

# Mobile Application Development

Google Maps

# Google Maps

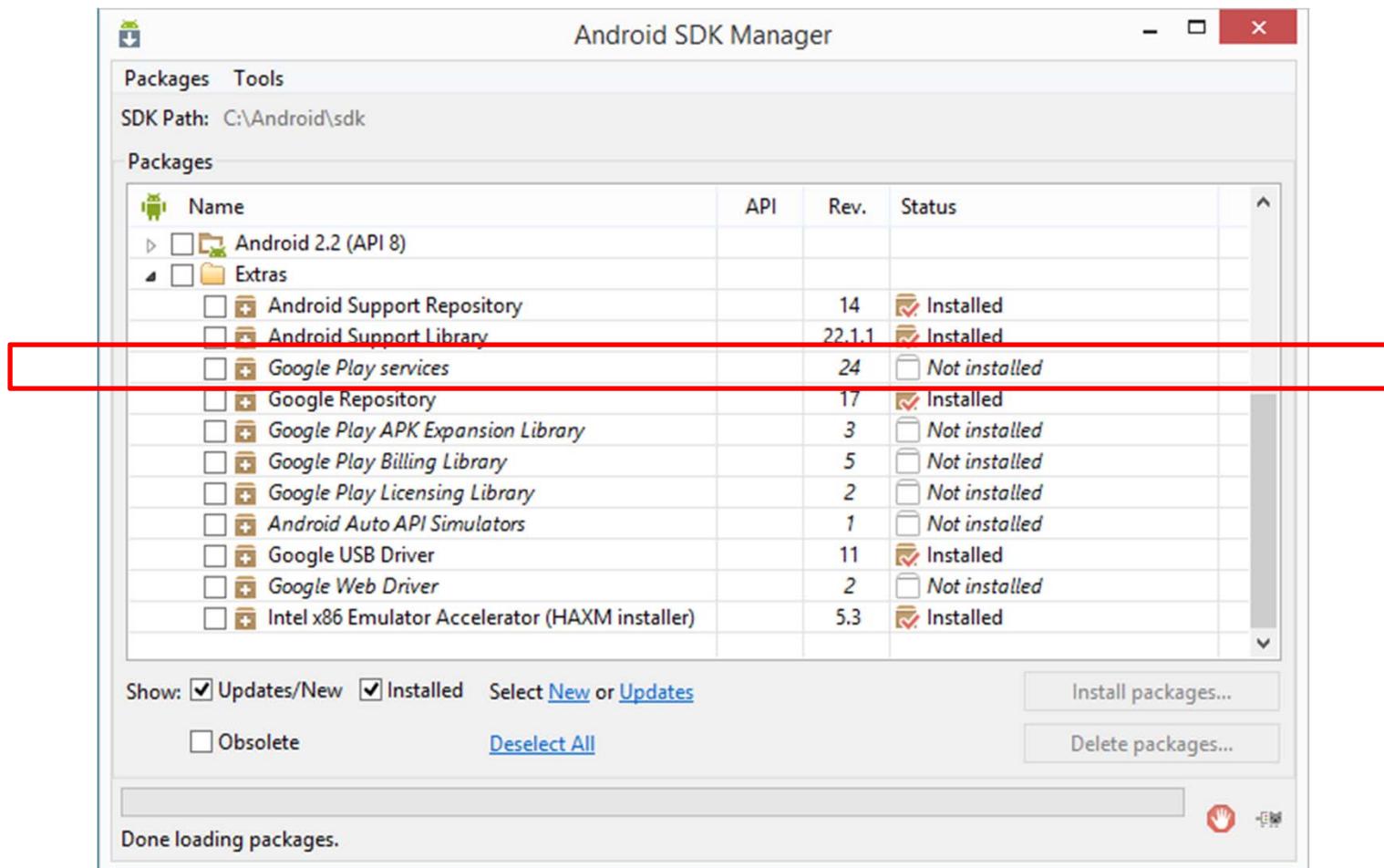
- Set Up
  - Install **Google Play Services SDK & Google APIs**
  - Create a **Virtual Device** that uses “Google APIs” Platform
- Create Your App
  - Get an **App Certificate** and the **Google Maps API Key**
  - **Add Google Play Services to Your Project**
  - Specify app settings in the **AndroidManifest.xml**
  - Add a **Map**

# **SETTING UP GOOGLE PLAY SERVICES**

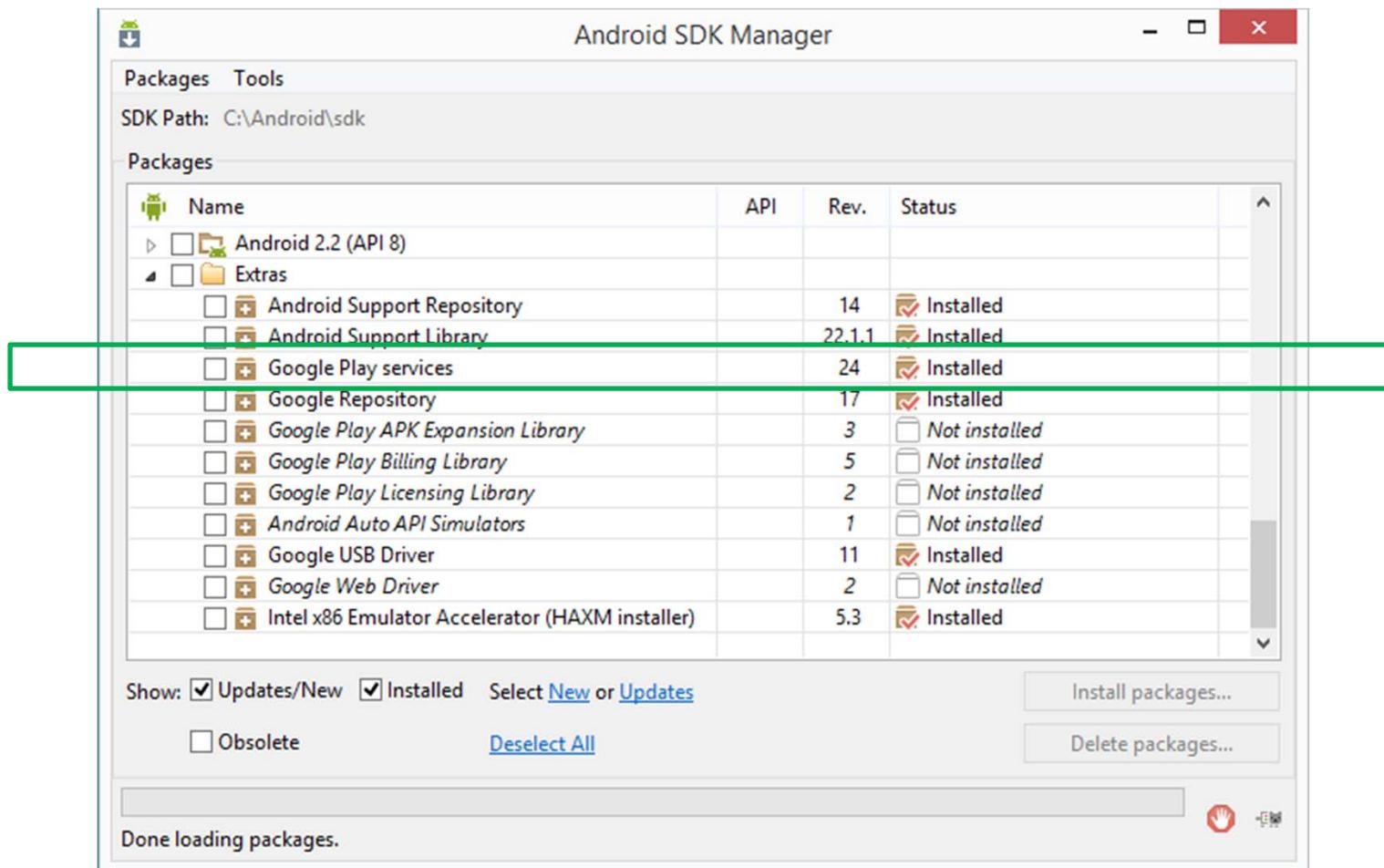
# Install Google Play Services SDK

- Google Maps Android API v2 is part of the Google Play services platform. To use Google Maps, set up the Google Play services SDK.
  - Open SDK Manager
  - Under “Extras”, see if “Google Play Services” is installed

