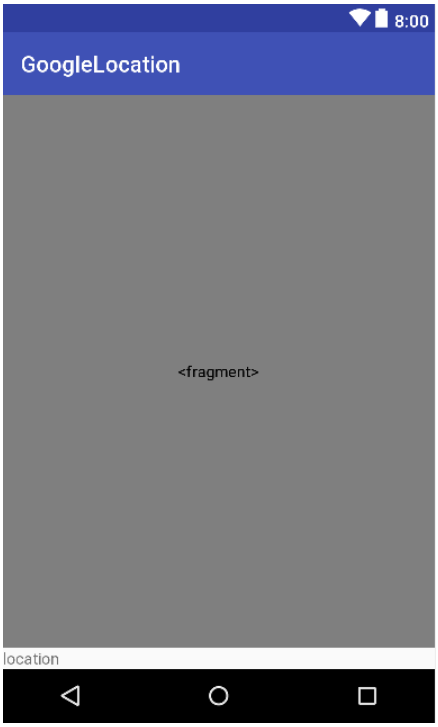


Task 2: Access to current location

This task shows you how to find the current location of an Android device. It builds an Android app using the Maps SDK for Android and the fused location provider in the Google Play services location APIs. The content is partially adopted from the following tutorial https://developers.google.com/maps/documentation/android-sdk/current-place-tutorial

1. Create a new “Google Maps Activity” project. Enter the following information as prompted:
    - Name: “GoogleLocation”
    - Package name: “comp5216.sydney.edu.au.googlelocation”
    - Choose your preferred Save location
    - Language: Java
    - Minimum API level: API 23 Android 6.0 (Marshmallow)
    - Click Finish

2. Copy the ***activity_maps.xml*** (in lab files) into your project's res/layout folder. It only contains a location TextView at the bottom to show the latitude and longitude, and a Google Map Fragment above it to cover the rest of the screen.



3. Get a Google Maps API key and add the API key to your app. Refer to step 2 to 5 from Task 1 on how to obtain a key and use it for your app.

4. In your MapsActivity.java file, declare variables to reference the location ***TextView,*** the current location (***mLastKnownLocation***), default location (***mDefaultLocation***), default zoom and other supporting variables.

```
private static final String TAG = MapsActivity.class.getSimpleName();
private GoogleMap mMap;
private CameraPosition mCameraPosition;
private TextView locationTextView;

// The entry point to the Fused Location Provider.
private FusedLocationProviderClient mFusedLocationProviderClient;

// A default location (Sydney, Australia) and default zoom to use when location
permission is not granted.
private final LatLng mDefaultLocation = new LatLng(-33.8523341, 151.2106085);
private static final int DEFAULT_ZOOM = 15;
private static final int PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION = 1;
private boolean mLocationPermissionGranted;

// The geographical location where the device is currently located. That is, the last-
known location retrieved by the Fused Location Provider.
private Location mLastKnownLocation = new Location("");

// Keys for storing activity state.
private static final String KEY_CAMERA_POSITION = "camera_position";
private static final String KEY_LOCATION = "location";
```

5. Add Google Play Services libraries in build.gradle file inside your app module directory

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

6. In ***onCreate***() method, use the layout activity_maps, and initialise the Fused Location Provider Client. Also initialise the Google Map.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Retrieve location and camera position from saved instance state.
    if (savedInstanceState != null) {
        mLastKnownLocation = savedInstanceState.getParcelable(KEY_LOCATION);
        mCameraPosition = savedInstanceState.getParcelable(KEY_CAMERA_POSITION);
    }

    // Retrieve the content view that renders the map.
    setContentView(R.layout.activity_maps);

    // Obtain the location TextView
    locationTextView = (TextView) this.findViewById(R.id.location);

    // Construct a FusedLocationProviderClient.
    mFusedLocationProviderClient =
LocationServices.getFusedLocationProviderClient(this);

    // Obtain the SupportMapFragment and get notified when the map is ready to be used.
    SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
            .findFragmentById(R.id.map);
    mapFragment.getMapAsync(this);
}
```

7. Make sure your activity extends AppCompatActivity instead of Activity, and implements OnMapReadyCallback. We will add the required methods in the next step.

