



LOCATION & MAPS

TODAY'S TOPICS

LOCATION

LOCATION SUPPORT CLASSES

MAPS

MAP SUPPORT CLASSES

LOCATION SERVICES

MOBILE APPLICATIONS CAN BENEFIT FROM
BEING LOCATION-AWARE

ALLOW APPLICATIONS TO DETERMINE &
MANIPULATE LOCATION

USING LOCATION INFORMATION

FIND STORES NEAR THE USER'S CURRENT LOCATION

DIRECT A USER FROM A CURRENT TO A PARTICULAR STORE

DEFINE A GEOFENCE

INITIATE ACTION WHEN USER ENTERS OR EXITS THE GEOFENCE

LOCATION

REPRESENTS A POSITION ON THE EARTH

A LOCATION INSTANCE CONSISTS OF:

LATITUDE, LONGITUDE, TIMESTAMP AND,
OPTIONALLY, ACCURACY, ALTITUDE, SPEED,
AND BEARING

LOCATION PROVIDER

REPRESENTS A LOCATION DATA SOURCE

ACTUAL DATA MAY COME FROM

GPS SATELLITES

CELL PHONE TOWERS

WIFI ACCESS POINTS

LOCATION PROVIDER TYPES

NETWORK – WIFI AND CELL TOWER

GPS – SATELLITE

PASSIVE – PIGGYBACK ON THE READINGS
REQUESTED BY OTHER APPLICATIONS

NETWORK PROVIDER

DETERMINES LOCATION BASED ON CELL TOWER
AND WIFI ACCESS POINTS

REQUIRES EITHER

`android.permission.`

`ACCESS_COARSE_LOCATION`

`android.permission.`

`ACCESS_FINE_LOCATION`

GPS PROVIDER

DETERMINES LOCATION USING SATELLITES

REQUIRES

`android.permission.`

`ACCESS_FINE_LOCATION`

PASSIVE PROVIDER

RETURNS LOCATIONS GENERATED BY OTHER PROVIDERS

REQUIRES

android.permission.

ACCESS_FINE_LOCATION

LOCATION PROVIDER

DIFFERENT LOCATION PROVIDERS OFFER
DIFFERENT TRADEOFFS BETWEEN COST,
ACCURACY, AVAILABILITY & TIMELINESS

PROVIDER TRADEOFFS

GPS – EXPENSIVE, ACCURATE, SLOWER,
AVAILABLE OUTDOORS

NETWORK – CHEAPER, LESS ACCURATE,
FASTER, AVAILABILITY VARIES

PASSIVE – CHEAPEST, FASTEST, NOT
ALWAYS AVAILABLE

LOCATIONMANAGER

SYSTEM SERVICE FOR ACCESSING LOCATION
DATA

```
getSystemService(  
    Context.LOCATION_SERVICE)
```

LOCATIONMANAGER

DETERMINE THE LAST KNOWN USER LOCATION

REGISTER FOR LOCATION UPDATES

REGISTER TO RECEIVE INTENTS WHEN THE
DEVICE NEARS OR MOVES AWAY FROM A
GIVEN GEOGRAPHIC AREA

LOCATIONLISTENER

DEFINES CALLBACK METHODS THAT ARE
CALLED WHEN LOCATION OR
LOCATIONPROVIDER STATUS CHANGES

LOCATIONLISTENER

```
void onLocationChanged(  
    Location location)
```

```
void onProviderDisabled(  
    String provider)
```

```
void onProviderEnabled(  
    String provider)
```

```
void onStatusChanged(  
    String provider,  
    int status,  
    Bundle extras)
```

