Mobile Application Development

Produced by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Android Google Services Part 3

Google Maps

Google Services Overview

- Overview of Google Play Services and Setup
- Detailed look at
  - Google Sign-in and Authentication (Part 1)
  - Location & Geocoding (Part 2)
  - Google Maps (Part 3)

Google Services Overview

- Detailed look at
  - Google Maps (Part 3)

Agenda

- Overview
- Installation & Registration of the Google Maps API ‘Key’
- Creating interactive Maps with GoogleMaps, (Support)MapFragments & FragmentActivitiys
- Creating & Adding Markers to Maps
- Custom Styling our Maps (Video)

Overview – What is it?

- “A mapping and navigation application for desktop and mobile devices from Google. Maps provides turn-by-turn directions to a destination along with 2D and 3D satellite views, as well as public transit information. Maps also offers photographic views of the turns, which show the real streets and surroundings (Google "street views").” – www.pcmag.com

- **Google Maps APIs** are available for Android, iOS, web browsers and via HTTP web services.

Overview – Accessing Google Maps

- Google maps can be accessed in two ways
  - Through a browser or a `WebView`
  - Through the **Google Maps Android API v2**
- Google Maps Android API v2
  - allows you to incorporate Google Maps into applications
  - is distributed as part of the Google Play Services SDK
  - encapsulates maps in a `MapFragment` or a `SupportMapFragment`

> `MapFragment` and `SupportMapFragment` essentially replace the `MapActivity` class used in version 1.

Overview – Usage *

- Using **Google Maps Android API v2**, you can
  - Add maps to your app
    - 3D maps, Terrain maps, Satellite maps.
  - Customize the map
    - Markers, Image Overlays, Polylines/Polygons
  - Control the user's view
    - Zoom, Pan, Rotate
  - Apply Custom Styles
    - Day/Night view, Custom Colours, Look & Feel.

Aside - Attribution Requirements

If you use the Google Maps Android API in your application, you must include the Google Play Services attribution text as part of a “Legal Notices” section in your application. Including legal notices as an independent menu item, or as part of an “About” menu item, is recommended.

The attribution text is available by making a call to method

```
GooglePlayServicesUtil.getOpenSourceSoftwareLicenseInfo()
```

and you should probably use a WebView not a TextView!

Part 3
Google Maps Android API

Introduction

- With the **Google Maps Android API**, 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.

Introduction

- 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).

Google Maps API Requirements

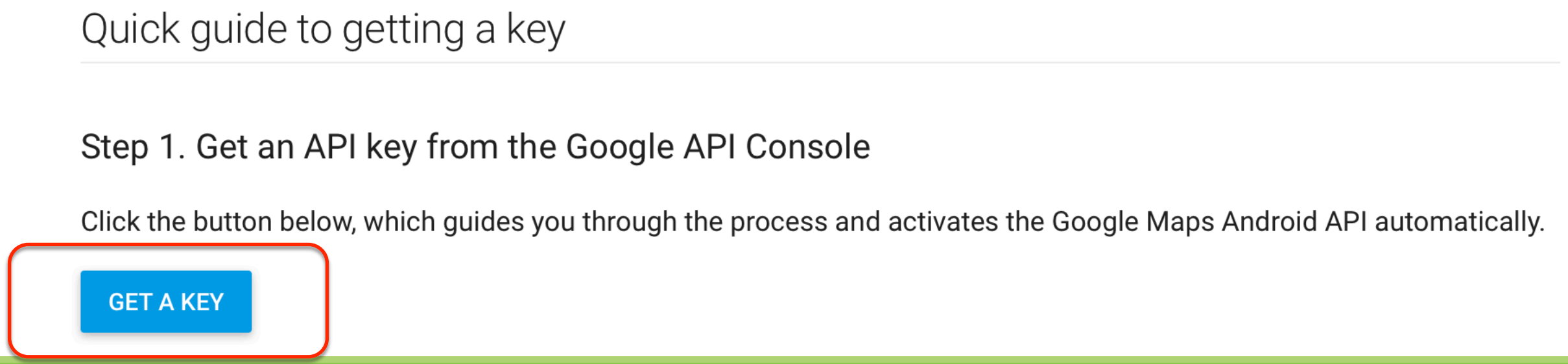
- For integrating Google Maps into your Android Application, you need to complete the following :

1. Enable Google Maps API V2 on The Developers Console and create credentials for your application authentication
2. Configuring Google Play Services in Android Studio
3. Create your Android Application with Google Maps integration

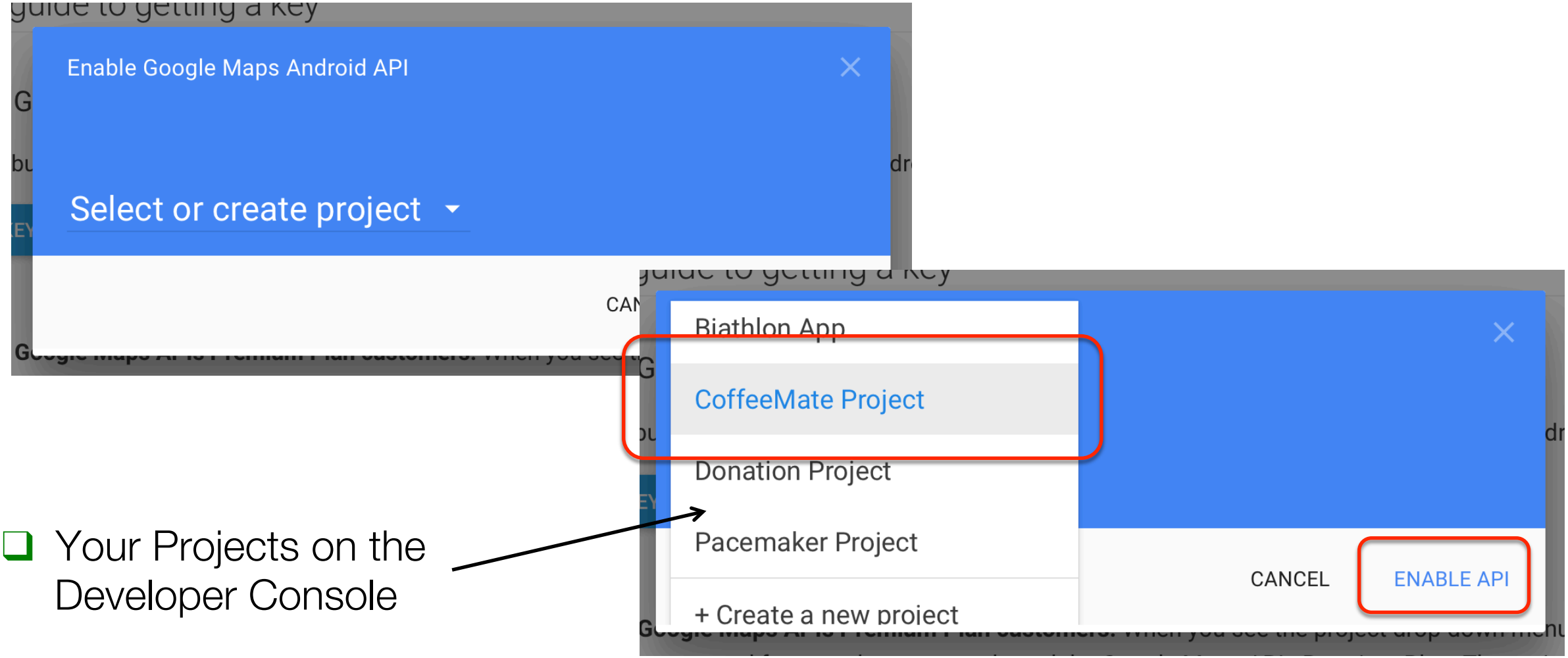
1. Enable Google Maps API V2 *

① You can take the same approach as we did for enabling the Google Sign-In OR you can let Google guide you through the process (as follows)