# Install Google Play Services SDK



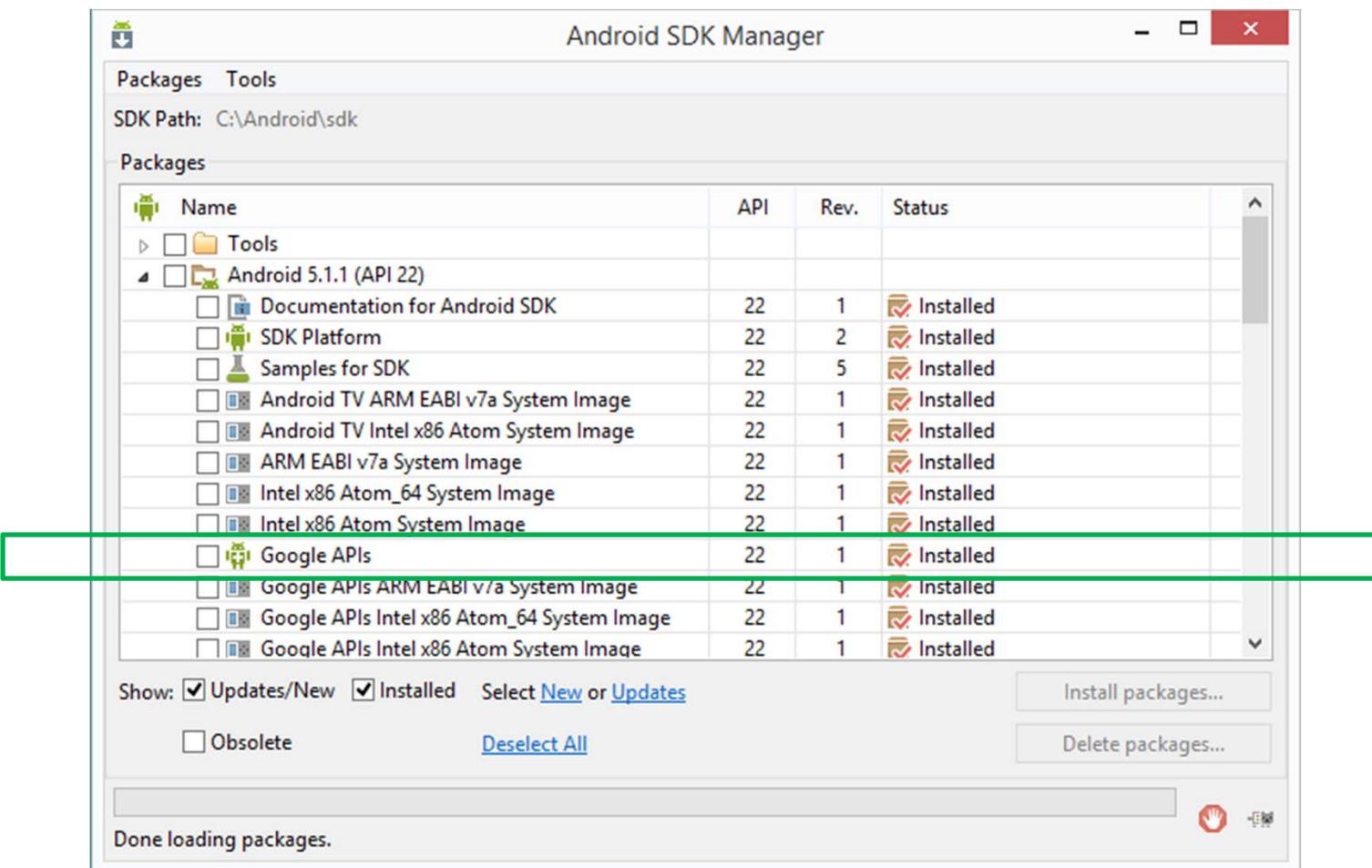
# Install Google Play Services SDK



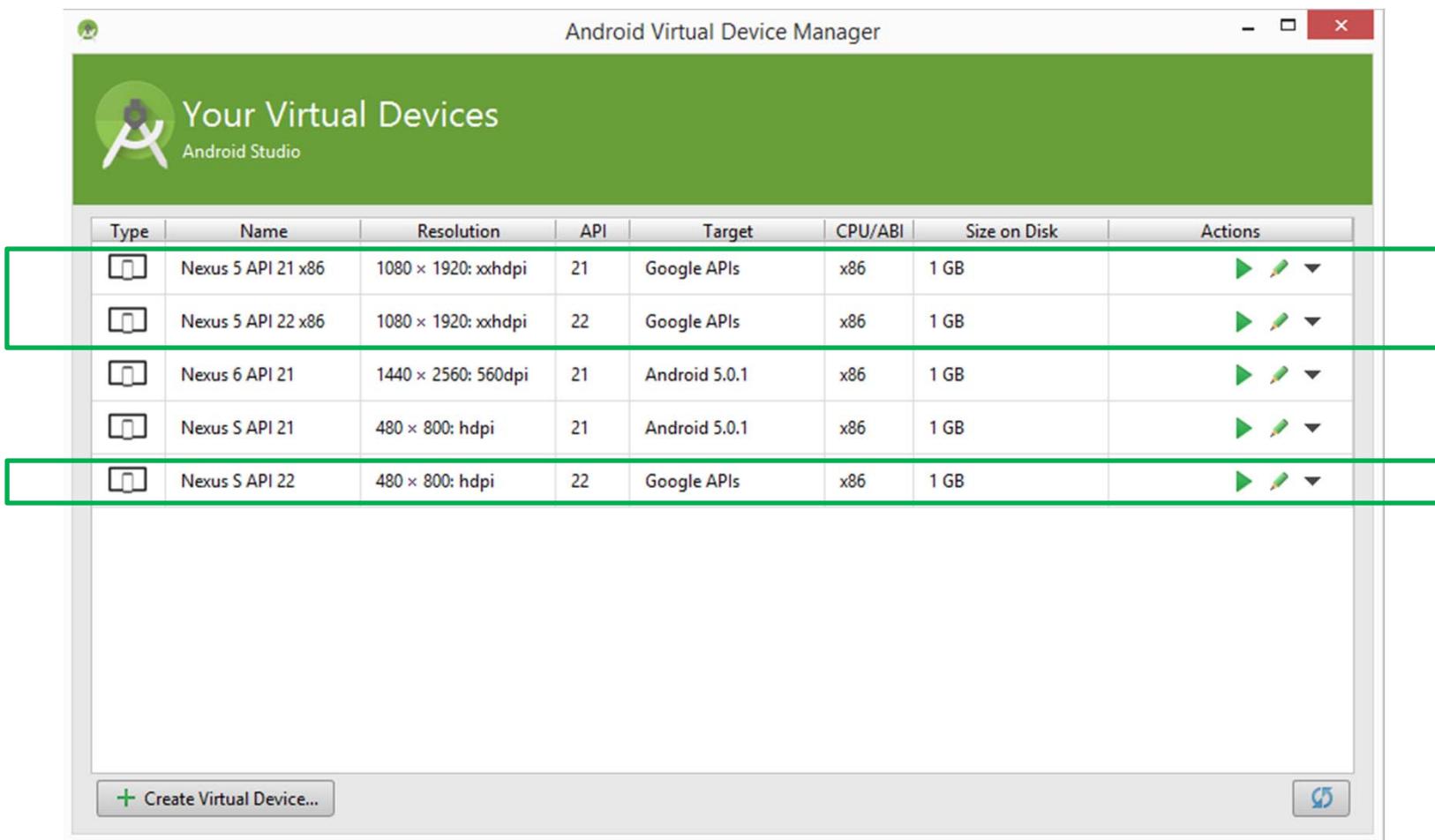
# Install Google APIs

- To test your app when using the Google Play services SDK, you must use either:
  - A compatible **Android device that runs Android 2.3 or higher** and includes **Google Play Store**.
  - The **Android emulator** with an AVD that runs the **Google APIs platform** based on Android 4.2.2 or higher.

# Install Google APIs



# Virtual Device (Google APIs)



# **INTEGRATING GOOGLE MAPS IN YOUR APPS**

# Google Maps

- Set Up
  - Install **Google Play Services SDK & Google APIs** 
  - Create a **Virtual Device** that uses “Google APIs” Platform 
- Create Your App
  - Get an **App Certificate** and the **Google Maps API Key**
  - **Add Google Play Services to Your Project**
  - Specify app settings in the **AndroidManifest.xml**
  - Add a **Map**

# Create Your App

- To access the Google Maps servers with the Maps API, you have to add a **Maps API key** to your application.
- You obtain a Maps API key from the **Google Developers Console** by providing your application's signing certificate and its package name.

