Lab on apps development for tablets, smartphones and smartwatches

Week 6(a): System services and Sensors

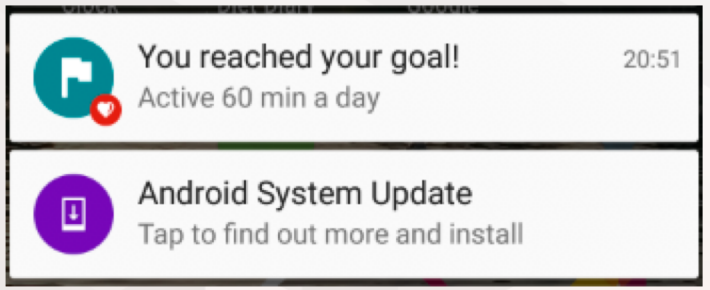
Dr. Marina Zapater, Prof. David Atienza

Mr. Grégoire Surrel, Ms. Elisabetta de Giovanni,
Mr. Dionisijie Sopic, Ms. Halima Najibi, Ms. Farnaz Forooghifar

Embedded Systems Laboratory (ESL) – Faculty of Engineering (STI)

Outline of the class

- **System Services**
  - We leave other types of services for next week
- Reading from sensors
  - Using system services
- Geo-location services
- Today's lab
  - Sensors
  - GoogleMaps!



Services

- Reason to use them → The UI must be always fast:
  - Screen is updated every 16ms → UI thread has 16ms to do all the work

- We will execute long-running tasks on a background thread
  - Complete all work in less than 16 ms for each screen
  - Run slow non-UI work on a non-UI thread

Services

- A **Service** is an subclass that can perform long-running operations in background and does not provide an user interface
- A Service provides a robust environment for background tasks
- Activity vs. service
  - Activity: UI, can be disposed when it loses visibility
  - Service: No UI, disposed when it terminates
- We use them for
  - Network transactions, playing music, perform file I/O, interact with content providers…

System Services

- We can create our own services or use the **system services**
- There is a wide list of system services available
  - **Sensor Service**
  - ***Notification Service*** → *Next week*
  - ***Alarm Service*** → *Next week*
  - Power Manager Service
  - Vibrator Service, Audio Service
  - Telephony Service, Connectivity Service, Wi-Fi Service
  - ...
- We can access the system services programmatically

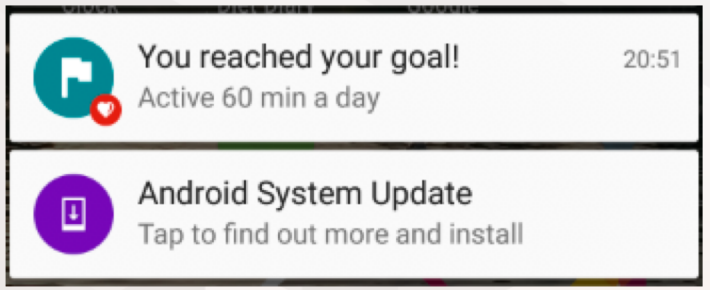
```
SensorManager sm = (SensorManager) getSystemService(SENSOR_SERVICE);
```

```
NotificationManager mNotifMgr = (NotificationManager) getSystemService (NOTIFICATION_SERVICE);
```

```
AlarmManager myAlarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);
```

Outline of the class

- System Services
  - We leave all other types of services for next week
- **Reading from sensors**
  - Using system services
- Geo-location services
- Today's lab
  - Sensors
  - GoogleMaps!



Reading from sensors

- We interact with sensors via a System Service:

```
SensorManager sm = (SensorManager) getSystemService(SENSOR_SERVICE);
```

- Many types of sensors
  - Accelerometer, Gyroscope, Light, …
  - GPS
  - Heart Rate (watch only!)
  - …
- Each Sensor contains information about the vendor, type and others
- Not all smartphones/watches have the same sensors!
  - How to know which sensors does your smartphone have?

```
public List<Sensor> getSensorList(int type);
type → TYPE_ACCELEROMETER, TYPE_LIGHT, TYPE_ALL, …
```