Visit Get API Key and follow the instructions



1. Enable Google Maps API V2 *

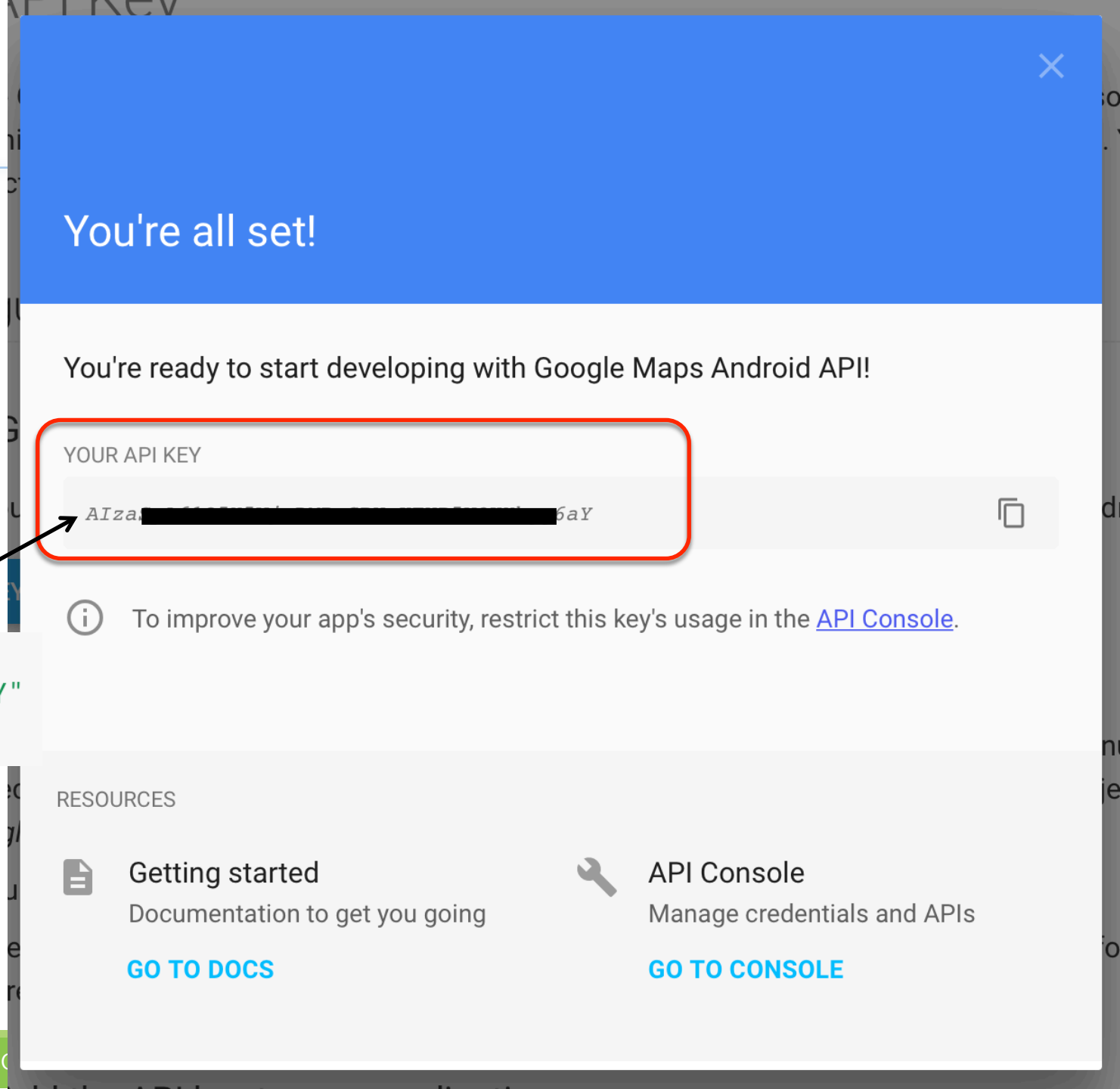


- Your Projects on the Developer Console

1. Enable Google Maps API V2 *

- In AndroidManifest.xml, add the following by inserting it just before the closing `</application>` tag:

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



2. Configure Google Play Services

- Already Done! (should be, from previous slides…)

3. Create your Android App (CoffeeMate)

- You'll cover this in the Labs, but we'll have a look at some of the setup and code next

Integrating Google Maps into Your Android App

https://developers.google.com/maps/documentation/android-api/

https://code.tutsplus.com/series/getting-started-with-google-maps-for-android--cms-891

1. Setting Up Your Android Project *

- First thing to do (after you've got your API Key) is to open your build.gradle file and confirm/import the Play Services library for maps and the locations Play Services library in order to set an initial position for your map. Place the following lines into the dependencies node of the build.gradle file. (version numbers may differ, currently 11)

```
1  compile 'com.google.android.gms:play-services-maps:7.8.0'
2  compile 'com.google.android.gms:play-services-location:7.8.0'
```

1. Setting Up Your Android Project *

- Next, open your AndroidManifest.xml file. ***Above*** the `<application>` node, you need to declare that the application uses OpenGL ES 2.0 and define the permissions needed by your application.

```
1  <uses-feature
2      android:glEsVersion="0x00020000"
3      android:required="true"/>
4
5  <uses-permission android:name="android.permission.INTERNET" />
6  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

1. Setting Up Your Android Project *

- Then, ***within*** the `<application>` node, add two pieces of metadata. The first informs the application that Play Services are used and the second binds the Maps API key with your application (`@string/google_api_key`).

```
1  <meta-data
2      android:name="com.google.android.gms.version"
3      android:value="@integer/google_play_services_version" />
4
5  <meta-data
6      android:name="com.google.android.geo.API_KEY"
7      android:value="@string/google_api_key"/>
```

1. Setup - Creating your Map Class

- You'll need to create a new class (your class MapFragment), which extends `SupportMapFragment`
  - used here rather than `com.google.android.gms.maps.MapFragment` for backwards compatibility before API 12.

And implement the following interfaces (next slide for expl.)

```
public class MapFragment extends SupportMapFragment implements
        GoogleApiClient.ConnectionCallbacks,
        GoogleApiClient.OnConnectionFailedListener,
        GoogleMap.OnInfoWindowClickListener,
        GoogleMap.OnMapLongClickListener,
        GoogleMap.OnMapClickListener,
        GoogleMap.OnMarkerClickListener {
```

1. Setup - The Necessary interfaces

- `ConnectionCallbacks` and `OnConnectionFailedListener` are designed to monitor the state of the `GoogleApiClient`, which is used in this application for getting the user's current location.
- `OnInfoWindowClickListener` is triggered when the user clicks on the info window that pops up over a marker on the map.
- `OnMapLongClickListener` and `OnMapClickListener` are triggered when the user either taps or holds down on a portion of the map.
- `OnMarkerClickListener` is called when the user clicks on a marker on the map, which typically also displays the info window for that marker.

1. Setup - Updating the Layout

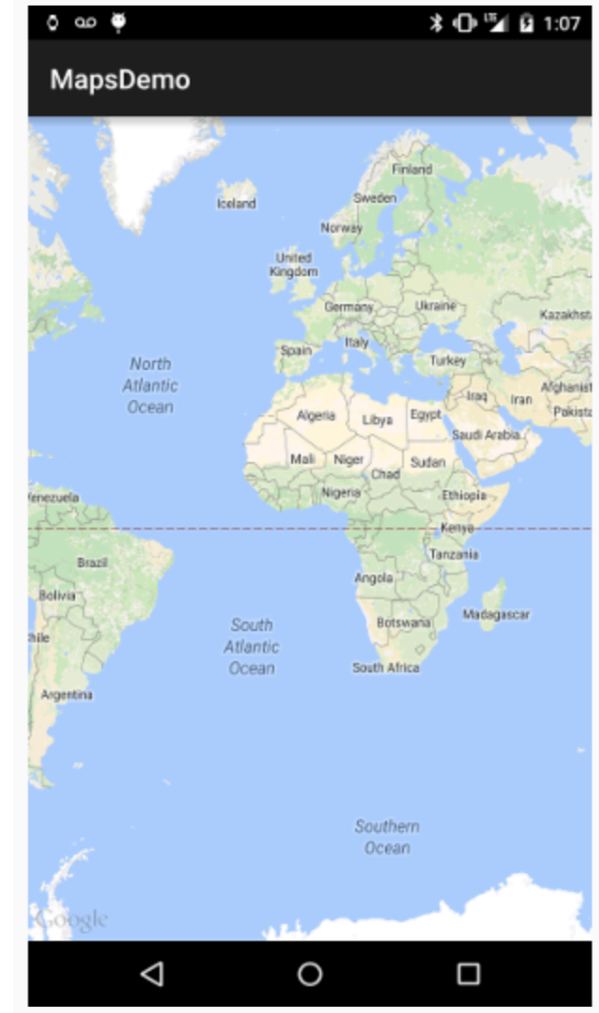
- Once you have the initial fragment built, you need to let your **MainActivity** (or wherever you plan on displaying the map) know that it should use this fragment. Open your **xml layout** from your resources folder and change it so that it includes the fragment as a view.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <fragment
        android:id="@+id/map"
        android:name="com.tutsplus.mapsdemo.MapFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</RelativeLayout>
```

