Romain Chiappinelli

RLChiapp

Android

07.03.14

Location & Map

location

- position on the Earth
  - latitude
  - longitude
  - timestamp
  - optional
    - accuracy
    - altitude
    - speed
    - bearing
- from
  - GPS
  - Cell phone tower
  - WiFi
  - Network
- LocationProvider
  - Permission
    - ACCESS_COARSE_LOCATION
    - _FINE
  - Passive
  - last known location
- LocationManager .requestLocationUpdates()
  - getSystemService(context.LOCATION_SERVICE)
- LocationListener))
  - @ onLocationChanged()
  - @ onProviderDisabled()
  - Enabled
  - @ onStatusChanged()

Google Maps Android V2 API

- Types
  - Normal
  - Satellite
  - Hybrid
  - Terrain
- permission
  - INTERNET
  - ACCESS_NETWORK_STATE
  - WRITE_EXTERNAL_STORAGE
  - READ_GSERVICES
- Google play SDK
  - API key
- Google Map
  - Marker
  - MapFragment
  - Camera

battery Saving

- check last know measurement
- limit measurement time
- infrequently
- turn off updates() in onPause()

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

LocationProvider

Represents a location data source

Actual data may come from

GPS satellites

Cell phone towers

WiFi access points

LocationProvider 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

LocationProvider

Different LocationProviders 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)
```

get reference to location manager

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)
```

new location

user enables/dis

Obtaining Location

Start listening for updates from location providers

register a listener

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 — how long? find restaurant

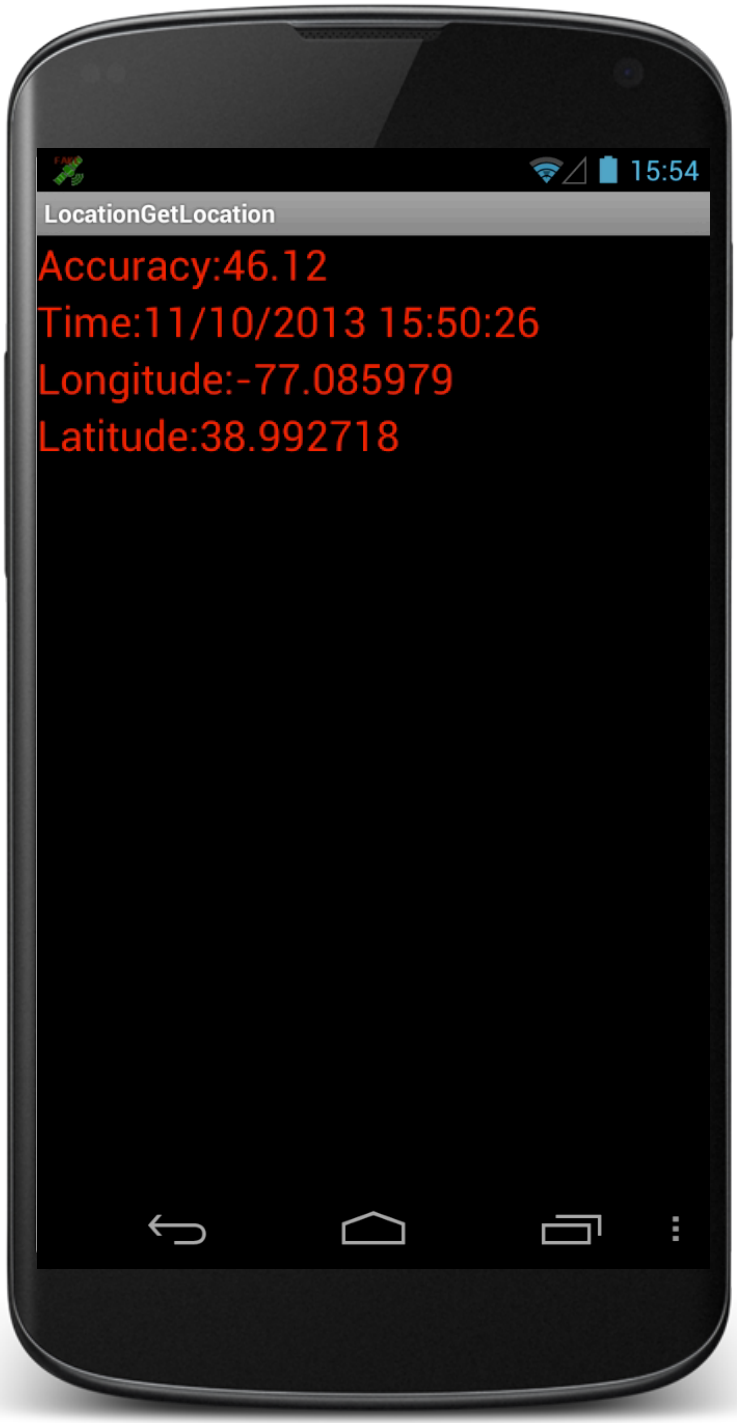
Accuracy

power needs — impact battery

LocationGetLocation

Application acquires and displays the last known locations from all providers

If necessary, acquires and displays new readings from all providers


15:54
LocationGetLocation
Accuracy:46.12
Time:11/10/2013 15:50:26
Longitude:-77.085979
Latitude:38.992718