Monitoring sensor events

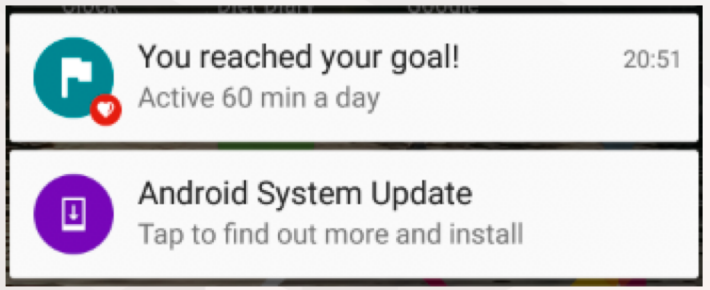
- To get raw sensor data, we need to implement two callbacks, exposed via the SensorEventListener interface
  - A sensor's accuracy changes
    - onAccuracyChanged() is called
  - A sensor reports a new value
    - onSensorChanged() is called

- Steps:
  1. Implementing SensorEventListener
  2. Creating the sensor manager
  3. Registering to the listener
  4. Doing something when sensor accuracy/value changes

```
public class SensorActivity extends Activity implements SensorEventListener {   1.
  private SensorManager mSensorManager;
  private Sensor mLight;

  @Override
  public final void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

2.  mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
  }

  @Override
  public final void onAccuracyChanged(Sensor sensor, int accuracy) {
    // Do something here if sensor accuracy changes.
  }
4.

  @Override
  public final void onSensorChanged(SensorEvent event) {
    // The light sensor returns a single value.
    // Many sensors return 3 values, one for each axis.
    float lux = event.values[0];
    // Do something with this sensor value.
  }

3.  @Override
    protected void onResume() {
      super.onResume();
      mSensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
    }

    @Override
    protected void onPause() {
      super.onPause();
      mSensorManager.unregisterListener(this);
    }
}
```

Outline of the class

- System Services
  - We leave all other types of services for next week
- Reading from sensors
  - Using system services
- **Geo-location services**
  - … and the GoogleMaps API
- Today's lab
  - Sensors
  - GoogleMaps!



LocationService

- The central component of the location framework is the LocationManager **system service**
- Very similar way of working than any other sensor:
- Once your application has a LocationManager, your can do three things:
  - Query the list of all LocationProvider instances to determine the last known user location.
  - Register/unregister for periodic updates of the user's current location from a location provider (specified either by criteria or name).
  - Register/unregister for a given Intent to be fired if the device comes within a given proximity (specified by radius in meters) of a given lat/long.

Using the LocationService

- Requesting Location updates via the Location service:
  1. Acquiring LocationManager
  2. Defining a listener
  3. Registering the listener to receive the network updates
     1. Here we choose the “source” of location
     2. For example, cell network and WiFi → NETWORK_PROVIDER
     3. Or using only GPS → GPS_PROVIDER
- You will need specific permissions

1.
```
// Acquire a reference to the system Location Manager
LocationManager locationManager = (LocationManager) this.getSystemService(Context.LOCATION_SERVICE);
```