- You'll use your own class reference

1. Test the Setup

- After updating your activity layout, you should be able to run your application and view a map of Earth that is fully zoomed out and focused on latitude 0, longitude 0.



2. Initializing the Map - Declaring Map Types *

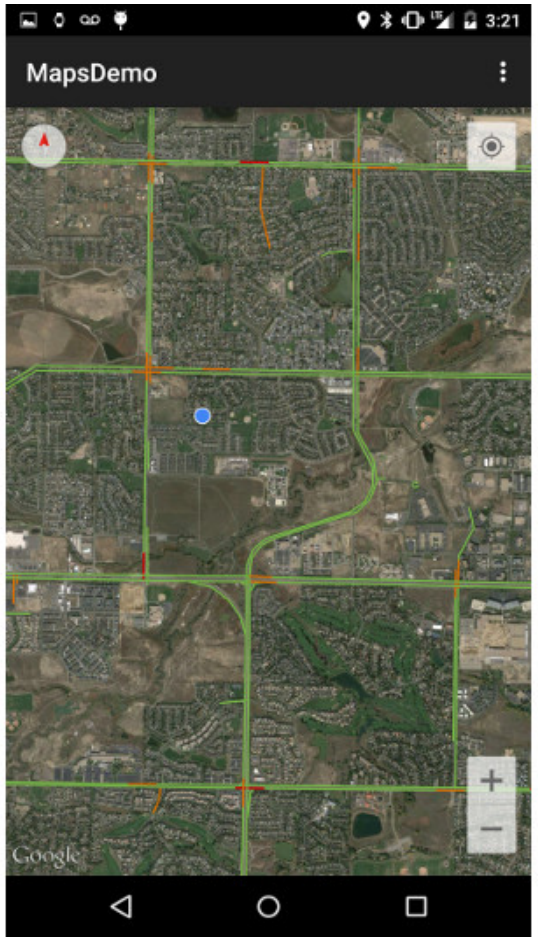
- Returning to our `MapFragment` class, you need to define some global values at the top of the class for use in your application.

```
private GoogleApiClient mGoogleApiClient;
private Location mCurrentLocation;

private final int[] MAP_TYPES = { GoogleMap.MAP_TYPE_SATELLITE,
                                  GoogleMap.MAP_TYPE_NORMAL,
                                  GoogleMap.MAP_TYPE_HYBRID,
                                  GoogleMap.MAP_TYPE_TERRAIN,
                                  GoogleMap.MAP_TYPE_NONE };

private int curMapTypeIndex = 0;
```

- Each of the map types serves a different purpose, so one or more may be suitable for your own applications.

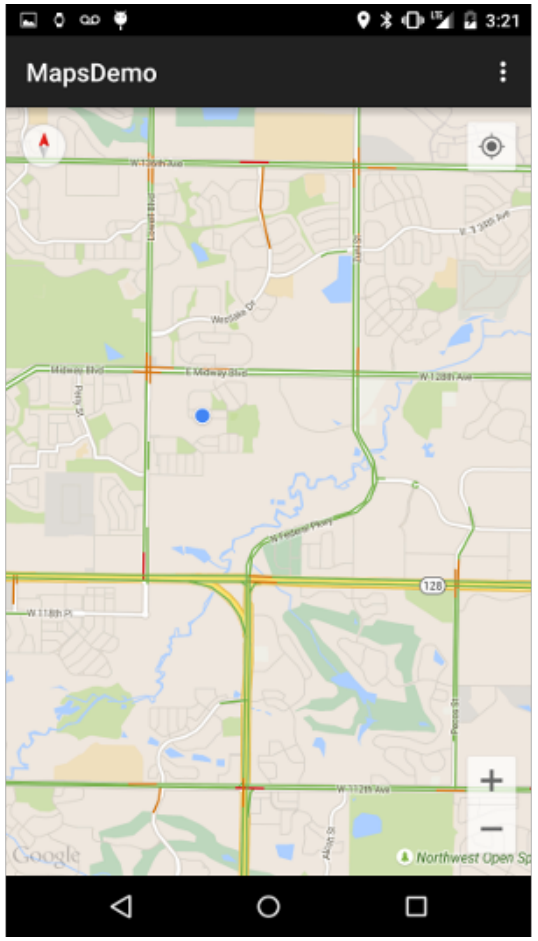
2. Initializing the Map - **Declaring Map Types** *

- `GoogleMap.MAP_TYPE_SATELLITE` displays a satellite view of the area without street names or labels.



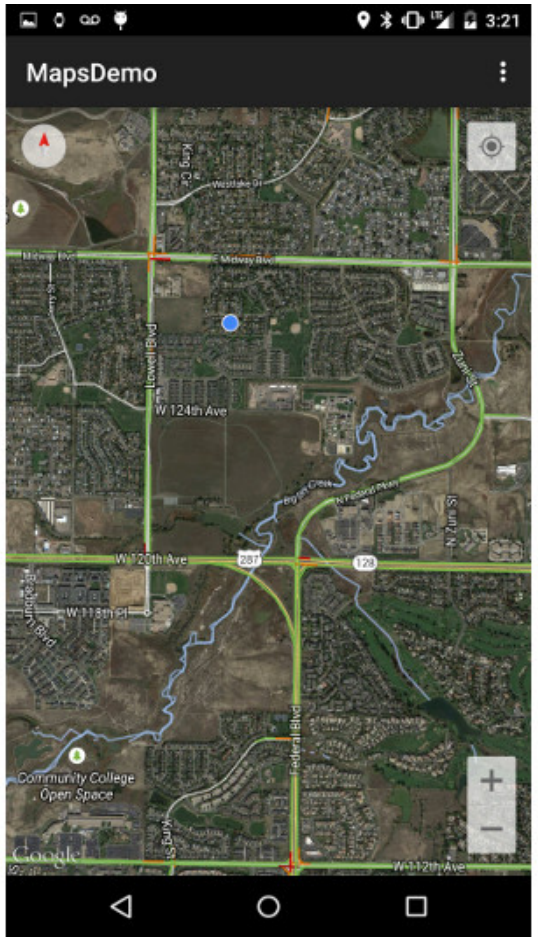
2. Initializing the Map - **Declaring Map Types** *

- `GoogleMap.MAP_TYPE_Normal` shows a generic map with street names and labels.



2. Initializing the Map - **Declaring Map Types** *

- `GoogleMap.MAP_TYPE_HYBRID` combines satellite and normal mode, displaying satellite images of an area with all labels.