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 != mLocationManager
                .getProvider(LocationManager.GPS_PROVIDER)) {
            mLocationManager.requestLocationUpdates(
                    LocationManager.GPS_PROVIDER, POLLING_FREQ,
                    MIN_DISTANCE, mLocationListener);
        }

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

            @Override
            public void run() {

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

                mLocationManager.removeUpdates(mLocationListener);

            }
        }, 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 ← part of map that is visible and user viewpoint

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

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"/>
```

allows maps API to access Google services

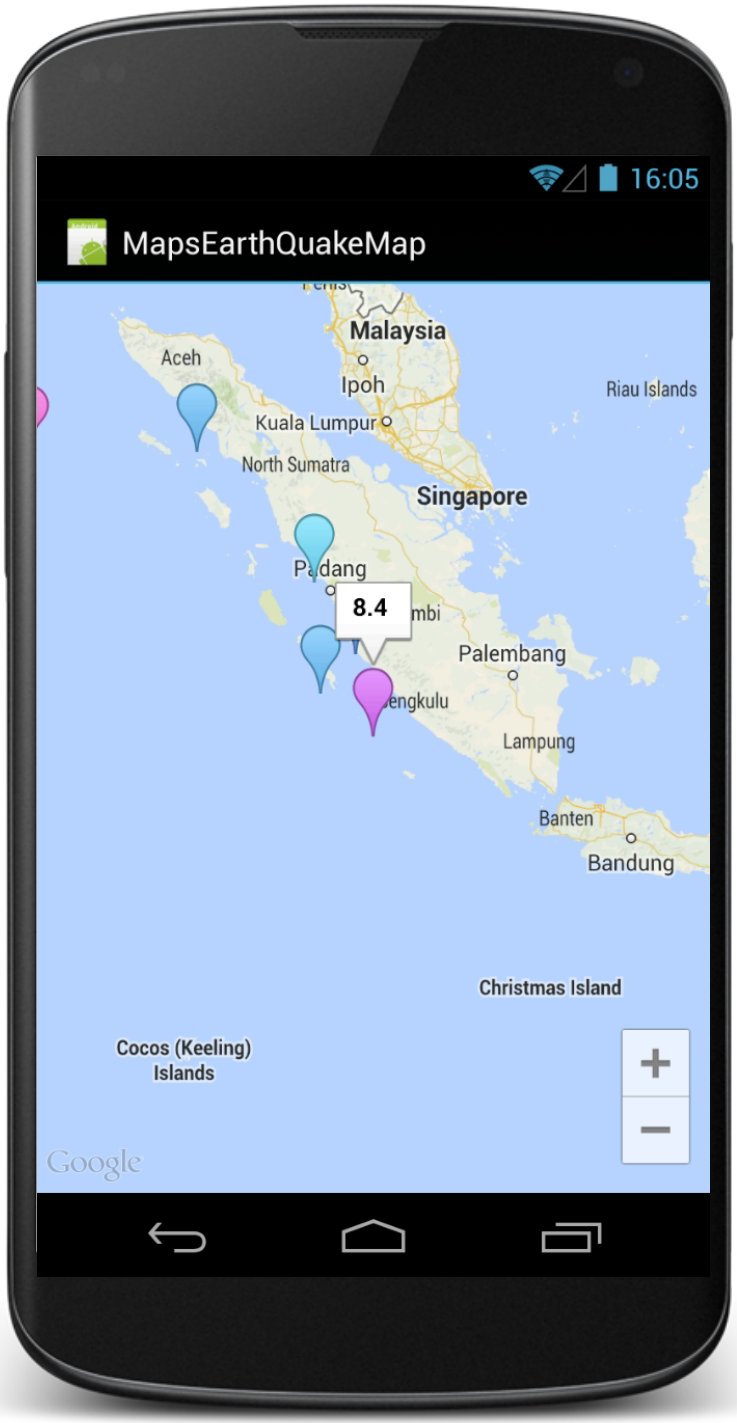
Map Permissions

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

MapEarthQuakeMap

This application acquires earthquake data from a server

Then it displays the data on a map, using clickable markers


16:05
MapsEarthQuakeMap
Malaysia
Aceh
Ipoh
Riau Islands
Kuala Lumpur
North Sumatra
Singapore
Padang
8.4
Palembang
Lampung
Banten
Bandung
Christmas Island
Cocos (Keeling) Islands
Google

MapEarthQuakeMap

```
// 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);

}
```

MapEarthQuakeMap

```
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;

    }
```

MapEarthQuakeMap

```
@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