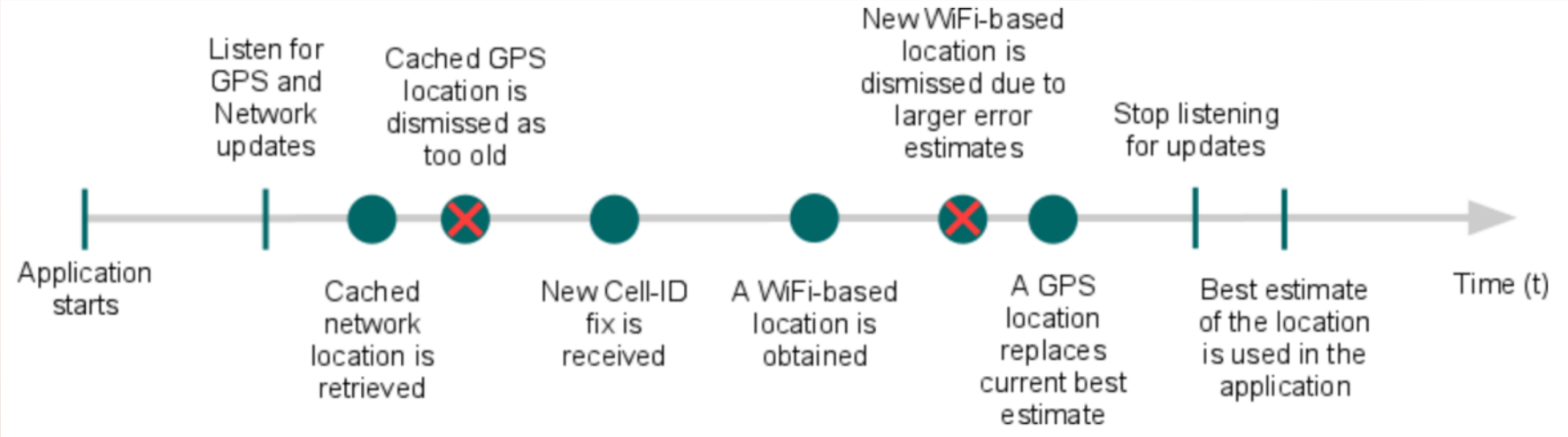
2.
```
// Define a listener that responds to location updates
LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
      // Called when a new location is found by the network location provider.
      makeUseOfNewLocation(location);
    }

    public void onStatusChanged(String provider, int status, Bundle extras) {}

    public void onProviderEnabled(String provider) {}

    public void onProviderDisabled(String provider) {}
  };
```

3.
```
// Register the listener with the Location Manager to receive location updates
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0, locationListener);
```

```
<manifest ... >
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    ...
    <!-- Needed only if your app targets Android 5.0 (API level 21) or higher. -->
    <uses-feature android:name="android.hardware.location.gps" />
    ...
</manifest>
```

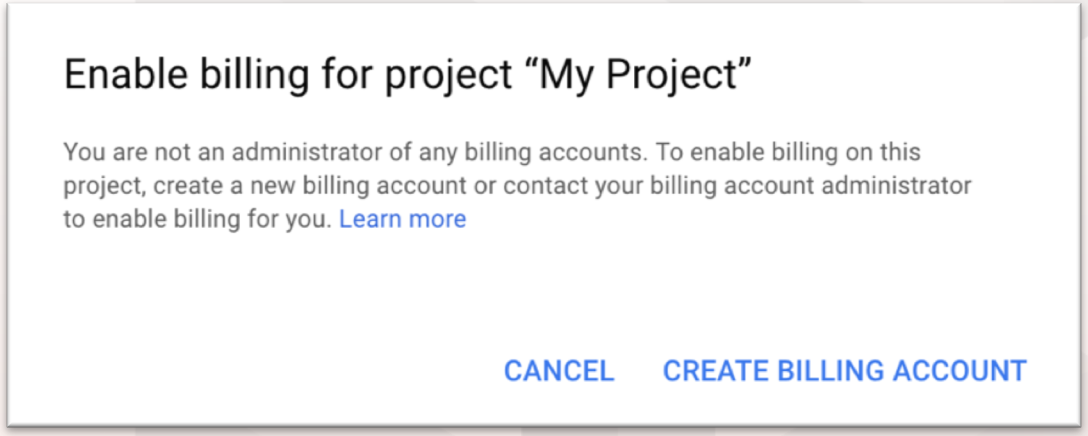
Correct flow for obtaining user locations

1. Start application.
2. Some time later, start listening for updates from desired location providers.
3. Maintain a "current best estimate" of location by filtering out new, but less accurate fixes.
4. Stop listening for location updates.
5. Take advantage of the last best location estimate.