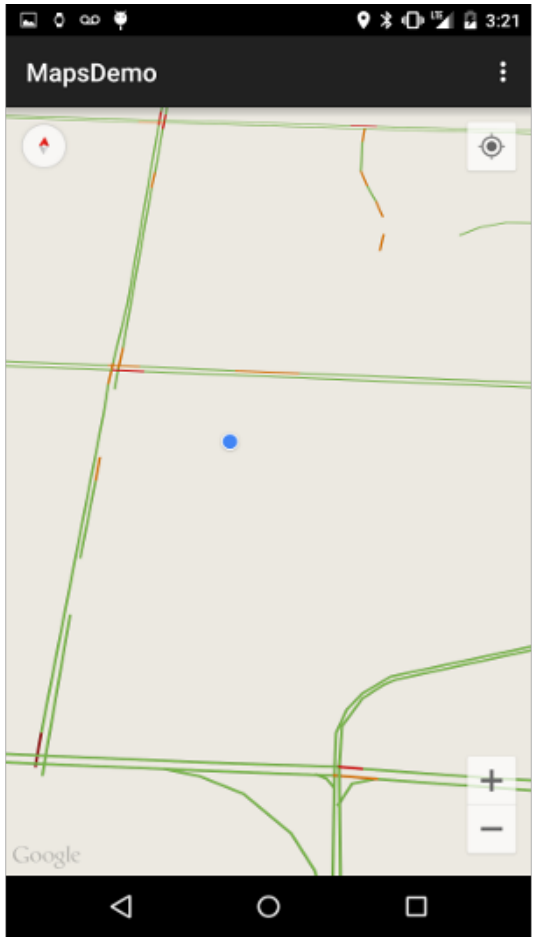


2. Initializing the Map - **Declaring Map Types** *

- `GoogleMap.MAP_TYPE_TERRAIN` is similar to a normal map, but textures are added to display changes in elevation in the environment. These textures are most visible when the map is angled with a two-finger drag.

2. Initializing the Map - **Declaring Map Types** *

- `GoogleMap.MAP_TYPE_NONE` is similar to a normal map, but doesn't display any labels or coloration for the type of environment in an area. It does allow for displaying traffic and other overlays on the map.



2. Initializing the Map – Creating the API Client *

- Create your `GoogleApiClient` and initiate `LocationServices` to get your user's current location.

```
@Override
public void onViewCreated(View view, Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);

    setHasOptionsMenu(true);

    mGoogleApiClient = new GoogleApiClient.Builder( getActivity() )
            .addConnectionCallbacks( this )
            .addOnConnectionFailedListener( this )
            .addApi( LocationServices.API )
            .build();

    initListeners();
}
```

2. Initializing the Map – Creating the API Client

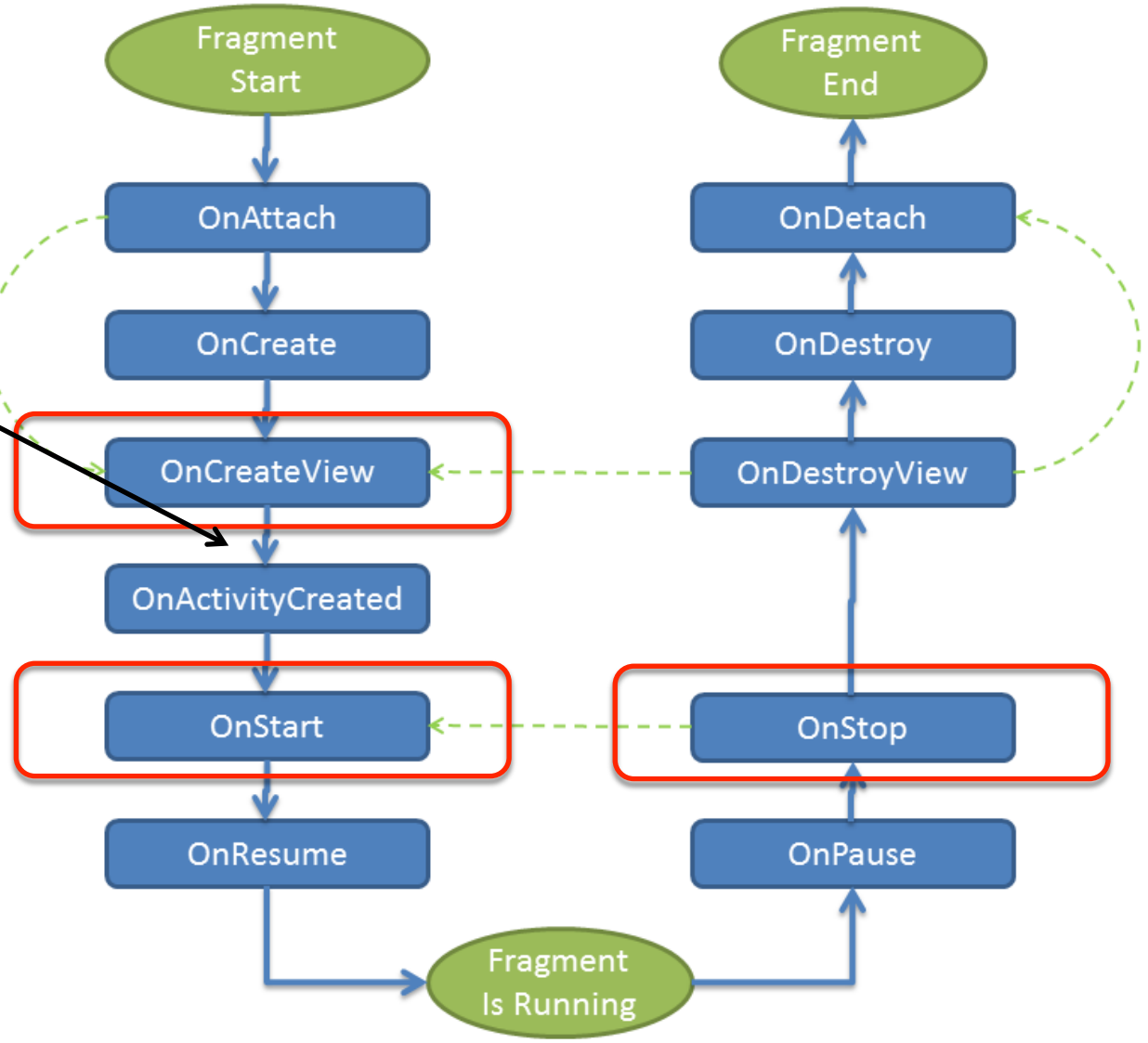
- The `initListeners` method binds the interfaces that you declared at the top of the class with the `GoogleMap` object associated with `SupportMapFragment`.

```
private void initListeners() {
    getMap().setOnMarkerClickListener(this);
    getMap().setOnMapLongClickListener(this);
    getMap().setOnInfoWindowClickListener( this );
    getMap().setOnMapClickListener(this);
}
```

- Note: the `GoogleApiClient` and listeners are created and bound from `onViewCreated` rather than the typical `onCreate`. The `GoogleMap` object has not been initialized when `onCreate` is called - need to wait until the view is fully created before trying to call `getMap` in order to avoid a `NullPointerException`.