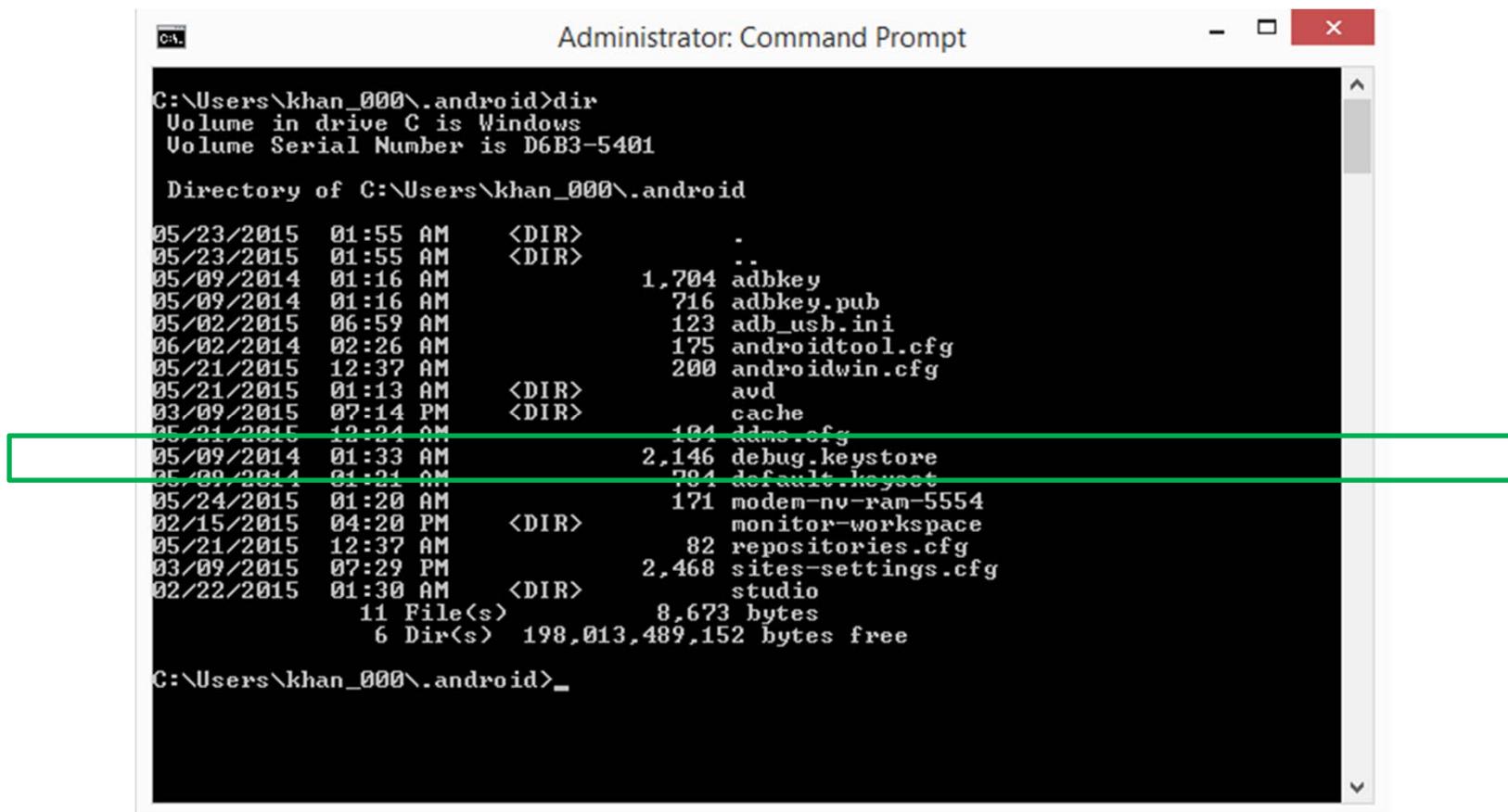
# Create Your App

- Obtaining a Google Maps API Key for your application requires the following steps:
  - Retrieve information about your **Application's Certificate**.
  - **Register a project in the Google Developers Console** and add the Maps API as a service for the project.
  - **Request Google Maps API Key** for your project.

# Retrieve Application's Certificate

- Your application's digital certificate (known as its SHA-1 fingerprint), is a unique text string generated from the commonly-used SHA-1 hashing algorithm.
- For development purposes, you need to get debug certificate fingerprint.  
To displaying the debug certificate fingerprint:
  - Locate your debug keystore file. The file name is **debug.keystore**, and is created the first time you build project.
  - On Windows you will typically find this file in this location:  
`C:\Users\your_user_name\.android\`

# Retrieve Application's Certificate



The screenshot shows an Administrator Command Prompt window with the title "Administrator: Command Prompt". The command entered is "C:\Users\khan\_000\.android>dir". The output lists various files and directories within the .android folder, including adbkey, adbkey.pub, adb\_usb.ini, androidtool.cfg, androidwin.cfg, avd, cache, ddmc.cfg, debug.keystore, default keystore, modem-nv-ram-5554, monitor-workspace, repositories.cfg, sites-settings.cfg, and studio. A green rectangular box highlights the "debug.keystore" file.

```
C:\Users\khan_000\.android>dir
Volume in drive C is Windows
Volume Serial Number is D6B3-5401

Directory of C:\Users\khan_000\.android

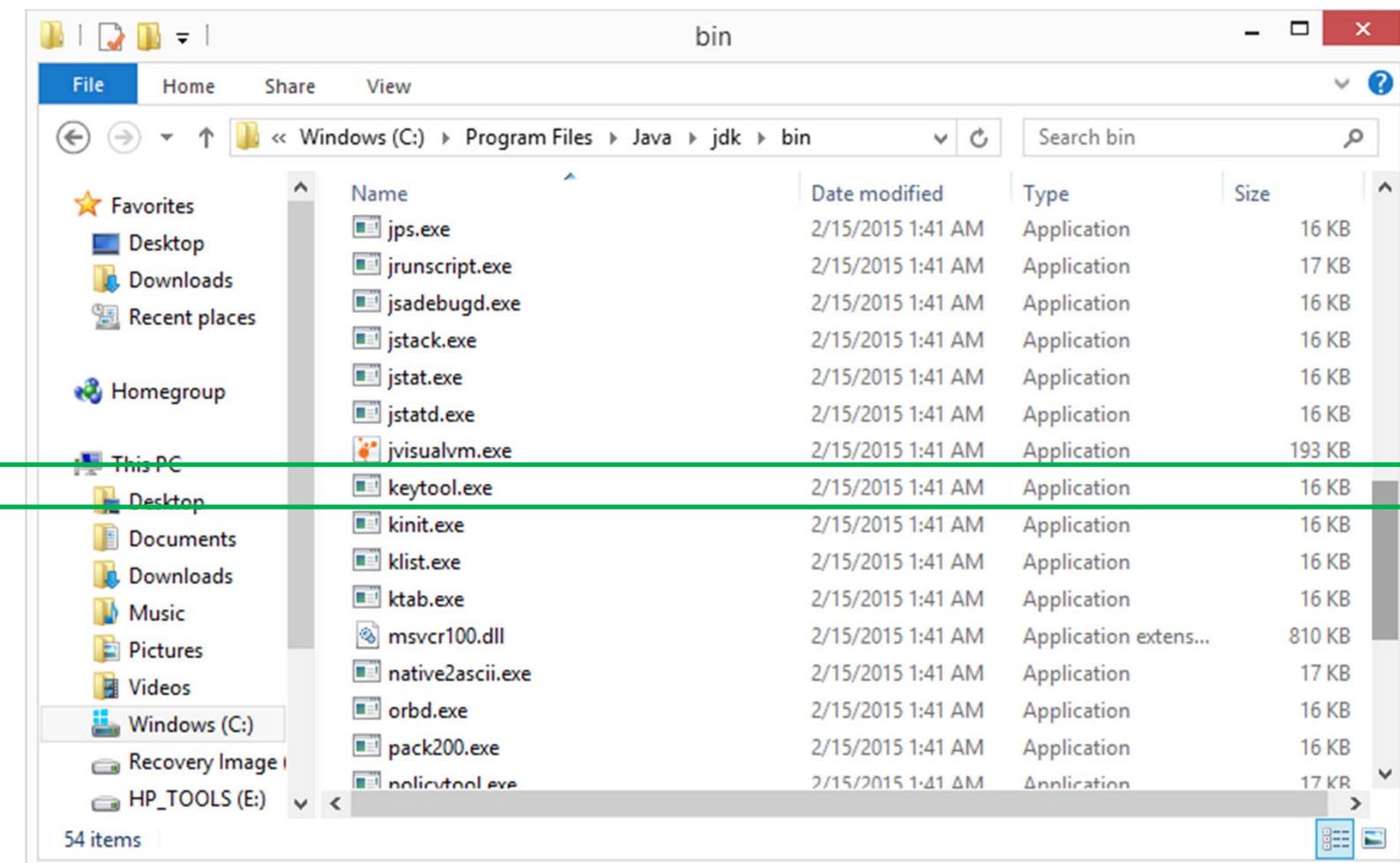
05/23/2015  01:55 AM    <DIR>      .
05/23/2015  01:55 AM    <DIR>      ..
05/09/2014  01:16 AM            1,704 adbkey
05/09/2014  01:16 AM            716 adbkey.pub
05/02/2015  06:59 AM            123 adb_usb.ini
06/02/2014  02:26 AM            175 androidtool.cfg
05/21/2015  12:37 AM            200 androidwin.cfg
05/21/2015  01:13 AM    <DIR>      avd
03/09/2015  07:14 PM    <DIR>      cache
05/21/2015  12:24 AM            184 ddmc.cfg
05/09/2014  01:33 AM            2,146 debug.keystore
05/09/2014  01:21 AM            794 default keystore
05/24/2015  01:20 AM            171 modem-nv-ram-5554
02/15/2015  04:20 PM    <DIR>      monitor-workspace
05/21/2015  12:37 AM            82 repositories.cfg
03/09/2015  07:29 PM            2,468 sites-settings.cfg
02/22/2015  01:30 AM    <DIR>      studio
                           11 File(s)      8,673 bytes
                           6 Dir(s)   198,013,489,152 bytes free

C:\Users\khan_000\.android>
```