Google Maps API warning

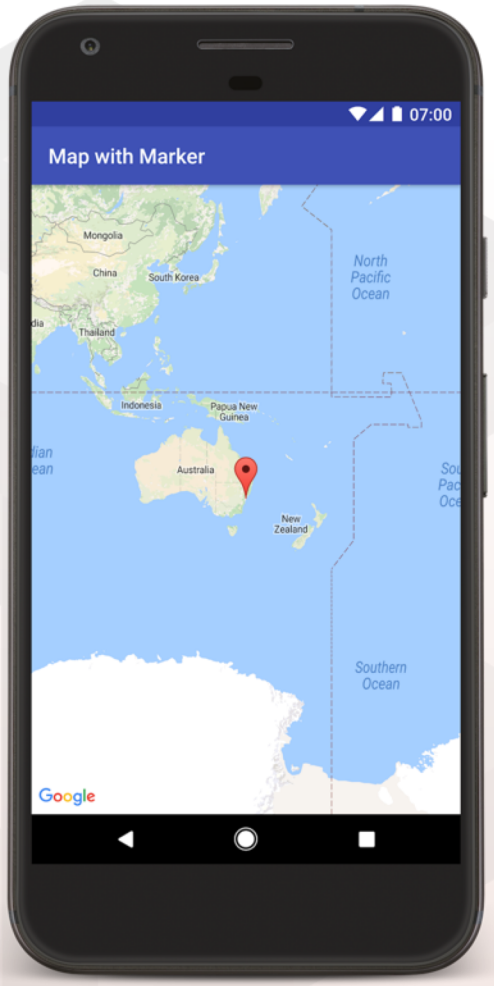
- Google Maps has a de-facto monopoly situation
- Recently changed their Maps API usage policy
  - it's still free for little usage (the first 200$ are not billed)
  - Prices went up dramatically in July 2018 (up to 30x!)
  - A credit card is required, rather than blocking the map display as previously
- Alternatives exist!
  - Data: OpenStreetMap
  - API: third-party providers (Mapbox)
  - Different features (no StreetView)
  - You can use it if you prefer
    - But we won't be using it for this lab
    - Mapbox only requires to register to get started



Mapbox's Android library: https://www.mapbox.com/android-docs/maps/overview/
Other exist: https://wiki.openstreetmap.org/wiki/Android#Maps

The GoogleMaps API

- The API allows you to add maps to your app that are based on Google Maps data.



- Takes care of:
  - Access Google maps servers
  - Data downloading
  - Map display
  - Touch gestures on the map.

- The key class is MapView.
  - A MapView displays a map with data obtained from the Google Maps service.

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

Inserting a map inside the app

- Permissions should be added to the AndroidManifest.xml, and the Activation Key must be specified in the meta-data.
  - Internet Access
  - Localization capabilities
  - Access to Google Web services
  - OpenGL ES version 2 libraries
  - Access to network state

```
resValue "string", "google_maps_key",
        (project.findProperty("GOOGLE_MAPS_API_KEY") ?: "")
```

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

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

        <!--
             The API key for Google Maps-based APIs.
        -->
        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="@string/google_maps_key" />

        <activity
            android:name=".MapsMarkerActivity"
            android:label="@string/title_activity_maps">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Adding a map using a fragment

1. Add a fragment to your activity layout
2. In onCreate() get a handle to the map
3. Implement the onReadyCallback() to setup the map

1.

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.mapwithmarker.MapsMarkerActivity" />
```

2.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Retrieve the content view that renders the map.
    setContentView(R.layout.activity_maps);
    // Get the SupportMapFragment and request notification
    // when the map is ready to be used.
    SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
            .findFragmentById(R.id.map);
    mapFragment.getMapAsync(this);
}
```

