Android Google Map

Android provides facility to integrate Google map in our application. Google map displays your current location, navigate location direction, search location etc.

Customizing Google Map

You can easily customize google map from its default view , and change it according to your demand.

Types of Google Maps

There are four different types of Google maps, as well as an optional to no map at all. Each of them gives different view on map. These maps are as follow:

1. **Normal:** This type of map displays typical road map, natural features like river and some features build by humans.
2. **Hybrid:** This type of map displays satellite photograph data with typical road maps. It also displays road and feature labels.
3. **Satellite:** Satellite type displays satellite photograph data, but doesn't display road and feature labels.
4. **Terrain:** This type displays photographic data. This includes colors, contour lines and labels and perspective shading.
5. **None:** This type displays an empty grid with no tiles loaded.

Syntax of different types of map

1. googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
2. googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
3. googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
4. googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);

Google Map - Layout file

syntax to add the map fragment into xml layout file is given below

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

Google Map - AndroidManifest file

The next thing you need to do is to add some permissions along with the Google Map API key in the AndroidManifest.XML file.

Its syntax is given below −

```
<!--Permissions-->

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="com.google.android.providers.gsf.permission.
   READ_GSERVICES" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<!--Google MAP API key-->

<meta-data
   android:name="com.google.android.maps.v2.API_KEY"
   android:value="AIzaSyDKymeBXNeiFWY5jRUejv6zItpmr2MVyQ0" />
```

Adding Marker

You can place a maker with some text over it displaying your location on the map. It can be done by via **addMarker()** method. Its syntax is given below −

```
final LatLng TutorialsPoint = new LatLng(21 , 57);
Marker TP = googleMap.addMarker(new MarkerOptions()
   .position(TutorialsPoint).title("TutorialsPoint"));
```

Enable/Disable zoom

You can also enable or disable the zoom gestures in the map by calling the **setZoomControlsEnabled(boolean)** method.

Its syntax is given below −

```
googleMap.getUiSettings().setZoomGesturesEnabled(true);
```

zoomIn() or **zoomOut()** method are provided by android MapController class to zoom to a particular function programatically.

methods available in the **GoogleMap class** , to customize the map are listed below −

Sr.No	Method & description
1	**addCircle(CircleOptions options)** This method add a circle to the map
2	**addPolygon(PolygonOptions options)** This method add a polygon to the map
3	**addTileOverlay(TileOverlayOptions options)** This method add tile overlay to the map

4	**animateCamera(CameraUpdate update)** This method Moves the map according to the update with an animation
5	**clear()** This method removes everything from the map.
6	**getMyLocation()** This method returns the currently displayed user location.
7	**moveCamera(CameraUpdate update)** This method repositions the camera according to the instructions defined in the update
8	**setTrafficEnabled(boolean enabled)** This method Toggles the traffic layer on or off.
9	**snapshot(GoogleMap.SnapshotReadyCallback callback)** This method Takes a snapshot of the map
10	**stopAnimation()** This method stops the camera animation if there is one in progress

Navigating to a Specific Location

- **GeoPoint** object to represent a geographical location.
- For this class, the latitude and longitude of a location are represented in micro degrees.
- This means that they are stored as integer values.
- For a latitude value of 40.747778, for example, you need to multiply it by 1e6 (which is one million) to obtain 40747778.
- **To navigate the map** to a particular location, you can use the **animateTo()** method of the MapController class.
- The **setZoom()** method enables you to specify the zoom level at which the map is displayed (the bigger the number, the more details you see on the map).
- The **invalidate()** method forces the MapView to be redrawn

---

<u>Android Location-Based Services:</u>

- The applications like **Google Maps, Waze, MapQuest**, etc helps to rack the location of the device.

- They also provide services like finding nearby restaurants, hospitals, petrol pumps, etc. Even the cab drivers now use maps to locate their route.
- As technology is emerging, it becomes quite essential for an android developer to learn about location-based services.

What are Android Location-Based Services?

Location-Based Services(LBS) are present in Android to provide you with features like current location detection, display of nearby places, geofencing, etc. It fetches the location using your device's GPS, Wifi, or Cellular Networks.