OBTAINING LOCATION

START LISTENING FOR UPDATES FROM
LOCATION PROVIDERS

MAINTAIN A "CURRENT BEST ESTIMATE" OF
LOCATION

WHEN ESTIMATE IS "GOOD ENOUGH", STOP
LISTENING FOR LOCATION UPDATES

USE BEST LOCATION ESTIMATE

DETERMINING BEST LOCATION

SEVERAL FACTORS TO CONSIDER

MEASUREMENT TIME

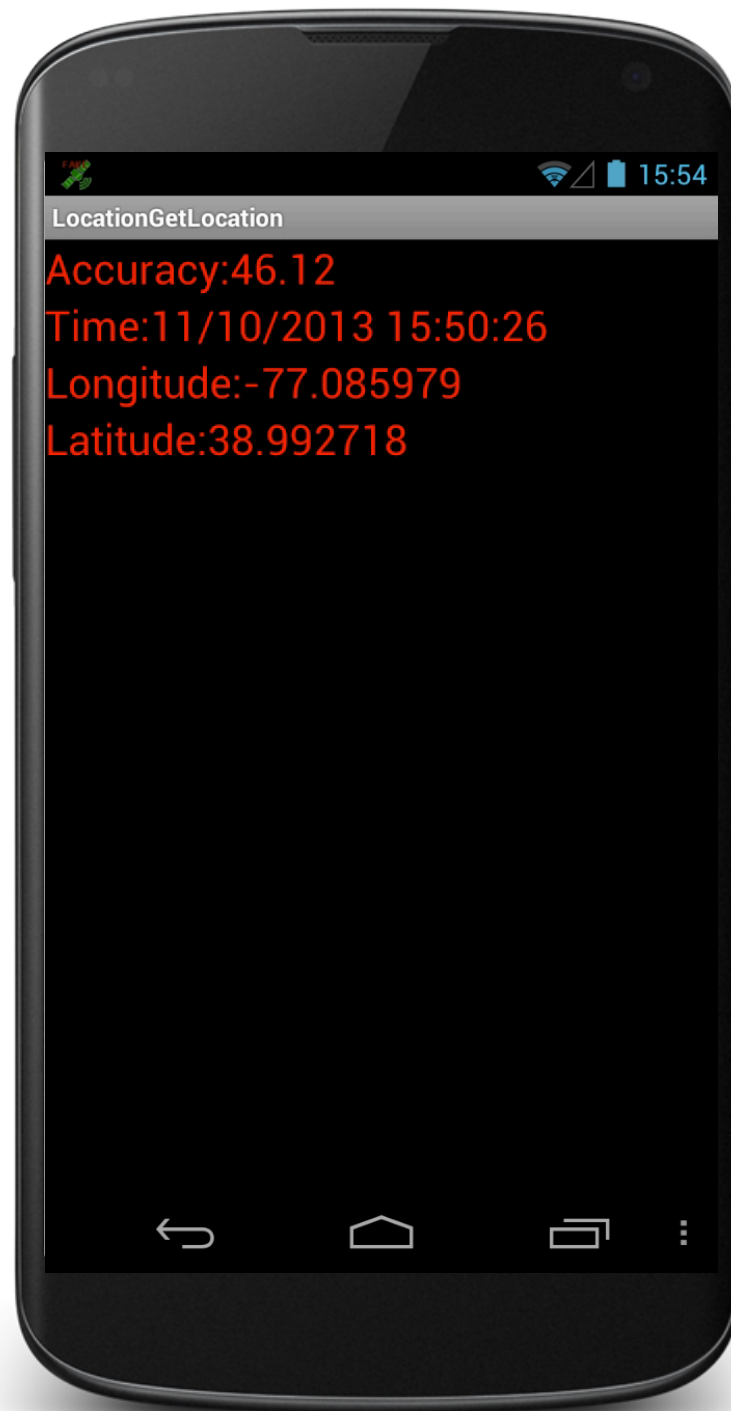
ACCURACY

POWER NEEDS

LOCATIONGETLOCATION

APPLICATION ACQUIRES AND DISPLAYS
THE LAST KNOWN LOCATIONS FROM ALL
PROVIDERS

IF NECESSARY, ACQUIRES AND DISPLAYS
NEW READINGS FROM ALL PROVIDERS



LOCATIONGETLOCATION

```
// Acquire reference to the LocationManager
if (null == (mLocationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE)))
    finish();

// Get best last location measurement
mBestReading = bestLastKnownLocation(MIN_LAST_READ_ACCURACY, FIVE_MIN);

// Display last reading information
if (null != mBestReading) {

    updateDisplay(mBestReading);

} else {

    mAccuracyView.setText("No Initial Reading Available");

}
```

LOCATIONGETLOCATION

```
mLocationListener = new LocationListener() {  
    // Called back when location changes  
  
    public void onLocationChanged(Location location) {  
        ensureColor();  
  
        // Determine whether new location is better than current best  
        // estimate  
  
        if (null == mBestReading  
            || location.getAccuracy() < mBestReading.getAccuracy()) {  
  
            // Update best estimate  
            mBestReading = location;  
  
            // Update display  
            updateDisplay(location);  
  
            if (mBestReading.getAccuracy() < MIN_ACCURACY)  
                mLocationManager.removeUpdates(mLocationListener);  
        }  
    }  
}
```

LOCATIONGETLOCATION

```
@Override
protected void onResume() {
    super.onResume();

    // Determine whether initial reading is
    // "good enough". If not, register for
    // further location updates

    if (null == mBestReading
        || mBestReading.getAccuracy() > MIN_LAST_READ_ACCURACY
        || mBestReading.getTime() < System.currentTimeMillis()
            - TWO_MIN) {

        // Register for network location updates
        if (null != mLocationManager
            .getProvider(LocationManager.NETWORK_PROVIDER)) {
            mLocationManager.requestLocationUpdates(
                LocationManager.NETWORK_PROVIDER, POLLING_FREQ,
                MIN_DISTANCE, mLocationListener);
        }
    }
}
```