```
public class MapsActivity extends AppCompatActivity implements OnMapReadyCallback {
    ...
}
```

8. Add the required callback methods.

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;

    // Prompt the user for permission.
    getLocationPermission();

    // Turn on the My Location layer and the related control on the map.
    updateLocationUI();

    // Get the current location of the device and set the position of the map.
    getDeviceLocation();
}
```

9. Make sure getLocationPermission, updateLocationUI, and getDeviceLocation methods are implemented and you understand what each function does. Refer to MapsActivity.java in the lab files.

```
private void getDeviceLocation() {
    try {
        if (mLocationPermissionGranted) {
            Task<Location> locationResult =
mFusedLocationProviderClient.getLastLocation();
            locationResult.addOnCompleteListener(this, new
OnCompleteListener<Location>() {
                @Override
                public void onComplete(@NonNull Task<Location> task) {
                    if (task.isSuccessful()) {
                        // Obtain the current location of the device
                        mLastKnownLocation = task.getResult();
                        String currentOrDefault = "Current";

                        if (mLastKnownLocation != null) {
                            mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(
                                    new LatLng(mLastKnownLocation.getLatitude(),
                                            mLastKnownLocation.getLongitude()),
DEFAULT_ZOOM));
                        } else {
                            Log.d(TAG, "Current location is null. Using defaults.");
                            currentOrDefault = "Default";

mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(mDefaultLocation, DEFAULT_ZOOM));
                            mMap.getUiSettings().setMyLocationButtonEnabled(false);

                            // Set current location to the default location
                            mLastKnownLocation = new Location("");
                            mLastKnownLocation.setLatitude(mDefaultLocation.latitude);

mLastKnownLocation.setLongitude(mDefaultLocation.longitude);
                        }

                        // Show location details on the location TextView
                        String msg = currentOrDefault + " Location: " +
                                Double.toString(mLastKnownLocation.getLatitude()) + ",
" +
                                Double.toString(mLastKnownLocation.getLongitude());
                        locationTextView.setText(msg);

                        // Add a marker for my current location on the map
                        MarkerOptions marker = new MarkerOptions().position(
                                new LatLng(mLastKnownLocation.getLatitude(),
mLastKnownLocation.getLongitude()))
                                .title("I am here");
                        mMap.addMarker(marker);
                    } else {
                        Log.d(TAG, "Current location is null. Using defaults.");
                        Log.e(TAG, "Exception: %s", task.getException());
                        mMap.moveCamera(CameraUpdateFactory
                                .newLatLngZoom(mDefaultLocation, DEFAULT_ZOOM));
                        mMap.getUiSettings().setMyLocationButtonEnabled(false);
                    }
                }
            });
        }
    } catch (SecurityException e)  {
        Log.e("Exception: %s", e.getMessage());
    }
}
```

10. Setup the permission for accessing location. You may refer to ***AndroidManifest.xml*** file in the lab material to update your own version.

    The ***ACCESS_COARSE/FINE_LOCATION*** permissions are not required to use Google Maps Android API v2, but are recommended.

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

11. Build and run the app, preferably on a mobile phone.
    The app should display your current location. The final code will be the similar to that of ***MapsActivity.java*** (in lab files).

To learn more about the Location Services API, see Google Location Services for Android:

- Build location-aware apps - https://developer.android.com/training/location/
- Get the last known location - https://developer.android.com/training/location/retrieve-current

To learn more about the step by step process of getting an embedded Google Map working within an Android emulator, refer to this tutorial:
https://github.com/codepath/android_guides/wiki/Google-Maps-Fragment-Guide