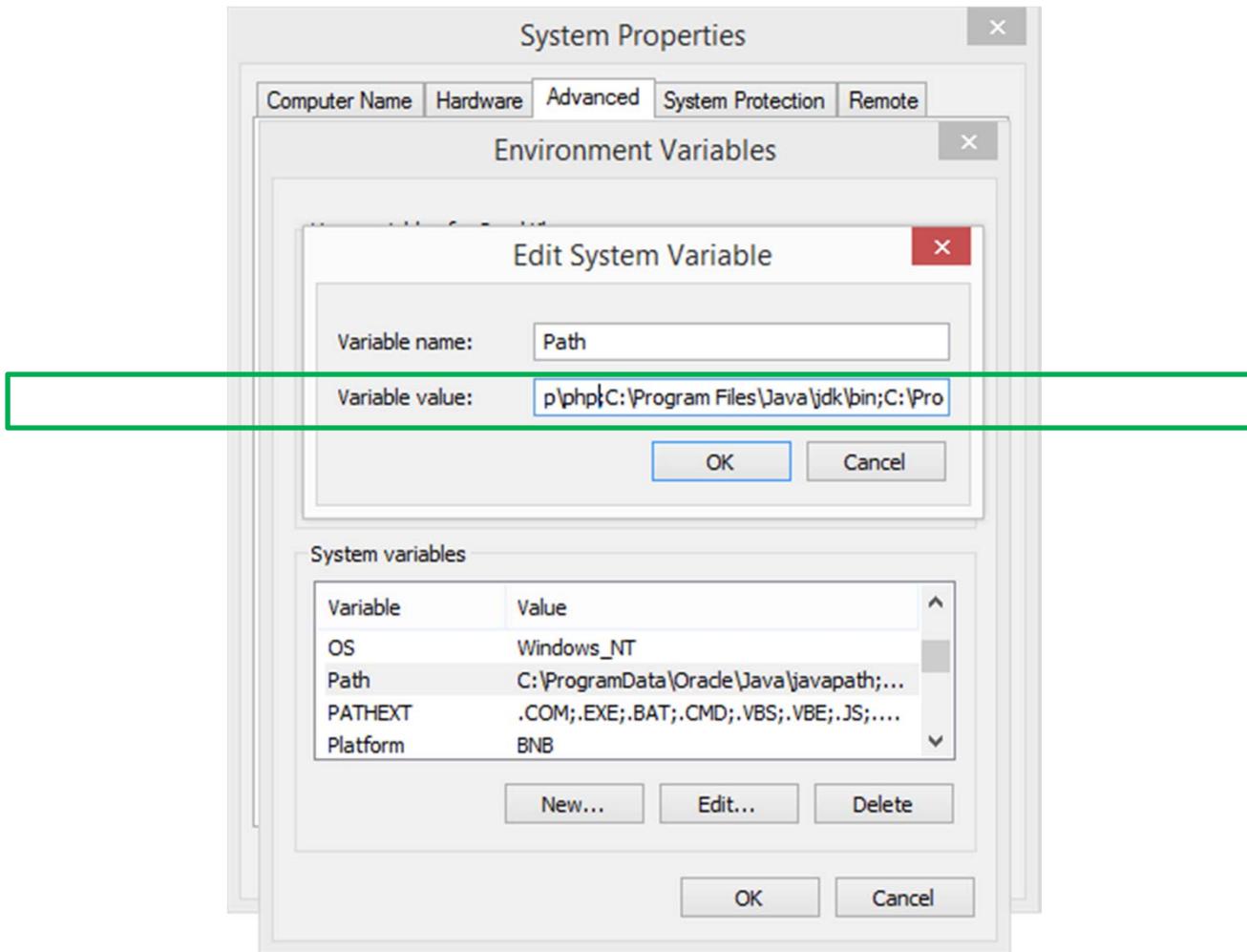
# Retrieve Application's Certificate

- You can display a certificate's SHA-1 fingerprint using the **keytool program**.
- Keytool program is located in your JDK's "bin" folder. In order to use this program from other locations, you will need to **set JDK's "bin" folder path in your system's environment variables**.

# Retrieve Application's Certificate



# Retrieve Application's Certificate



# Retrieve Application's Certificate

- Once you have **located debug.keystore** and are able to use **keytool program** in its folder, type the following command:

```
keytool -list -v -keystore "%USERPROFILE%\.android\debug.keystore" -  
alias androiddebugkey -storepass android -keypass android
```

- To **save output in a text file**, type the following command:

```
keytool -list -v -keystore "%USERPROFILE%\.android\debug.keystore" -  
alias androiddebugkey -storepass android -keypass android >  
somefilename.txt
```

# Retrieve Application's Certificate

- You should see output similar to this:

```
Alias name: androiddebugkey
Creation date: Jan 01, 2013
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 4aa9b300
Valid from: Mon Jan 01 08:04:04 UTC 2013 until: Mon Jan 01 18:04:04 PST
2033
Certificate fingerprints:
    MD5: AE:9F:95:D0:A6:86:89:BC:A8:70:BA:34:FF:6A:AC:F9
    SHA1: BB:0D:AC:74:D3:21:E1:43:07:71:9B:62:90:AF:A1:66:6E:44:5D:75
    Signature algorithm name: SHA1withRSA
    Version: 3
```

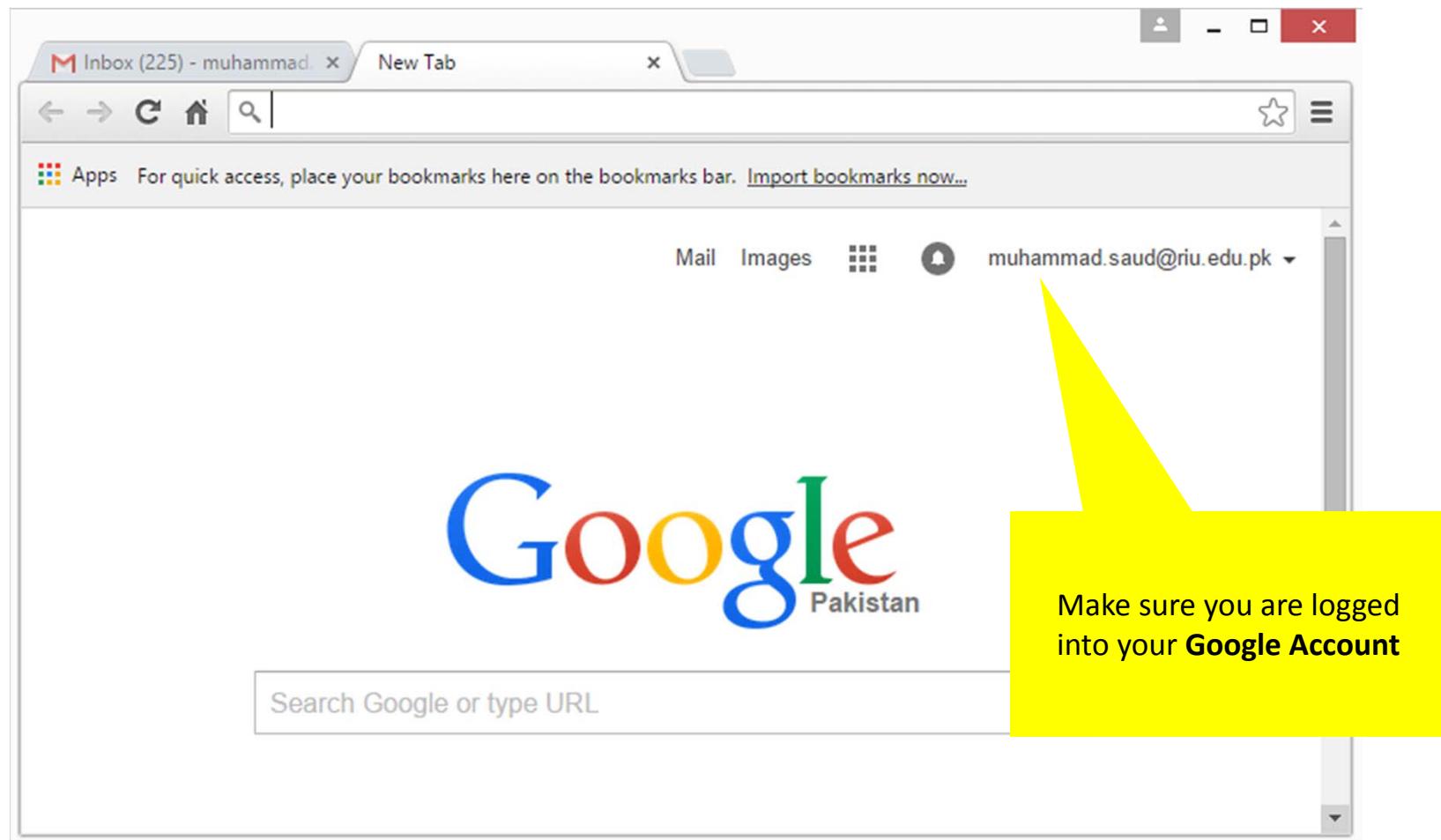
# Create Your App

- Obtaining a Google Maps API Key for your application requires the following steps:
  - Retrieve information about your **Application's Certificate**. 
  - **Register a project in the Google Developers Console** and add the Maps API as a service for the project.
  - **Request Google Maps API Key** for your project.