Recap - Fragment Lifecycle *

- `OnViewCreated()` is called here
- Immediately after `OnCreateView()`



2. Initializing the Map – Configuring the Map *

- Connect the `GoogleApiClient` in `onStart`.

```
@Override
public void onStart() {
    super.onStart();
    mGoogleApiClient.connect();
}

@Override
public void onStop() {
    super.onStop();
    if( mGoogleApiClient != null && mGoogleApiClient.isConnected() ) {
        mGoogleApiClient.disconnect();
    }
}
```

2. Initializing the Map – Configuring the Map *

- Once connected, you can grab the user's most recently retrieved location and use that for aiming the map camera.

```
@Override
public void onConnected(Bundle bundle) {
    mCurrentLocation = LocationServices
        .FusedLocationApi
        .getLastLocation( mGoogleApiClient );

    initCamera( mCurrentLocation );
}
```

2. Initializing the Map – Configuring the Map *

- Once connected, you can grab the user's most recently retrieved location and use that for aiming the map camera.

```
private void initCamera( Location location ) {
    CameraPosition position = CameraPosition.builder()
            .target( new LatLng( location.getLatitude(),
                location.getLongitude() ) )
            .zoom( 16f )
            .bearing( 0.0f )
            .tilt( 0.0f )
            .build();

    getMap().animateCamera( CameraUpdateFactory
            .newCameraPosition( position ), null );

    getMap().setMapType( MAP_TYPES[curMapTypeIndex] );
    getMap().setTrafficEnabled( true );
    getMap().setMyLocationEnabled( true );
    getMap().getUiSettings().setZoomControlsEnabled( true );
}
```

3. Marking Locations *

- One of the most used map features involves indicating locations with markers. Since a latitude and longitude are needed for adding a marker, use the `OnMapClickListener` to allow the user to pick a spot on the map to place a `Marker` object.

```
@Override
public void onMapClick(LatLng latLng) {

    MarkerOptions options = new MarkerOptions().position( latLng );
    options.title( getAddressFromLatLng( latLng ) );

    options.icon( BitmapDescriptorFactory.defaultMarker() );
    getMap().addMarker( options );
}
```

- This method creates a generic red marker where the user has tapped.

3. Marking Locations *

- If you want to avoid using the generic colored pins for your location markers, or setting a marker as draggable you can set those options via the `MarkerOptions` object.

```
MarkerOptions options = new MarkerOptions().position( latLng );
options.title( getAddressFromLatLng( latLng ) );

options.icon( BitmapDescriptorFactory.fromBitmap(
        BitmapFactory.decodeResource( getResources(),
                R.mipmap.ic_launcher ) ) );

getMap().addMarker( options );
```

3. Marking Locations *

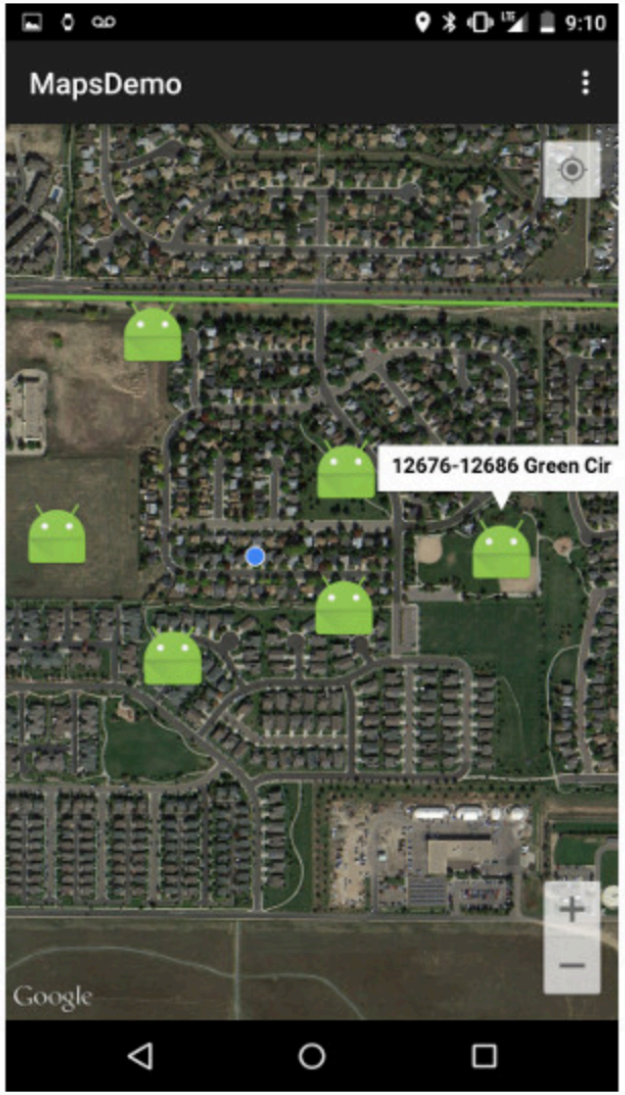
- The `getAddressFromLatLng` method is being used in both click methods. This is a helper method that takes a `LatLng` and runs it through a `Geocoder` to get a street address.

```
private String getAddressFromLatLng( LatLng latLng ) {
    Geocoder geocoder = new Geocoder( getActivity() );

    String address = "";
    try {
        address = geocoder
          .getFromLocation( latLng.latitude, latLng.longitude, 1 )
          .get( 0 ).getAddressLine( 0 );
    } catch (IOException e ) {
    }

    return address;
}
```

```
@Override
public boolean onMarkerClick(Marker marker) {
    marker.showInfoWindow();
    return true;
}
```

3. Marking Locations

- The previous few slides would give us something like this on a map



4. Drawing on the Map

- The `GoogleMap` object has a set of methods that make it easy to draw shapes and place images onto the map.
- To draw a simple circle, you only need to create a `CircleOptions` object, set a radius and center location, and define the stroke/fill colors and size.
- Once you have a `CircleOptions` object, you can call `addCircle` to draw the defined circle on top of the map.
- Just like when placing markers, objects that are drawn on the map return an object of the drawn item type so it can be referenced later if needed.

4. Drawing on the Map

- A `drawCircle` helper method

```
private void drawCircle( LatLng location ) {
    CircleOptions options = new CircleOptions();
    options.center( location );
    //Radius in meters
    options.radius( 10 );
    options.fillColor( getResources()
        .getColor( R.color.fill_color ) );
    options.strokeColor( getResources()
        .getColor( R.color.stroke_color ) );
    options.strokeWidth( 10 );
    getMap().addCircle(options);
}
```