LOCATIONGETLOCATION

```
// Register for GPS location updates
if (null != locationManager
    .getProvider(LocationManager.GPS_PROVIDER)) {
    locationManager.requestLocationUpdates(
        LocationManager.GPS_PROVIDER, POLLING_FREQ,
        MIN_DISTANCE, mListener);
}

// Schedule a runnable to unregister location listeners
Executors.newScheduledThreadPool(1).schedule(new Runnable() {

    @Override
    public void run() {

        Log.i(TAG, "location updates cancelled");

        locationManager.removeUpdates(mListener);
    }
}, MEASURE_TIME, TimeUnit.MILLISECONDS);
}
```

LOCATIONGETLOCATION

```
// Get the last known location from all providers
// return best reading that is as accurate as minAccuracy and
// was taken no longer then minAge milliseconds ago. If none,
// return null.

private Location bestLastKnownLocation(float minAccuracy, long maxAge) {

    Location bestResult = null;
    float bestAccuracy = Float.MAX_VALUE;
    long bestAge = Long.MIN_VALUE;

    List<String> matchingProviders = mLocationManager.getAllProviders();

    for (String provider : matchingProviders) {

        Location location = mLocationManager.getLastKnownLocation(provider);
```

LOCATIONGETLOCATION

```
    if (location != null) {  
        float accuracy = location.getAccuracy();  
        long time = location.getTime();  
  
        if (accuracy < bestAccuracy) {  
            bestResult = location;  
            bestAccuracy = accuracy;  
            bestAge = time;  
        }  
    }  
}  
  
// Return best reading or null  
if (bestAccuracy > minAccuracy  
    || (System.currentTimeMillis() - bestAge) > maxAge) {  
    return null;  
} else {  
    return bestResult;  
}  
}
```

BATTERY SAVING TIPS

ALWAYS CHECK LAST KNOWN MEASUREMENT

RETURN UPDATES AS INFREQUENTLY AS
POSSIBLE. LIMIT MEASUREMENT TIME

USE THE LEAST ACCURATE MEASUREMENT
NECESSARY

TURN OFF UPDATES IN ONPAUSE()

MAPS

A VISUAL REPRESENTATION OF AREA

ANDROID PROVIDES MAPPING SUPPORT
THROUGH THE GOOGLE MAPS ANDROID V2
API

MAP TYPES

NORMAL: TRADITIONAL ROAD MAP

SATELLITE - AERIAL PHOTOGRAPH

HYBRID - SATELLITE + ROAD MAP

TERRAIN - TOPOGRAPHIC DETAILS

CUSTOMIZING THE MAP

CHANGE THE CAMERA POSITION

ADD MARKERS & GROUND OVERLAYS

RESPOND TO GESTURES

INDICATE THE USER'S CURRENT LOCATION

SOME MAP CLASSES

GOOGLEMAP

MAPFRAGMENT

CAMERA

MARKER

SETTING UP A MAPS APPLICATION

SET UP THE GOOGLE PLAY SERVICES SDK

OBTAIN AN API KEY

SPECIFY SETTINGS IN APPLICATION
MANIFEST

ADD MAP TO PROJECT

SEE: [https://developers.google.com/maps
/documentation/android/start](https://developers.google.com/maps/documentation/android/start)