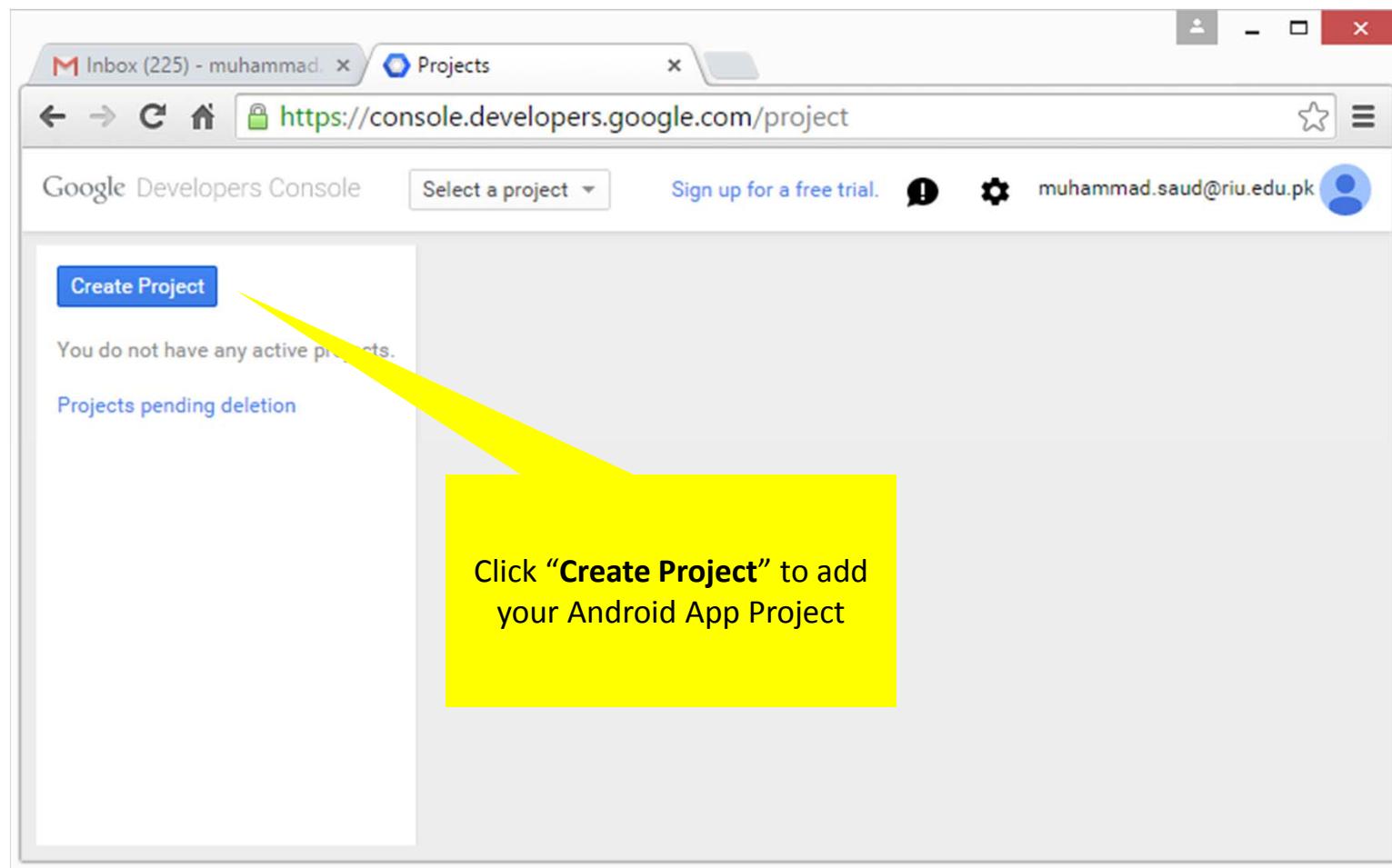
# Google Developers Console

- Follow these steps to create or modify a project for your application in the Google Developers Console and enable the Google Maps Android API.
  - Go to the Google Developers Console: <https://console.developers.google.com/>
  - Select a project, or create a new one.
  - In the sidebar on the left, **expand APIs & auth.**
    - Next, **click APIs**.
    - Select the Enabled APIs link in the API section to see a list of all your enabled APIs. Make sure that the API is on the list of enabled APIs. If you have not enabled it, select the API from the list of APIs, then select the **Enable API button for the API**.
    - The only API you need is the Google Maps Android API, although you can choose to enable other APIs for the same project too.
  - In the sidebar on the left, **select Credentials**.

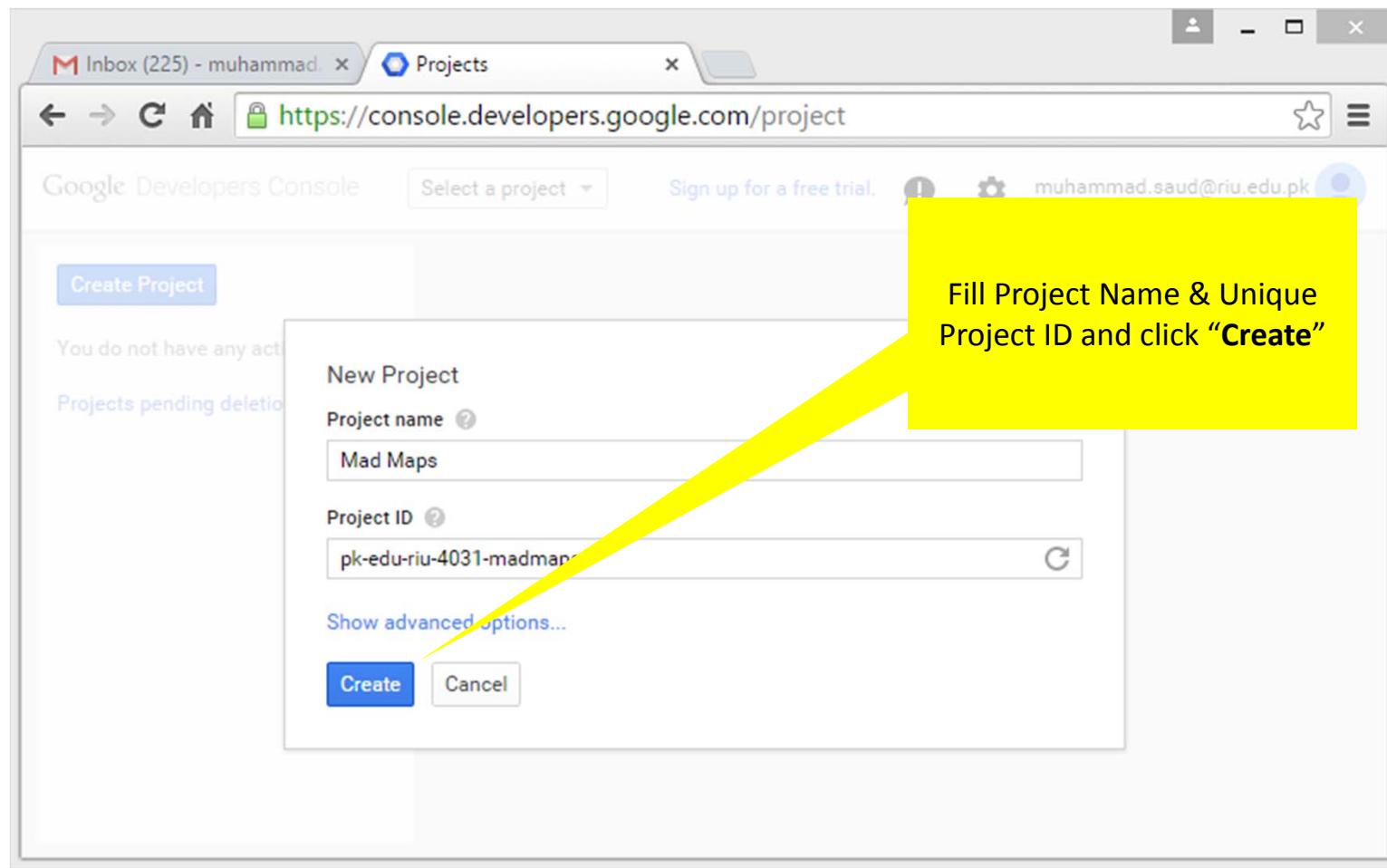
# Google Developers Console



# Google Developers Console



# Google Developers Console



# Google Developers Console

- Follow these steps to create or modify a project for your application in the Google Developers Console and enable the Google Maps Android API.
  - Go to the Google Developers Console: <https://console.developers.google.com/>
  - Select a project, or create a new one.
  - In the sidebar on the left, **expand APIs & auth.**
    - Next, **click APIs**.
    - Select the Enabled APIs link in the API section to see a list of all your enabled APIs. Make sure that the API is on the list of enabled APIs. If you have not enabled it, select the API from the list of APIs, then select the **Enable API button for the API**.
    - The only API you need is the Google Maps Android API, although you can choose to enable other APIs for the same project too.
  - In the sidebar on the left, **select Credentials**.

# Google Developers Console

The screenshot shows the Google Developers Console interface. The left sidebar contains navigation links such as Overview, Permissions, APIs & auth, APIs, Credentials, Consent screen, Push, Monitoring, Source Code, Deploy & Manage, Compute, Networking, and Storage. The main content area is titled "Enabled APIs (6)" and displays a list of APIs with their status and quota information. A yellow callout box points to the "Enabled APIs" tab with the instruction: "Click on “Enabled APIs” to see if Google Maps APIs are enabled or not".

API	Quota	Action
BigQuery API	0%	Disable
Debuglet Controller API	0%	Disable
Google Cloud Logging API	0%	Disable
Google Cloud SQL		Disable
Google Cloud Storage		Disable
Google Cloud Storage JSON API		

Enabled APIs (6)

Some APIs are enabled automatically. You can disable them if you're not using their services.

API ^ Quota

BigQuery API 0% Disable

Debuglet Controller API 0% Disable

Google Cloud Logging API 0% Disable

Google Cloud SQL

Google Cloud Storage

Google Cloud Storage JSON API

Click on “Enabled APIs” to see if Google Maps APIs are enabled or not

# Google Developers Console

The screenshot shows the Google Developers Console interface. The left sidebar lists various sections: Overview, Permissions, APIs & auth, APIs (which is selected), Credentials, Consent screen, Push, Monitoring, Source Code, Deploy & Manage, Compute, Networking, and Storage. The main content area is titled "API Library - Mad Maps". It shows a list of APIs under "Cloud" and "Google Maps APIs". Under "Cloud", there are links to BigQuery API, Cloud Storage API, Cloud Datastore API, Cloud Deployment Manager API, Cloud DNS API, and "More". Under "Google Maps APIs", there are links to Google Maps APIs, Google Maps Android API (which is highlighted with a yellow arrow and callout box), Google Maps SDK for iOS, Google Maps JavaScript API, Google Places API for Android, Google Places API for iOS, Google Maps Roads API, and "More". A yellow callout box points to the "Google Maps Android API" link with the text: "Select ‘Google Maps Android APIs’ link in APIs Library to enable them". The URL in the address bar is [https://console.developers.google.com/project/pk-edu-riu-4031-madmaps/apiui/apiview/maps\\_android\\_backend/overview](https://console.developers.google.com/project/pk-edu-riu-4031-madmaps/apiui/apiview/maps_android_backend/overview).