To build an app with location-based services, you need to access the Google Play Services Module. After that, you need to use a framework called Location Framework, which has many methods, classes, and interfaces to make your task easier.

location providers in Android : There are 3 location providers in Android. They are:

gps –> (GPS, AGPS): Name of the GPS location provider. This provider determines location using satellites. Depending on conditions, this provider may take a while to return a location fix. Requires the permission

android.permission.ACCESS_FINE_LOCATION.

network –> (AGPS, CellID, WiFi MACID): Name of the network location provider. This provider determines location based on availability of cell tower and WiFi access points. Results are retrieved by means of a network lookup. Requires either of the permissions

android.permission.ACCESS_COARSE_LOCATION or android.permission.ACCESS_FINE_LOCATION.

passive –> (CellID, WiFi MACID): A special location provider for receiving locations without actually initiating a location fix. This provider can be used to passively receive location updates when other applications or services request them without actually requesting the locations yourself. This provider will return locations generated by other providers. Requires the permission

android.permission.ACCESS_FINE_LOCATION,

although if the GPS is not enabled this provider might only return coarse fixes. This is what Android calls these location providers, however, the underlying technologies to make this stuff work is mapped to the specific set of hardware and telco provided capabilities (network service). The best way is to use the "network" or "passive" provider first, and then fallback on "gps", and depending on the task, switch between providers. This covers all cases, and provides a lowest common denominator service (in the worst case) and great service (in the best case).

Components of Location-Based Services in Android

The classes and the interfaces present in the Location Framework acts as the essential components for LBS. Those components are as follows:

- **LocationManager Class –** It is used to get Location Service access from the system.
- **LocationListener Interface –** It receives updates from the Location Manager class.
- **LocationProvider –** It is the class that provides us with the location for our devices.
- **Location Class –** Its objects carry information about the location. The information includes latitude, longitude, accuracy, altitude, and speed.

The location objects carry the location of your device. The place is in the form of latitude and longitude. On the location object, you can apply the below methods. The below methods help you to get location and other information regarding the location.

1. float distanceTo(Location destination) - It gives the approximate distance between our current location and the destination location.

2. float getAccuracy() - It gives us the accuracy of our location in metres.

3. **double getAltitude()** - It gives us the altitude of our place above sea level.

4. **double getLatitude()** - It gives the latitude coordinate of our place in degrees.

5. **double getLongitude()-** It gives the latitude coordinate of our place in degrees.

6**. float getSpeed()** - It gives the speed of our location change.

7. **void setAccuracy(float accuracy) -** Using setAccuracy(), you can set your custom accuracy in metres.

8. **void setAltitude(double altitude)-** Using setAltitude(), you can set the altitude of your place from sea level in metres.

9. **void setBearing(float bearing) -** Using the setBearing() method, you can set location bearing in degrees.

10. **void setLatitude(double Latitude) -**You can even set your location to some other latitude using the setLatitude() method.

11. void setLongitude(double longitude) - *y*ou can even set your location to some other longitude using the setLongitude() method.

12. **void setSpeed(float speed)** - You can even set speed using the setSpeed() method.

13**. void reset()-** It is used to reset your set location.

14. **boolean hasAccuracy()** - It says whether or not the location is accurate.

15. **boolean hasAltitude()** - It says if the place has an altitude or not

16**. boolean hasSpeed()** - It is true if the place has a speed attached.

17. **boolean hasBearing()** - It returns true if the place has a bearing or not**.**

Location Quality of Service

Quality of Service(QoS) is enabled through the location request object. Location request object requests the proper and accurate location. You can find below the methods that help in the process.

- **setPriority(int priority)** – It allows us to mark the request with priorities. The higher priority request is executed first.
- **setInterval(long millisecond)** – It allows us to set the intervals on which we seek the location updates.
- **setExpirationDuration(long millisecond)** – It allows us to set the duration of the request after which it shall stop.
- **setExpirationTime(long millisecond)** – It allows us to decide the expiration time of our request.
- **setNumUpdates(int number)** – It allows us to set the number of updates we require for a particular place.

Getting the Location That Was Touched

After using Google Maps for a while, you may want to know the latitude and longitude of a location corresponding to the position on the screen that was just touched. Knowing this information is very useful, as you can determine a location's address, a process known as reverse geocoding
If you have added an overlay to the map, you can override the **onTouchEvent**() method within the MapOverlay class. This method is fired every time the user touches the map. The method has two parameters: **MotionEvent** and **MapView**. Using the MotionEvent parameter, you can determine whether the user has lifted his or her finger from the screen using the **getAction()** method. The **getProjection()** method returns a projection for converting between screen-pixel coordinates and latitude/longitude coordinates. The **fromPixels()** method then converts the screen coordinates into a **GeoPoint** object.
Example:

```
public boolean onTouchEvent(MotionEvent event, MapView mapView)
{
//---when user lifts his finger---
if (event.getAction() == 1) {
GeoPoint p = mapView.getProjection().fromPixels(
(int) event.getX(),
(int) event.getY());
Toast.makeText(getBaseContext(),
“Location: “+
p.getLatitudeE6() / 1E6 + “,” +
p.getLongitudeE6() /1E6 ,
Toast.LENGTH_SHORT).show();
}
return false;
}
```