5. Styling your Map

- Video next…

All Available via Google Play Services

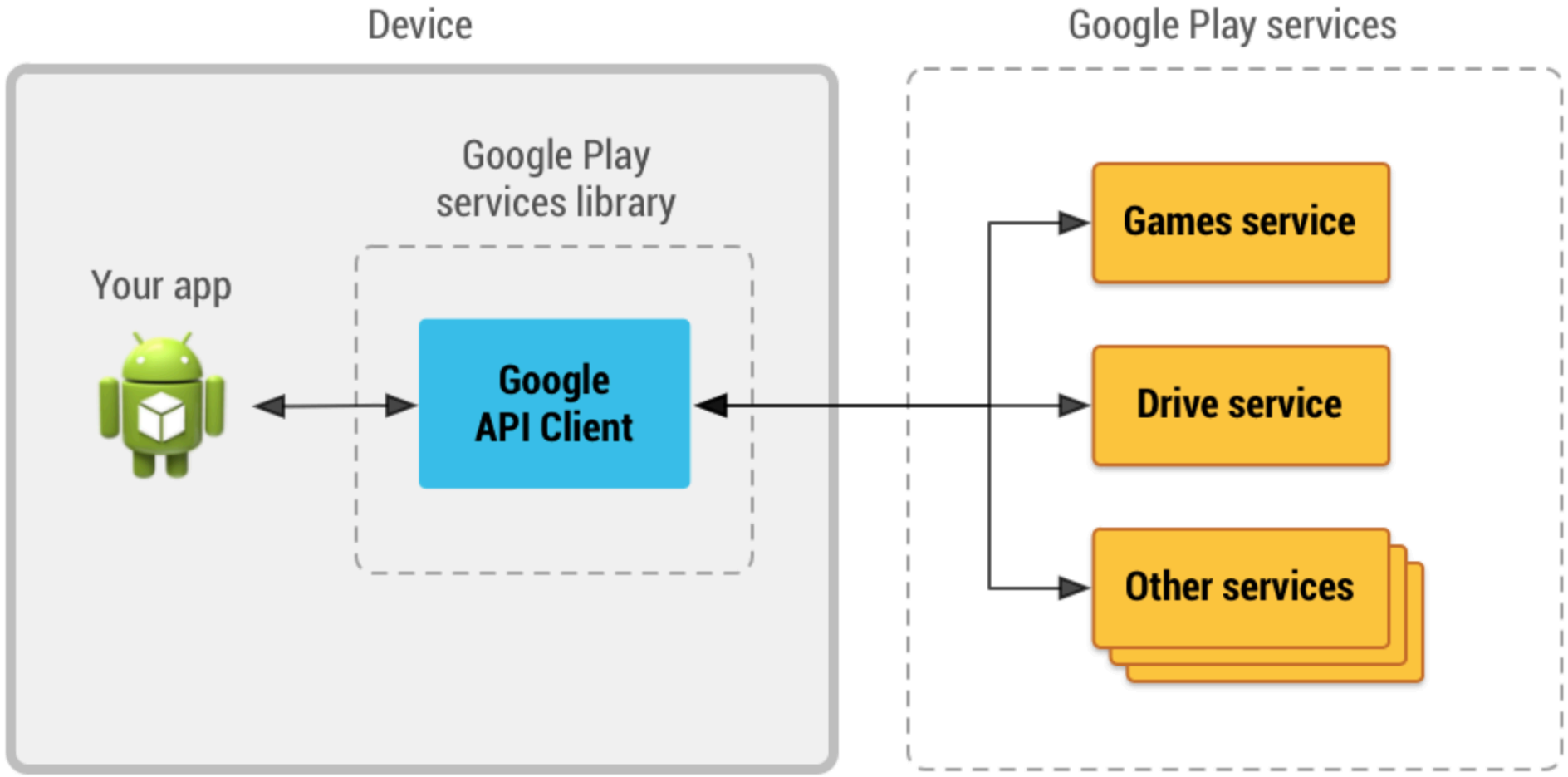


Figure 1: An illustration showing how the Google API Client provides an interface for connecting and making calls to any of the available Google Play services such as Google Play Games and Google Drive.

CoffeeMate 6.0

Code Highlights

fragment_map layout * (What you'll do initially in the labs)

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="6dp"
    android:paddingLeft="64dp"
    android:paddingRight="64dp"
    android:paddingTop="6dp"
    tools:context=".activities.Map">

    <fragment
        android:name="com.google.android.gms.maps.MapFragment"
        android:id="@+id/map"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</RelativeLayout>
```

- Here we declare the `MapFragment` element within our layout.
- Note the resource id.
- Our `Map` Activity.

```
public class Map extends AppCompatActivity {

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

`manifest` file *

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ie.cm">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="ie.cm.permission.MAPS_RECEIVE" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-feature android:name="android.hardware.location.gps"/>

    <application
        android:name=".main.CoffeeMateApp"
        android:allowBackup="true"
        android:icon="@mipmap/ic_cm_launcher"
        android:label="CoffeeMate"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity...>
        <activity...>
        <activity...>
        <activity android:name=".activities.Map"></activity>

        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="AIzaSy[redacted]-Khw" />
    </application>

</manifest>
```

- Setting the necessary permissions
- Our `Google API Key`

MapsFragment – interfaces/instance variables *

```
public class MapsFragment extends MapFragment implements
        GoogleMap.OnInfoWindowClickListener,
        GoogleMap.OnMapClickListener,
        GoogleMap.OnMarkerClickListener,
        OnMapReadyCallback,
        VolleyListener{

    private LocationRequest                 mLocationRequest;
    private FusedLocationProviderClient     mFusedLocationClient;
    private LocationCallback                mLocationCallback;
    private List<Coffee>                    mCoffeeList;
    private long                            UPDATE_INTERVAL = 5000; /* 5 secs */
    private long                            FASTEST_INTERVAL = 1000; /* 1 sec */
    private GoogleMap                       mMap;
    private float                           zoom = 13f;

    public CoffeeMateApp                    app = CoffeeMateApp.getInstance();

    private static final int                PERMISSION_REQUEST_CODE = 200;
```

- Here we declare the interfaces our custom `MapFragment` (MapsFragment) implements.
- Interfaces for `Volley` & Location Updates/Callbacks.
- Variables to keep track of location requests, the map etc.

GoogleApiClient Setup *

```
// [START build_client]
// Build a GoogleApiClient with access to the Google Sign-In API and the
// options specified by mGoogleSignInOptions.
app.mGoogleApiClient = new GoogleApiClient.Builder(this)
        .enableAutoManage(this /* FragmentActivity */,
                          this /* OnConnectionFailedListener */)
        .addApi(Auth.GOOGLE_SIGN_IN_API, app.mGoogleSignInOptions)
        .addApi(LocationServices.API)
        .build();
// [END build_client]
```

- Here we build our `GoogleApiClient` specifying the LocationServices API.
- It's actually common practice to 'rebuild' your api client (can actually improve performance)

MapsFragment – onResume() *

```
@Override
public void onResume() {
    super.onResume();
    getMapAsync(this);
    CoffeeApi.attachListener(this);
    CoffeeApi.getAll("/coffees/" + app.googleToken, null);
    if (checkPermission()) {
        if (app.mCurrentLocation != null) {
            Toast.makeText(getActivity(), "GPS location was found!", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(getActivity(), "Current location was null, Setting Default Values!",
                    Toast.LENGTH_SHORT).show();
            app.mCurrentLocation = new Location("Waterford City Default (WIT)");
            app.mCurrentLocation.setLatitude(52.2462);
            app.mCurrentLocation.setLongitude(-7.1202);
        }
        if(mMap != null) {
            initCamera(app.mCurrentLocation);
            mMap.setMyLocationEnabled(true);
        }
        startLocationUpdates();
    }
    else if (!checkPermission()) {
        requestPermission();
    }
}
```

- Acquire `GoogleMap` (automatically initializes the maps system and the view)
- Get all the coffees to display on map
- Zoom in Camera to current location

MapsFragment – onMapReady() *

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    mMap.setMapType(MAP_TYPES[curMapTypeIndex]);

    initListeners();
    if(checkPermission()) {
        mMap.setMyLocationEnabled(true);
        initCamera(app.mCurrentLocation);
    }
    else if (!checkPermission()) {
        requestPermission();
    }
        mMap.getUiSettings().setMapToolbarEnabled(true);
        mMap.getUiSettings().setCompassEnabled(true);
        mMap.getUiSettings().setMyLocationButtonEnabled(true);
        mMap.getUiSettings().setAllGesturesEnabled(true);
        mMap.setTrafficEnabled(true);
        mMap.setBuildingsEnabled(true);
        mMap.getUiSettings().setZoomControlsEnabled(true);
}
```

- Bind to our `GoogleMap` instance and set its initial type.
- Check for the necessary permissions & zoom
- Request Permissions if not already allowed
- Set the Map (*mMap*) properties

MapsFragment – Permissions *

```
//http://www.journaldev.com/10409/android-handling-runtime-permissions-example
private boolean checkPermission() {
    int result = ContextCompat.checkSelfPermission(getActivity(), ACCESS_FINE_LOCATION);
    int result1 = ContextCompat.checkSelfPermission(getActivity(), CAMERA);

    return result == PackageManager.PERMISSION_GRANTED && result1 == PackageManager.PERMISSION_GRANTED;
}

private void requestPermission() {
    ActivityCompat.requestPermissions(getActivity(), new String[]{ACCESS_FINE_LOCATION, CAMERA},
                                                                    PERMISSION_REQUEST_CODE);
}
```

- Checking to see if Location & Camera permissions are allowed
- Requesting Location & Camera permissions