Select “**Google Maps Android APIs**” link in APIs Library to enable them

https://console.developers.google.com/project/pk-edu-riu-4031-madmaps/apiui/apiview/maps\_android\_backend/overview

# Google Developers Console

The screenshot shows the Google Developers Console interface. In the top navigation bar, there are tabs for 'Inbox (225) - muhammad.' and 'Google Maps Android API'. The URL in the address bar is <https://console.developers.google.com/project/pk-edu-riu-4031-madmaps/apiui/apis>. The main content area displays the 'Google Maps Android API v2' page, which is described as 'An Android library for embedding a native Google Map in an Android app.' A blue 'Enable API' button is prominently displayed. On the left sidebar, under the 'APIs & auth' section, the 'APIs' option is selected, showing a list of other available APIs: Overview, Permissions, APIs & auth, APIs, Credentials, Consent screen, Push, Monitoring, Source Code, Deploy & Manage, Compute, Networking, and Storage. A yellow callout box with a black border and white text points to the 'Enable API' button, containing the instruction: 'Click "Enable API" button to enable'.

Google Developers Console

Mad Maps

Sign up for a free trial.

muhammad.saud@riu.edu.pk

Overview

Permissions

APIs & auth

APIs

Credentials

Consent screen

Push

Monitoring

Source Code

Deploy & Manage

Compute

Networking

Storage

Enable API

Google Maps Android API v2

An Android library for embedding a native Google Map in an Android app.

Learn more

Click "Enable API" button to enable

# Google Developers Console

The screenshot shows a web browser window with the URL <https://console.developers.google.com/project/pk-edu-riu-4031-madmaps/apiui/apis>. The title bar includes tabs for 'Inbox (225) - muhammad.' and 'Google Maps Android API'. The main content area is titled 'Google Developers Console' and shows the 'Google Maps Android API v2' overview. A yellow callout box points from the left sidebar to the central text: 'After enabling the APIs you can get API Key from "Credentials"'.

In the sidebar, under 'APIs & auth', the 'APIs' link is selected. Other options include Overview, Permissions, Credentials, Consent screen, Push, Monitoring, Source Code, Deploy & Manage, Compute, Networking, and Storage.

The central content area displays the following text:

Google Maps Android API v2

Overview

An Android library for embedding a native Google Map in an Android app.

Learn more

[View reports in API Console](#)

After enabling the APIs you can get API Key from  
"Credentials"

# Google Developers Console

- Follow these steps to create or modify a project for your application in the Google Developers Console and enable the Google Maps Android API.
  - Go to the Google Developers Console: <https://console.developers.google.com/>
  - Select a project, or create a new one.
  - In the sidebar on the left, **expand APIs & auth.**
    - Next, **click APIs**.
    - Select the Enabled APIs link in the API section to see a list of all your enabled APIs. Make sure that the API is on the list of enabled APIs. If you have not enabled it, select the API from the list of APIs, then select the **Enable API button for the API**.
    - The only API you need is the Google Maps Android API, although you can choose to enable other APIs for the same project too.
  - In the sidebar on the left, **select Credentials**.

# Google Developers Console

The screenshot shows the Google Developers Console interface. The left sidebar lists various services: Overview, Permissions, APIs & auth, APIs, Credentials (which is selected and highlighted in blue), Consent screen, Push, Monitoring, Source Code, Deploy & Manage, Compute, Networking, and Storage. The main content area is titled "Credentials - Mad Maps". It features a "Create new Client ID" button and a "Public API access" section with the message "No keys found." Below this is a detailed description of what Public API access means. A yellow callout bubble points to the "Create new Key" button at the bottom of the main content area. The URL in the browser bar is <https://console.developers.google.com/project/pk-edu-riu-4031-madmaps/apiui/cr>.

Google Developers Console

Mad Maps

Sign up for a free trial.

muhammad.saud@riu.edu.pk

Overview

Permissions

APIs & auth

APIs

Credentials

Consent screen

Push

Monitoring

Source Code

Deploy & Manage

Compute

Networking

Storage

Inbox (225) - muhammad.saud@riu.edu.pk

Credentials - Mad Maps

<https://console.developers.google.com/project/pk-edu-riu-4031-madmaps/apiui/cr>

Learn more

Create new Client ID

Public API access

No keys found.

Use of this key does not require any user action or consent, does not grant access to any account information, and is not used for authorization.

Learn more

Create new Key

Scroll down to “Public API access” in Credentials and click “Create new Key”

# Google Developers Console

The screenshot shows the Google Developers Console interface. On the left, there is a sidebar with various project settings like Overview, Permissions, APIs & auth, APIs, Credentials, Consent screen, Push, Monitoring, Source Code, Deploy & Manage, Compute, Networking, and Storage. The 'Credentials' option is currently selected. In the main area, there is a sub-menu titled 'Create new Client ID'. A modal window titled 'Create a new key' is open. Inside the modal, it says: 'The APIs represented in the Google Developers Console require that requests include a unique project identifier. This enables the Console to tie a request to a specific project in order to monitor traffic, enforce quotas, and handle billing.' Below this text are four buttons: 'Server key', 'Browser key', 'Android key', and 'iOS key'. The 'Android key' button is highlighted with a yellow arrow pointing to it from the bottom right. At the bottom of the modal, there is a note: 'Information, and is not used for authorization.' followed by a 'Learn more' link. At the very bottom of the modal is a blue 'Create new Key' button.

Click “Android Key” button

# Google Developers Console

The screenshot shows a browser window with two tabs: "Inbox (225) - muhammad." and "Credentials - Mad Maps". The main content area is titled "Credentials - Mad Maps" and shows the configuration for a new API key. A yellow callout box points to the "SHA-1 fingerprint" input field, which contains the placeholder text "Enter your app's SHA-1 fingerprint, then a semicolon (;) then your app's package name and press 'Create'".

This key can be deployed in your Android application.

API requests are sent directly to Google from your client Android device. Google verifies that each request originates from an Android application that matches one of the certificate SHA1 fingerprints and package names listed below. You can discover the SHA1 fingerprint of your developer certificate using the following command:

```
keytool -list -v -keystore mystore.keystore
```

[Learn more](#)

Accept requests from an Android application with one of the certificate fingerprints and package names listed below (Optional)

One SHA1 certificate fingerprint and package name (separated by a semicolon) per line. Example:  
45:B5:E4:6F:36:AD:0A:98:94:B4:02:66:2B:12:17:F2:56:26:A0:E0;com.example

Or if you leave this blank, requests will be accepted from any Android application. Be sure to add SHA1 certificates before using this key in production.

**Create** **Cancel**

# Google Developers Console

The screenshot shows the Google Developers Console interface. On the left, there is a sidebar with various options: Overview, Permissions, APIs & auth (selected), APIs, Credentials (highlighted with a yellow box), Consent screen, Push, Monitoring, Source Code, Deploy & Manage, Compute, Networking, and Storage. The main area is titled "Create new Client ID" and shows "Public API access" information. It includes fields for "API key" (containing a long string of characters), "Android applications" (with a placeholder), "Activation date" (May 23, 2015, 7:37:00 PM), and "Activated by" (muhammad.saud@riu.edu.pk). Below these fields are buttons for "Edit allowed Android applications", "Regenerate key", and "Delete". A yellow callout box points to the "Credentials" option in the sidebar and contains the text: "You have successfully created a forty-character **API key** for your android application".

# Google Developers Console

- Follow these steps to create or modify a project for your application in the Google Developers Console and enable the Google Maps Android API.
  - Go to the Google Developers Console: <https://console.developers.google.com/>
  - Select a project, or create a new one.
  - In the sidebar on the left, **expand APIs & auth.**
    - Next, **click APIs**.
    - Select the Enabled APIs link in the API section to see a list of all your enabled APIs. Make sure that the API is on the list of enabled APIs. If you have not enabled it, select the API from the list of APIs, then select the **Enable API button for the API**.
    - The only API you need is the Google Maps Android API, although you can choose to enable other APIs for the same project too.
  - In the sidebar on the left, **select Credentials**.

# Create Your App

- Obtaining a Google Maps API Key for your application requires the following steps:
  - Retrieve information about your **Application's Certificate**. 
  - **Register a project in the Google Developers Console** and add the Maps API as a service for the project. 
  - **Request Google Maps API Key** for your project. 

# Google Maps

- Set Up
  - Install **Google Play Services SDK & Google APIs** 
  - Create a **Virtual Device** that uses “Google APIs” Platform 
- Create Your App
  - Get an **App Certificate** and the **Google Maps API Key** 
  - **Add Google Play Services to Your Project**
  - Specify app settings in the **AndroidManifest.xml**
  - Add a **Map**

# Add Google Play Services

- To make the Google Play services APIs available to your app:
  - Make sure your **minSdkVersion** is **17**
  - Open the **build.gradle (Module: app)** file inside your application module directory.
  - Add a **new build rule under dependencies** for the play-services.

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    compile 'com.android.support:appcompat-v7:22.1.1'  
    compile 'com.google.android.gms:play-services-maps:6.5.87'  
}
```
  - Save the changes and click **Sync Project** with Gradle Files.
  - You can now begin developing features with the Google Play services APIs.

# Google Maps

- Set Up
  - Install **Google Play Services SDK & Google APIs** 
  - Create a **Virtual Device** that uses “Google APIs” Platform 
- Create Your App
  - Get an **App Certificate** and the **Google Maps API Key** 
  - Add **Google Play Services to Your Project** 
  - Specify app settings in the **AndroidManifest.xml**
  - Add a **Map**

# AndroidManifest.xml

- An Android application that uses the Google Maps Android API should specify the following settings in its manifest file, `AndroidManifest.xml`:
  - A reference to the Google Play services version.
  - The Maps API key for the application. The key confirms that you've registered with the Google Maps service via the Google Developers Console.
  - Permissions that give the application access to Android system features and to the Google Maps servers.
  - Notification that the application requires OpenGL ES version 2. External services can detect this notification and act accordingly. (Recommended)

# AndroidManifest.xml

- **Add the Google Play Services Version**

Add the following declaration within the <application> element. This **embeds the version of Google Play services** that the app was compiled with.