Geocoding and Reverse Geocoding

- Geocoding refers to transforming street address or any address into latitude and longitude.
- Reverse Geocoding refers to transforming latitude and longitude into its corresponding street address.
- android **Geocoder class** is used for Geocoding as well as Reverse Geocoding.
- **Address class** helps in fetching the street address, locality, sub-locality, city, country, landmark etc. features of the location.
- Using the above two classes we’ll be fetching the current marker address on the Google Maps in our application.
- For achieving Geocode or Reverse Geocode you must first import the proper package.

  import android.location.Geocoder;
- The geocoding or reverse geocoding operation needs to be done on a separate thread and should never be used on the UI thread as it will cause the system to display an application Not Responding (ANR) dialog to the user.
- **getFromLocation**() -- Provides an array of Addresses that attempt to describe the area immediately surrounding the given latitude and longitude.
- Example

```
getFromLocation(double latitude, double longitude, int
maxResults, Geocoder.GeocodeListener listener)
```

Geocode, use the below code

```
Geocoder gc = new Geocoder(context);
if(gc.isPresent()){
List<Address> list = gc.getFromLocationName(“155 Park Theater, Palo
Alto, CA”, 1);
Address address = list.get(0);
double lat = address.getLatitude();
double lng = address.getLongitude();
}
```

Reverse Geocode, use the below code

```
Geocoder gc = new Geocoder(context);
if(gc.isPresent()){
List<address> list = gc.getFromLocation(37.42279, -122.08506,1);
Address address = list.get(0);
StringBuffer str = new StringBuffer();
str.append("Name: " + address.getLocality() + "\n");
str.append("Sub-Admin Ares: " + address.getSubAdminArea() + "\n");
str.append("Admin Area: " + address.getAdminArea() + "\n");
str.append("Country: " + address.getCountryName() + "\n");
str.append("Country Code: " + address.getCountryCode() + "\n");
String strAddress = str.toString();
}
```

GETTING LOCATION DATA

- In Android, location-based services are provided by the LocationManager class, located in the android .location package.
- Using the LocationManager class, your application can obtain periodic updates of the device's geographical locations, as well as fire an intent when it enters the proximity of a certain location.
- First obtained a reference to the LocationManager class using the **getSystemService()** method.

```
//---use the LocationManager class to obtain locations data---
lm = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
locationListener = new MyLocationListener();
```

- Next, you created an instance of the **MyLocationListener** class.
- The MyLocationListener class implements the LocationListener abstract class. You need to override four methods in this implementation:
  - ➢ **onLocationChanged**(Location location) — Called when the location has changed
  - ➢ **onProviderDisabled**(String provider) — Called when the provider is disabled by the user
  - ➢ **onProviderEnabled**(String provider) — Called when the provider is enabled by the user
  - ➢ **onStatusChanged**(String provider, int status, Bundle extras) — Called when the provider status changes

```
public void onLocationChanged(Location loc) {
if (loc != null) {
Toast.makeText(getBaseContext(),
“Location changed : Lat: “ + loc.getLatitude() +
“ Lng: “ + loc.getLongitude(),
Toast.LENGTH_SHORT).show();
p = new GeoPoint(
(int) (loc.getLatitude() * 1E6),
(int) (loc.getLongitude() * 1E6));
mc.animateTo(p);
mc.setZoom(18);
}
}
```

To be notified whenever there is a change in location, you needed to register a request for location changes so that your program can be notified periodically. This is done via the **requestLocationUpdates**() method. You did this in the onResume() method of the activity:

```
@Override
public void onResume() {
super.onResume();
//---request for location updates---
lm.requestLocationUpdates(
LocationManager.GPS_PROVIDER,
0,
0,
locationListener);
}
```

The **requestLocationUpdates**() method takes four arguments:

➢ provider — The name of the provider with which you register. In this case, you are using GPS to
obtain your geographical location data.

➢ minTime — The minimum time interval for notifi cations, in milliseconds. 0 indicates that you want
to be continually informed of location changes.

➢ minDistance — The minimum distance interval for notifi cations, in meters. 0 indicates that you
want to be continually informed of location changes.

➢ listener — An object whose onLocationChanged() method will be called for each location update

To use the network provider, you need to add the ACCESS_COARSE_LOCATION permission to the AndroidManifest.xml file:

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

MONITORING A LOCATION

- One very cool feature of the LocationManager class is its ability to monitor a specific location. This is achieved using the addProximityAlert() method.
- The addProximityAlert() method takes fi ve arguments: latitude, longitude, radius (in meters),expiration and the pending intent.
- Example

```
lm.addProximityAlert(37.422006, -122.084095, 5, -1, pendingIntent);
```

s