MapsFragment – Permissions *

```
@Override
public void onRequestPermissionsResult(int requestCode, String permissions[], int[] grantResults) {
    switch (requestCode) {
        case PERMISSION_REQUEST_CODE:
            if (grantResults.length > 0) {

                boolean locationAccepted = grantResults[0] == PackageManager.PERMISSION_GRANTED;
                boolean cameraAccepted = grantResults[1] == PackageManager.PERMISSION_GRANTED;

                if (locationAccepted && cameraAccepted) {
                    Snackbar.make(getView(), "Permission Granted, Now you can access location data and camera.",
                            Snackbar.LENGTH_LONG).show();
                    if(checkPermission())
                        mMap.setMyLocationEnabled(true);
                    startLocationUpdates();
                }
                else {

                    Snackbar.make(getView(), "Permission Denied, You cannot access location data and camera.",
                            Snackbar.LENGTH_LONG).show();

                    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                        if (shouldShowRequestPermissionRationale(ACCESS_FINE_LOCATION)) {
                            showMessageOKCancel("You need to allow access to both the permissions",
                                    (dialog, which) -> {
                                            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                                                requestPermissions(new String[]{ACCESS_FINE_LOCATION, CAMERA},
                                                        PERMISSION_REQUEST_CODE);
                                            }
                                    });
                            return;
                        }
                    }
                }
            }
            break;
    }
}
```

- Retrieving permission status
- Updating the User and starting location updates on permission granted

MapsFragment – Tracking Location (1) *

```
public void startLocationUpdates() {
    try {
            mFusedLocationClient.requestLocationUpdates(mLocationRequest,
                mLocationCallback, Looper.myLooper());
    }
    catch(SecurityException se) {
        Toast.makeText(getActivity(),
                "Check Your Permissions on Location Updates",
                Toast.LENGTH_SHORT).show();
    }
}
```

- Use the `FusedLocationClient` instance to *requestLocationUpdates*

MapsFragment – Tracking Location (2) *

```
 /* Creates a callback for receiving location events.*/
private void createLocationCallback() {
    mLocationCallback = new LocationCallback() {
        @Override
        public void onLocationResult(LocationResult locationResult) {
            super.onLocationResult(locationResult);

            app.mCurrentLocation = locationResult.getLastLocation();
            initCamera(app.mCurrentLocation);
        }
    };
}
```

- Update our current location (`mCurrentLocation`) and initialise/reposition the camera

MapsFragment – Helper Methods *

```
public void initListeners() {
    mMap.setOnMarkerClickListener(this);
    mMap.setOnInfoWindowClickListener(this);
    mMap.setOnMapClickListener(this);
}
```

```
private void initCamera(Location location) {

    if (zoom != 13f && zoom != mMap.getCameraPosition().zoom)
            zoom = mMap.getCameraPosition().zoom;

    CameraPosition position = CameraPosition.builder()
            .target(new LatLng(location.getLatitude(),
                    location.getLongitude()))
            .zoom(zoom)
            .bearing(0.0f)
            .tilt(0.0f)
            .build();

    mMap.animateCamera(CameraUpdateFactory
            .newCameraPosition(position), null);
}
```

- Adding necessary listeners to our `GoogleMap` reference.
- Position/reposition the Camera based on current location and set zoom ratio.

MapsFragment – Adding Coffee Markers *

```
@Override
public void setList(List list) {
    addCoffees(list);
    Log.v("coffeemate", "List to add is : " + list);
}
```

- Triggered by our `CoffeeApi` callback
- Traversing our list of coffees and adding a location marker to the map

```
public void addCoffees(List<Coffee> list)
{
    for(Coffee c : list)
        mMap.addMarker(new MarkerOptions()
                .position(new LatLng(c.marker.coords.latitude, c.marker.coords.longitude))
                .title(c.name + " €" + c.price)
                .snippet(c.shop + " " + c.address)
                .icon(BitmapDescriptorFactory.fromResource(R.drawable.coffee_icon)));

}
```

AddFragment – Adding a single Coffee *

- To demonstrate the true value of using Fragments, we embed our existing `MapsFragment` inside our `AddFragment` (demonstrating even another new(ish) feature in Android)
- We get all the existing functionality of our custom map, and just need to make a few minor changes to our existing `AddFragment` to allow us to store the location of the coffee as we add it.

AddFragment – Helper Methods *

```
private String getAddressFromLocation( Location location ) {
    Geocoder geocoder = new Geocoder( getActivity() );

    String strAddress = "";
    Address address;
    try {
        address = geocoder
                .getFromLocation( location.getLatitude(),
                        location.getLongitude(), 1 )
                .get( 0 );
        strAddress = address.getAddressLine(0) +
                " " + address.getAddressLine(1) +
                " " + address.getAddressLine(2);
    }
    catch (IOException e ) {
    }

    return strAddress;
}
```

- Using the `Geocoder` class to extract an address from a location (for storing with the coffee data)

fragment_add *

- Modifying our fragment layout to include another fragment
- Referencing our existing `MapsFragment` fragment

```
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="ie.cm.fragments.AddFragment">

    <Button...>
    <RatingBar...>
    <EditText...>
    <EditText...>
    <TextView...>
    <TextView...>
    <EditText...>
    <TextView...>
    <TextView...>

    <fragment
        android:name="ie.cm.fragments.MapsFragment"
        android:id="@+id/addmap"
        android:layout_width="364dp"
        android:layout_height="138dp"
        android:layout_gravity="center_horizontal|bottom"
        android:layout_marginLeft="8dp"
        app:layout_constraintLeft_toLeftOf="parent"
        android:layout_marginRight="8dp"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        android:layout_marginBottom="8dp"
        app:layout_constraintTop_toTopOf="parent"
        android:layout_marginTop="8dp"
        app:layout_constraintVertical_bias="0.737"
        app:layout_constraintHorizontal_bias="0.533"
        tools:layout_editor_absoluteY="267dp"
        tools:layout_editor_absoluteX="5dp" />

</android.support.constraint.ConstraintLayout>
```

AddFragment – Adding a single Coffee *

```
public class AddFragment extends Fragment implements View.OnClickListener,
                                 OnMapReadyCallback {
```

```
if ((coffeeName.length() > 0) && (coffeeShop.length() > 0)
        && (price.length() > 0)) {
    Coffee c = new Coffee(coffeeName, coffeeShop, ratingValue,
            coffeePrice,
            false,
            app.googleToken,
            app.mCurrentLocation.getLatitude(),
            app.mCurrentLocation.getLongitude(),
            app.googlePhotoURL,getAddressFromLocation( app.mCurrentLocation ));

    CoffeeApi.post("/coffees/" + app.googleToken,c);
    resetFields();
    CoffeeApi.getAll("/coffees/" + app.googleToken, null);
      Intent intent = new Intent(getActivity(), Home.class);
      getActivity().startActivity(intent);
```

- Implement our map callback
- Create a new coffee using the current location, user photo url and full address.
- Retrieving all our coffees to update the map (showing the newly added coffee)