```
<meta-data  
    android:name="com.google.android.gms.version"  
    android:value="@integer/google_play_services_version" />
```

# AndroidManifest.xml

- **Add the API Key**

In AndroidManifest.xml, add the following element as a child of the <application> element, by inserting it just before the closing tag </application>:

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

Substitute your API key for YOU\_API\_KEY in the value attribute. This element sets the key com.google.android.geo.API\_KEY to the value of your API key, and makes the API key visible to any **MapFragment** in your application.

# AndroidManifest.xml

- **Specify Permissions**

Specify the permissions your application needs, by adding `<uses-permission>` elements as children of the `<manifest>` element:

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

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

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

<!-- The following two permissions are not required to use
     Google Maps Android API v2, but are recommended. -->

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

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

# AndroidManifest.xml

- **Specify Requirement for OpenGL ES Version 2**

The Google Maps Android API uses OpenGL ES version 2 to render the map. If OpenGL ES version 2 is not installed, your map will not appear.

Google recommends that you add the following <uses-feature> element as a child of the <manifest> element in AndroidManifest.xml::

```
<uses-feature  
    android:glEsVersion="0x00020000"  
    android:required="true" />
```

# Google Maps

- Set Up
  - Install **Google Play Services SDK & Google APIs** 
  - Create a **Virtual Device** that uses “Google APIs” Platform 
- Create Your App
  - Get an **App Certificate** and the **Google Maps API Key** 
  - Add **Google Play Services to Your Project** 
  - Specify app settings in the **AndroidManifest.xml** 
  - Add a **Map**

# Add Map

- The basic steps for adding a map are:
  - Add a **Fragment object** to the Activity that will handle the map. The easiest way to do this is to add a <fragment> element to the layout file for the Activity.
  - Implement the **OnMapReadyCallback** interface and use the **onMapReady(GoogleMap)** callback method to get a handle to the GoogleMap object. The GoogleMap object is the internal representation of the map itself. To set the view options for a map, you modify its **GoogleMap object**.
  - Call **getMapAsync()** on the fragment to register the callback.

# Add Map

- **Add Fragment Object**

Add a <fragment> element to the activity's layout file to define a Fragment object. In this element, set the android:name attribute to "com.google.android.gms.maps.MapFragment". This automatically attaches a MapFragment to the activity.

```
<fragment  
    android:layout_width="match_parent"  
    android:layout_height="0dp"  
    android:layout_weight="1"  
    android:name="com.google.android.gms.maps.MapFragment"  
    android:id="@+id/map" />
```

# Add Map

- **Add Map Code**

To work with the map inside your app, you'll need to implement the **OnMapReadyCallback** interface and set an instance of the callback on a MapFragment.

```
public class MainActivity extends Activity implements OnMapReadyCallback {  
    ...  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
    }  
    ...  
}
```

# Add Map

- **Add Map Code**

Get a handle to the fragment by calling **FragmentManager.findFragmentById()**, passing it the resource ID of your <fragment> element.

```
public class MainActivity extends Activity implements OnMapReadyCallback {  
    ...  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        MapFragment mapFragment= (MapFragment)  
            getFragmentManager().findFragmentById(R.id.map);  
  
        mapFragment.getMapAsync(this);  
    }  
    ...  
}
```

# Add Map

- **Add Map Code**

Use the **onMapReady(GoogleMap)** callback method to get a handle to the GoogleMap object. The callback is triggered when the map is ready to be used. You can use the GoogleMap object to set various options for the map.

```
public class MainActivity extends Activity implements OnMapReadyCallback {  
    ...  
    @Override  
    public void onMapReady(GoogleMap googleMap) {  
  
    }  
    ...  
}
```

# Add Map

- **Add Map Code**

Use the **onMapReady(GoogleMap)** callback method to get a handle to the GoogleMap object. The callback is triggered when the map is ready to be used. You can use the GoogleMap object to set various options for the map.

```
public class MainActivity extends Activity implements OnMapReadyCallback {  
    ...  
    @Override  
    public void onMapReady(GoogleMap googleMap) {  
        googleMap.addMarker(new MarkerOptions()  
            .position(new LatLng(33.6667, 73.1667))  
            .title("Main Campus"));  
    }  
    ...  
}
```

# Google Maps

- Set Up
  - Install **Google Play Services SDK & Google APIs** 
  - Create a **Virtual Device** that uses “Google APIs” Platform 
- Create Your App
  - Get an **App Certificate** and the **Google Maps API Key** 
  - Add **Google Play Services to Your Project** 
  - Specify app settings in the **AndroidManifest.xml** 
  - Add a **Map** 

# **GOOGLE MAP OBJECT**

# GoogleMap Object

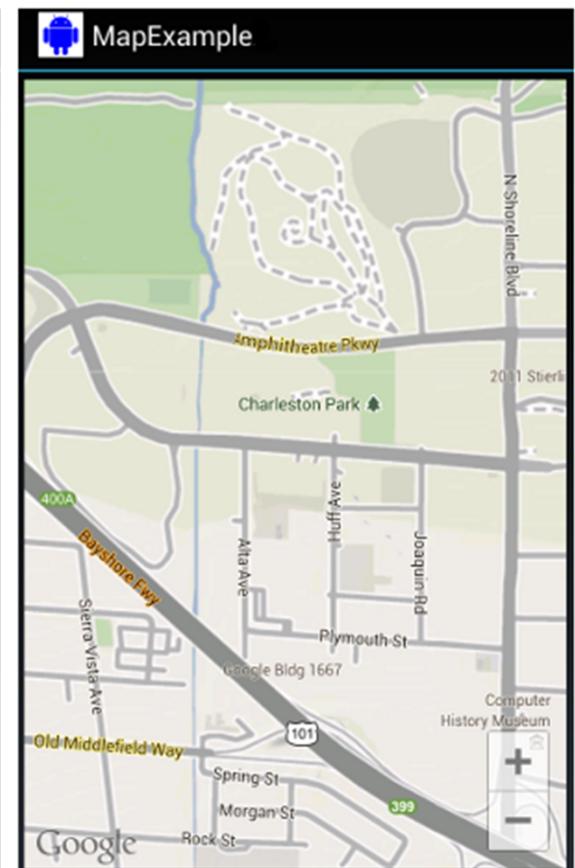
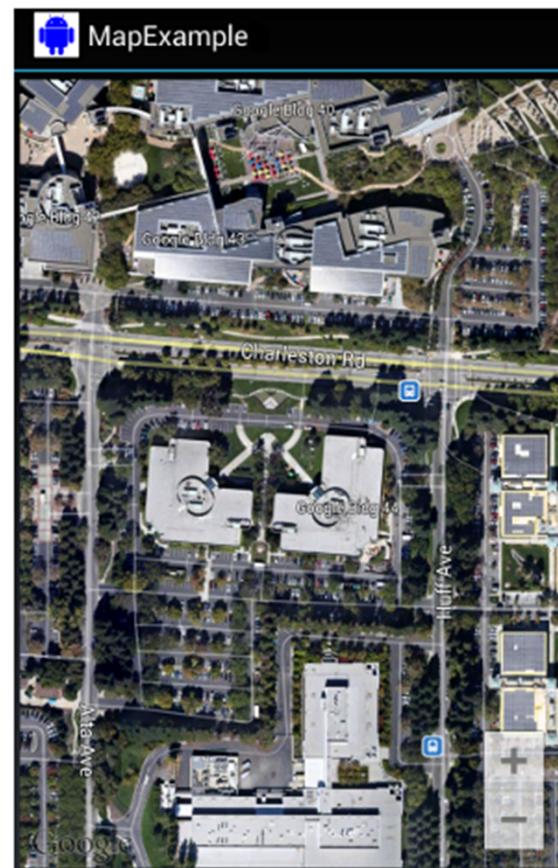
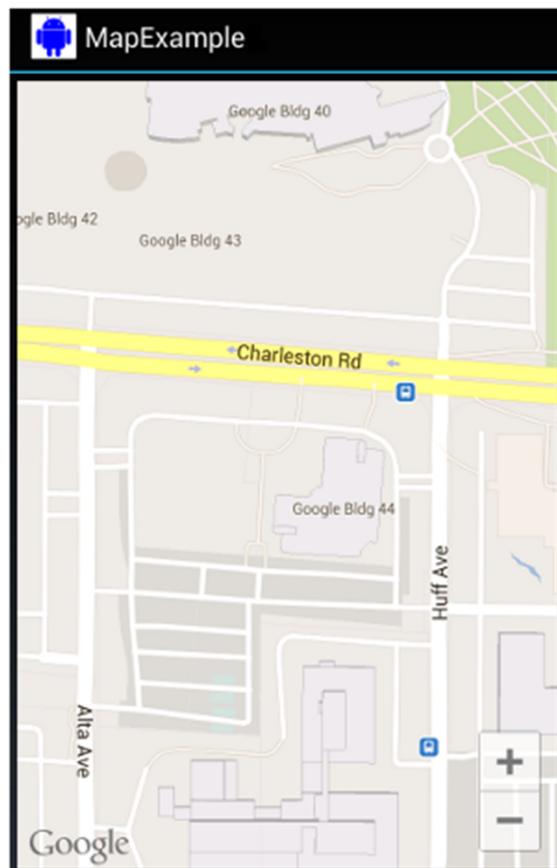
- The key class when working with a map object is the GoogleMap class. GoogleMap models the map object within your application. GoogleMap handles the following operations automatically:
  - Connecting to the Google Maps service.
  - Downloading map tiles.
  - Displaying tiles on the device screen.
  - Displaying various controls such as pan and zoom.
  - Responding to pan and zoom gestures by moving the map and zooming in or out.
- In addition to these automatic operations, you can control the behavior of maps with objects and methods of the API.

# Map Types

- The Google Maps Android API offers four types of maps, as well as an option to have no map at all:
  - **Normal:** Typical road map. Roads, some man-made features, and important natural features such as rivers are shown. 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.

# Map Types

Comparison of normal, hybrid and terrain maps



# Map Types

- **Change the Map Type**

To set the type of a map, call the `GoogleMap` object's `setMapType()` method, passing one of the type constants defined in `GoogleMap`.