MAP PERMISSIONS

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

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

MAP PERMISSIONS

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

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

MAP PERMISSIONS

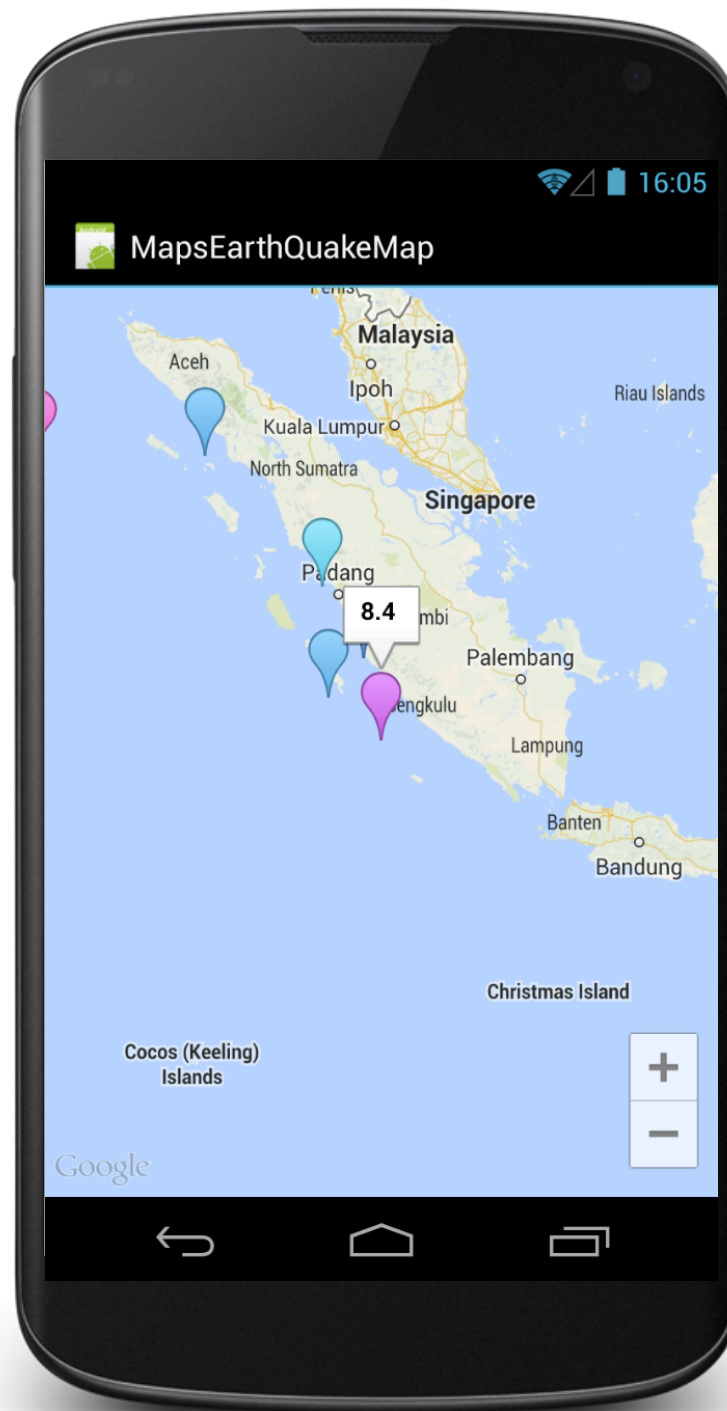
```
<uses-permission android:name=  
    "android.permission.ACCESS_COARSE_LOCATION"/>
```

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

MAP EARTHQUAKE MAP

THIS APPLICATION ACQUIRES EARTHQUAKE
DATA FROM A SERVER

THEN IT DISPLAYS THE DATA ON A MAP, USING
CLICKABLE MARKERS



MAP EARTHQUAKE MAP

```
// The Map Object
private GoogleMap mMap;

// URL for getting the earthquake
// replace with your own user name

private final static String UNAME = "aporter";
private final static String URL = "http://api.geonames.org/earthquakesJSON?north=44.1&south=-9.9&east=-22.4&west=55.2&username="
    + UNAME;

public static final String TAG = "MapsEarthquakeMapActivity";

// Set up UI and get earthquake data
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    new HttpGetTask().execute(URL);
}
```

MAP EARTHQUAKE MAP

```
private class HttpGetTask extends
    AsyncTask<String, Void, List<EarthquakeRec>> {

    AndroidHttpClient mClient = AndroidHttpClient.newInstance("");

    @Override
    protected List<EarthquakeRec> doInBackground(String... params) {

        HttpGet request = new HttpGet(params[0]);
        JSONResponseHandler responseHandler = new JSONResponseHandler();

        try {

            // Get Earthquake data in JSON format
            // Parse data into a list of EarthquakeRecs

            return mClient.execute(request, responseHandler);

        } catch (ClientProtocolException e) {
            Log.i(TAG, "ClientProtocolException");
        } catch (IOException e) {
            Log.i(TAG, "IOException");
        }

        return null;
    }
}
```


MAP EARTHQUAKE MAP

```
@Override
protected void onPostExecute(List<EarthQuakeRec> result) {

    // Get Map Object
    mMap = ((MapFragment) getFragmentManager().findFragmentById(
        R.id.map)).getMap();

    if (null != mMap) {

        // Add a marker for every earthquake

        for (EarthQuakeRec rec : result) {

            // Add a new marker for this earthquake
            mMap.addMarker(new MarkerOptions()

                // Set the Marker's position
                .position(new LatLng(rec.getLat(), rec.getLng()))

                // Set the title of the Marker's information window
                .title(String.valueOf(rec.getMagnitude()))

                // Set the color for the Marker
                .icon(BitmapDescriptorFactory
                    .defaultMarker(getMarkerColor(rec
                        .getMagnitude()))));

        }
    }
}
```

MAPEARTHQUAKEMAP

```
// Center the map
// Should compute map center from the actual data

mMap.moveCamera(CameraUpdateFactory.newLatLng(new LatLng(
    CAMERA_LAT, CAMERA_LNG)));
}

if (null != mClient)
    mClient.close();
}
```

NEXT TIME

DATA MANAGEMENT