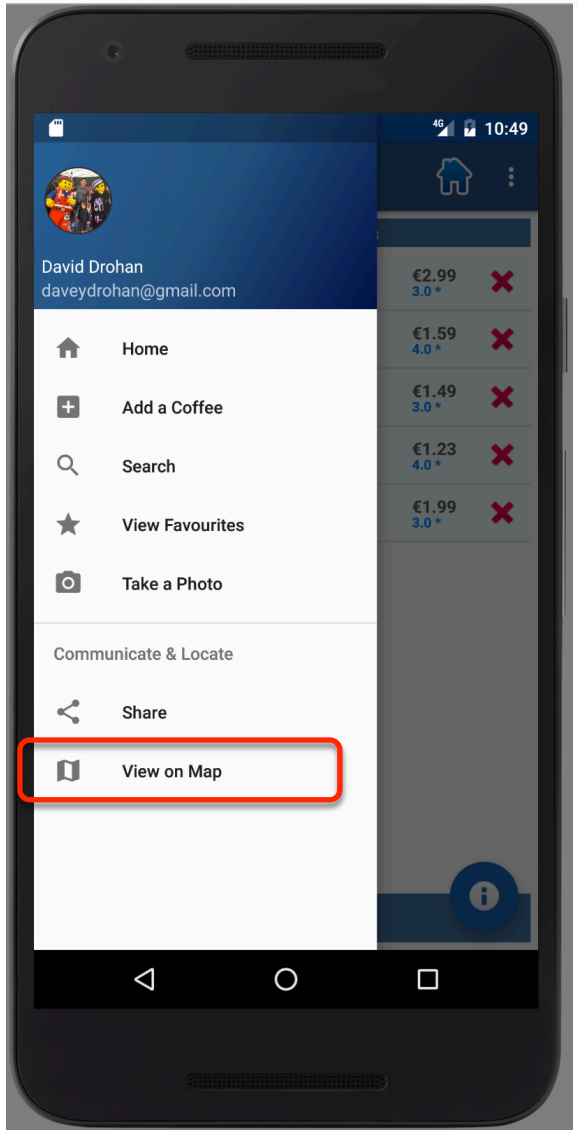
AddFragment – Updating the Map *

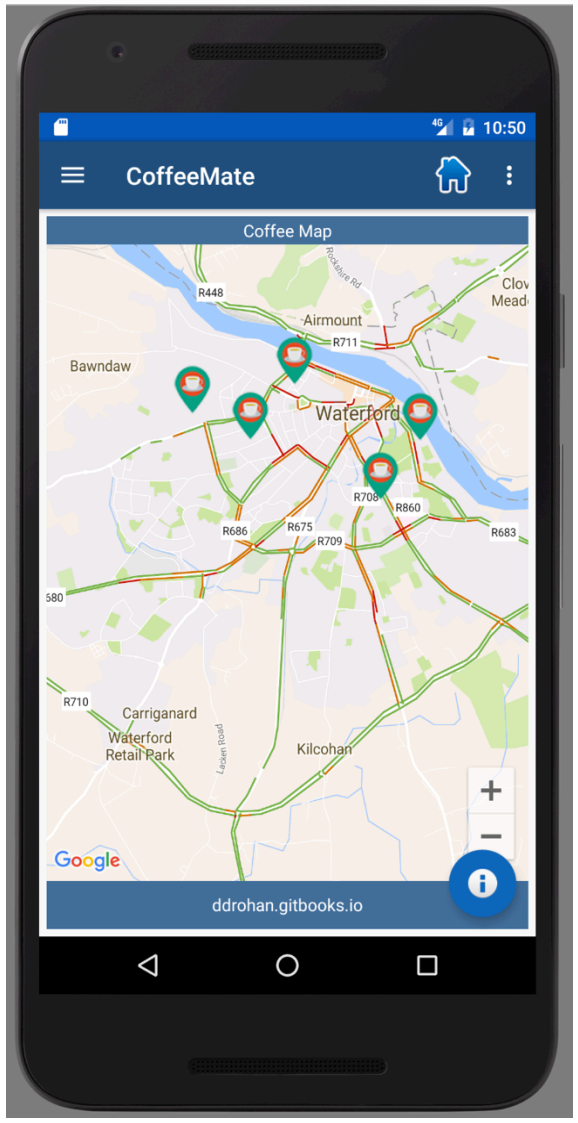
```
@Override
public void onMapReady(GoogleMap googleMap) {
    googleMap.clear();
    addCoffees(app.coffeeList,googleMap);
}
```

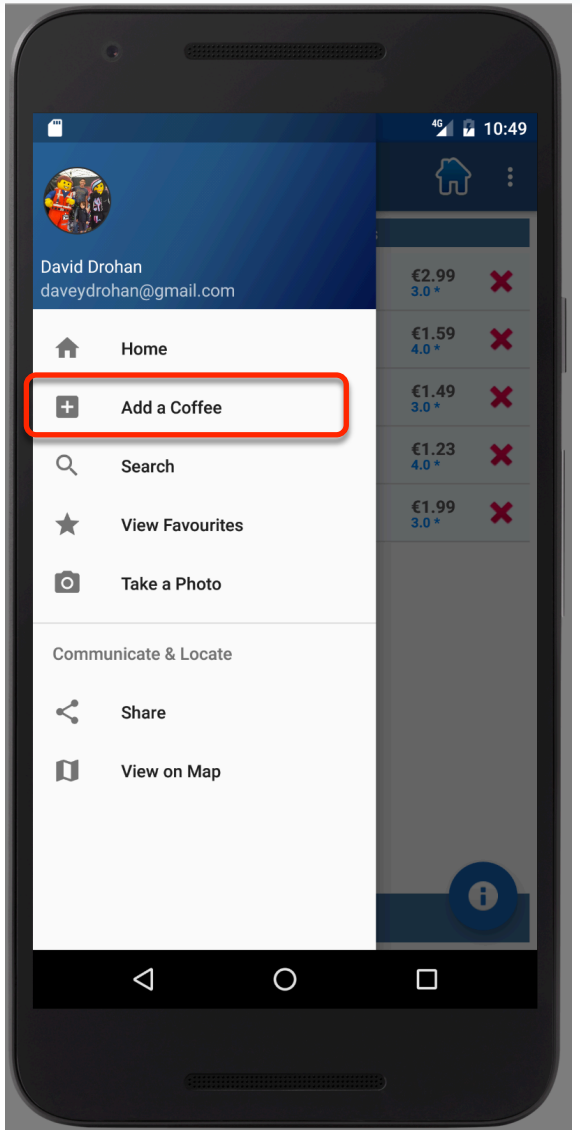
- Our callback for a map reference and clearing the map
- Adding the new list of coffees

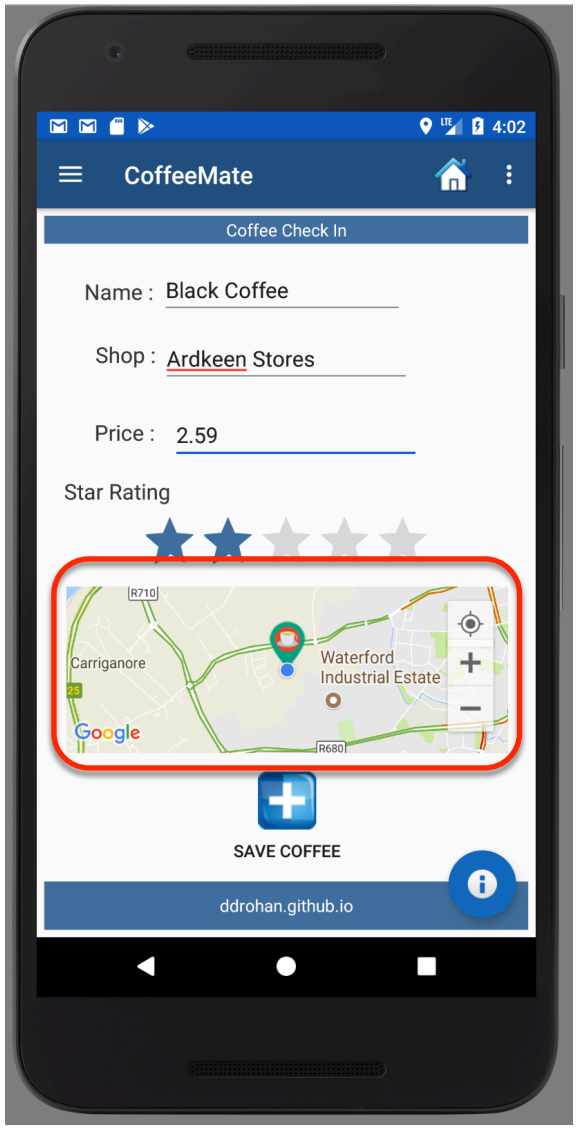
```
public void addCoffees(List<Coffee> list,GoogleMap mMap)
{
    for(Coffee c : list)
        mMap.addMarker(new MarkerOptions()
                .position(new LatLng(c.marker.coords.latitude, c.marker.coords.longitude))
                .title(c.name + " €" + c.price)
                .snippet(c.shop + " " + c.address)
                .icon(BitmapDescriptorFactory.fromResource(R.drawable.coffee_icon)));
}
```

CoffeeMate 6.0 *









Summary

- Overview
- Installation & Registration of the Google Maps API ‘Key’
- Creating interactive Maps with GoogleMaps, (Support)MapFragments & FragmentActivitiys
- Creating & Adding Markers to Maps
- Custom Styling our Maps (Video)

Questions?


Thanks!