```
public void onMapReady(GoogleMap googleMap) {  
    googleMap.addMarker(new MarkerOptions()  
        .position(new LatLng(33.6667, 73.1667))  
        .title("Main Campus"));  
  
    googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);  
}
```

# Customizing Markers

- Markers indicate single locations on the map. You can customize your markers by changing the default color, or replacing the marker icon with a custom image.

```
public void onMapReady(GoogleMap googleMap) {  
    googleMap.addMarker(new MarkerOptions()  
        .position(new LatLng(33.6667, 73.1667))  
        .title("Main Campus"));  
}
```

# Customizing Markers

- **Customize the Marker Color:**

```
public void onMapReady(GoogleMap googleMap) {  
    googleMap.addMarker(new MarkerOptions()  
        .position(new LatLng(33.6667, 73.1667))  
  
        .icon(BitmapDescriptorFactory  
            .defaultMarker(BitmapDescriptorFactory  
                .HUE_BLUE))  
  
        .title("Main Campus"));  
}
```

# Customizing Markers

- **Customize the Marker Opacity:**

```
public void onMapReady(GoogleMap googleMap) {  
    googleMap.addMarker(new MarkerOptions()  
        .position(new LatLng(33.6667, 73.1667))  
  
        .alpha(0.5f)  
  
        .title("Main Campus"));  
}
```

# Customizing Markers

- **Customize the Marker Image:**

```
public void onMapReady(GoogleMap googleMap) {  
    googleMap.addMarker(new MarkerOptions()  
        .position(new LatLng(33.6667, 73.1667))  
  
        .icon(BitmapDescriptorFactory  
            .fromResource(R.drawable.arrow))  
  
        .title("Main Campus"));  
}
```

# Configure Initial State

- The Maps API allows you to configure the initial state of the map to suit your application's needs. For example you can specify:
  - **The camera position**, including: location, zoom, bearing and tilt.
  - **The map type**.
  - Whether the **zoom buttons and/or compass** appear on screen.
  - The **gestures** a user can use to manipulate the camera.

# Configure Initial State

- The Maps API defines a set of custom XML attributes for a MapFragment that you can use to configure the initial state of the map directly from the layout file.
  - **mapType**. This allows you to specify the type of map to display. Valid values include: none, normal, hybrid, satellite and terrain.
  - **cameraTargetLat**, **cameraTargetLng**, **cameraZoom**, **cameraBearing**, **cameraTilt**. These allow you to specify the initial camera position.
  - **uiZoomControls**, **uiCompass**. These allow you to specify whether you want the zoom controls and compass to appear on the map.
  - **uiZoomGestures**, **uiScrollGestures**, **uiRotateGestures**, **uiTiltGestures**. These allow you to specify which gestures are enabled/disabled for interaction with the map.

# Configure Initial State

- In order to use these custom attributes within your XML layout file, **you must first add the following namespace declaration.** You can choose any namespace, it doesn't have to be map:

```
xmlns:map="http://schemas.android.com/apk/res-auto"
```

- You can then add the attributes with a **map: prefix** into your layout components, as you would with standard Android attributes.

# Configure Initial State

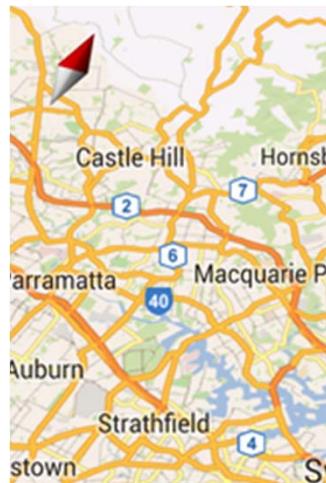
```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:id="@+id/map"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    map:cameraBearing="0"
    map:cameraTargetLat="33.6667"
    map:cameraTargetLng="73.1667"
    map:cameraTilt="30"
    map:cameraZoom="5"
    map:mapType="normal"
    map:uiCompass="false"
    map:uiRotateGestures="true"
    map:uiScrollGestures="false"
    map:uiTiltGestures="true"
    map:uiZoomControls="false"
    map:uiZoomGestures="true" />
```

# Interactively Changing UI Controls

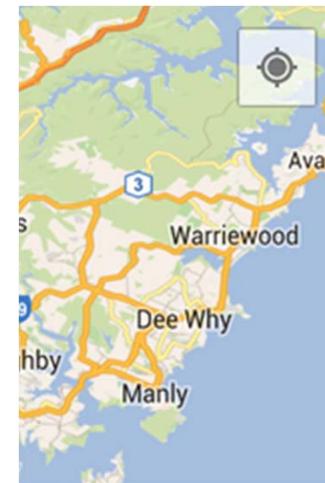
**ZoomControls**



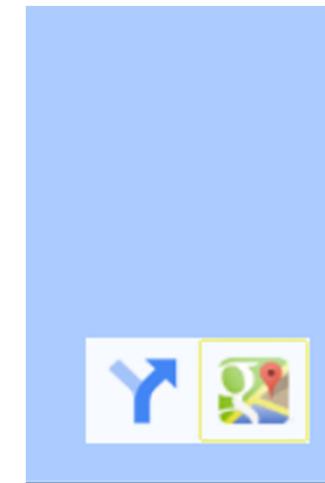
**Compass**



**MyLocation**



**MapToolbar**



# Interactively Changing UI Controls

- You can toggle the visibility of various controls in your map using the **UiSettings** class which can be obtained from a GoogleMap with the **GoogleMap.getUiSettings** method:

```
MapFragment mapFragment;
. . .
public void someMethod (View v) {
    GoogleMap gMap=mapFragment.getMap();

    gMap.getUiSettings().setZoomControlsEnabled(true);

}
```

# Interactively Changing UI Controls

- You can toggle the visibility of various controls in your map using the **UiSettings** class which can be obtained from a GoogleMap with the **GoogleMap.getUiSettings** method:
  - `UiSettings.setZoomControlsEnabled(boolean)`
  - `UiSettings.setCompassEnabled(boolean)`
  - `UiSettings.setMyLocationButtonEnabled(boolean)`
  - `UiSettings.setMapToolbarEnabled(boolean)`
  - `UiSettings.setZoomGesturesEnabled(boolean)`
  - `UiSettings.setScrollGesturesEnabled(boolean)`
  - `UiSettings.setTiltGesturesEnabled(boolean)`
  - `UiSettings.setRotateGesturesEnabled(boolean)`

# Map Events

- If you want to respond to a user tapping on a point on the map, you can use an **OnMapClickListener** which you can set on the map by calling **GoogleMap.setOnMapClickListener(OnMapClickListener)**.
- When a user clicks (taps) somewhere on the map, you will receive an **onMapClick(LatLng)** event that indicates the location on the map that the user clicked.

# Map Events

- You can also listen for long click events with an **OnMapLongClickListener** which you can set on the map by calling **GoogleMap.setOnMapLongClickListener(OnMapLongClickListener)**.
- This listener behaves similarly to the click listener and will be notified on long click events with an **onMapLongClick(LatLng)** callback.

# Map Events

```
public void onMapReady(GoogleMap googleMap) {  
    . . .  
    googleMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {  
        @Override  
        public void onMapClick(LatLng latLng) {  
  
            String location = String.valueOf(latLng.latitude)  
                + ", " +String.valueOf(latLng.longitude);  
  
            Toast.makeText(getApplicationContext(),  
                "Your Location: " +  
                location,Toast.LENGTH_SHORT).show();  
        }  
    });  
}
```

# Camera

- 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: target (latitude/longitude location), zoom, bearing and tilt.

# Camera

```
MapFragment mapFragment;
private static final LatLng A = new LatLng(33.616492, 72.971892);
private static final LatLng B = new LatLng(33.5980297, 73.0414803);
private static final LatLng C = new LatLng(33.7332804,73.0884844);
. . .
public void gotoC (View v) {
    GoogleMap gMap=mapFragment.getMap();

    CameraPosition cameraPosition = new CameraPosition.Builder()
        .target(C)
        .zoom(15)
        .bearing(90)
        .tilt(30)
        .build();

    gMap.animateCamera(CameraUpdateFactory
        .newCameraPosition(cameraPosition));
}
```

# Camera

```
MapFragment mapFragment;
private static final LatLng A = new LatLng(33.616492, 72.971892);
private static final LatLng B = new LatLng(33.5980297, 73.0414803);
private static final LatLng C = new LatLng(33.7332804,73.0884844);
. . .
public void gotoC (View v) {
    GoogleMap gMap=mapFragment.getMap();

    CameraPosition cameraPosition = new CameraPosition.Builder()
        .target(B)
        .zoom(15)
        .build();

    gMap.moveCamera(CameraUpdateFactory
        .newCameraPosition(cameraPosition));
}
```

# References

- <https://developers.google.com/maps/documentation/android/start>
- <http://developer.android.com/google/play-services/maps.html>
- <http://docs.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html>
- <https://developer.android.com/google/play-services/setup.html>
- <http://stackoverflow.com/questions/29425015/this-app-wont-run-unless-you-update-google-play-services-when-app-is-installe>
- <https://developer.android.com/tools/publishing/app-signing.html>
- <https://developer.android.com/reference/com/google/android/gms/maps/MapFragment.html>
- <https://developer.android.com/training/location/index.html>
- <https://developers.google.com/console/help/new/#apikeybestpractices>

Q & A