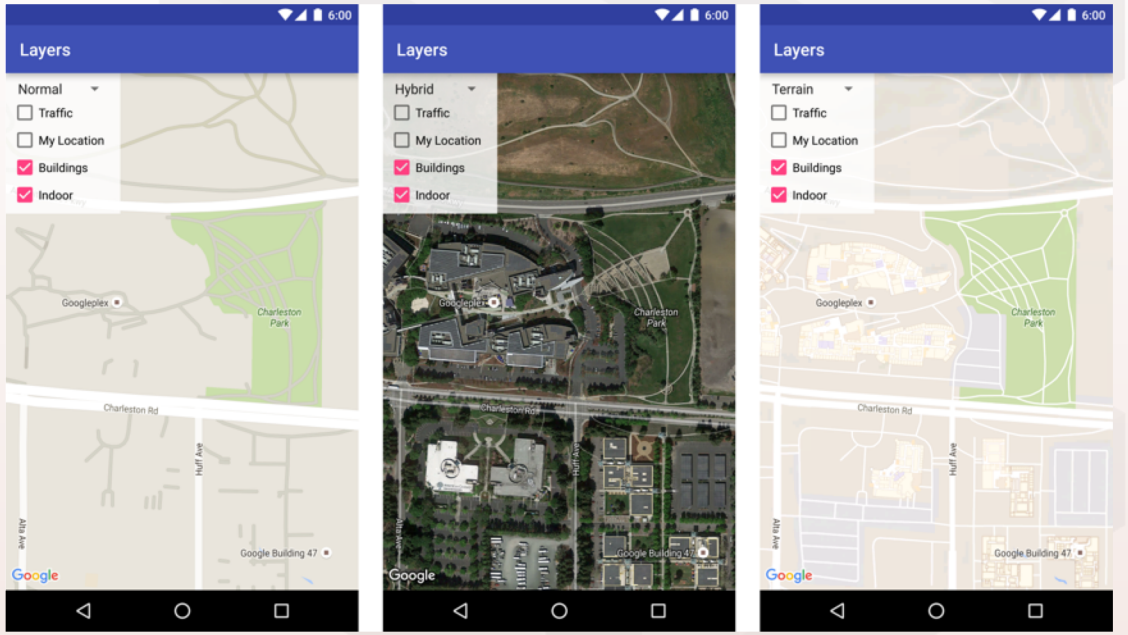
3.

```
public class MapsMarkerActivity extends AppCompatActivity
        implements OnMapReadyCallback {
    // Include the OnCreate() method here too, as described above.
    @Override
    public void onMapReady(GoogleMap googleMap) {
        // Add a marker in Sydney, Australia,
        // and move the map's camera to the same location.
        LatLng sydney = new LatLng(-33.852, 151.211);
        googleMap.addMarker(new MarkerOptions().position(sydney)
                .title("Marker in Sydney"));
        googleMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
    }
}
```

Customizing the map: Type

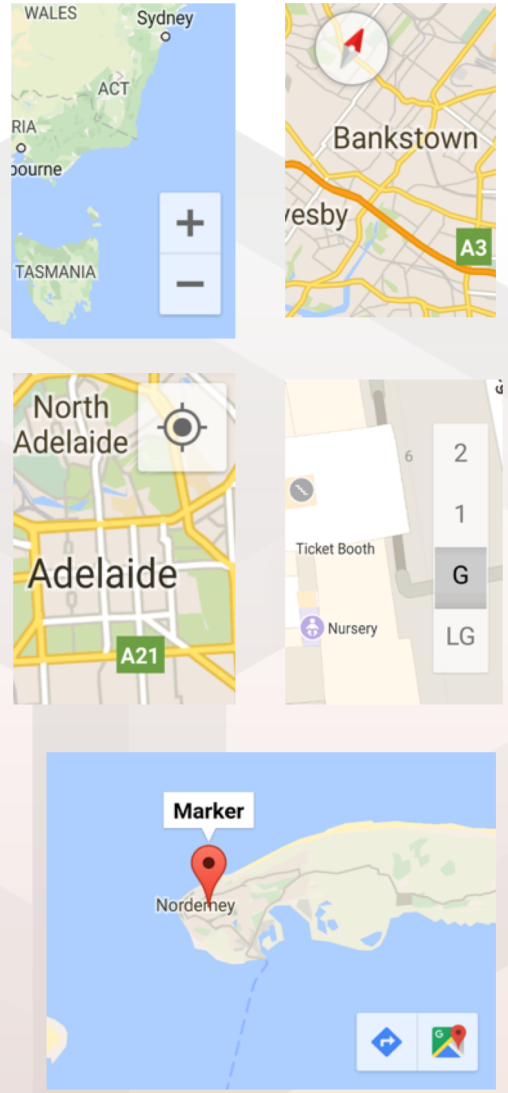
- Define the Map type, governing the overall representation of the map
  - Normal → Typical road map.
  - Hybrid → Satellite photograph data with road maps added.
  - Satellite → Satellite photograph data. Road and feature labels are not visible.
  - Terrain → Topographic data. The map includes colors, contour lines and labels, and perspective shading.
  - None → no tiles, empty grid.

```
GoogleMap map;
...
// Sets the map type to be "hybrid"
map.setMapType(GoogleMap.MAP_TYPE_HYBRID);
```



Interacting with the map: Controls and gestures

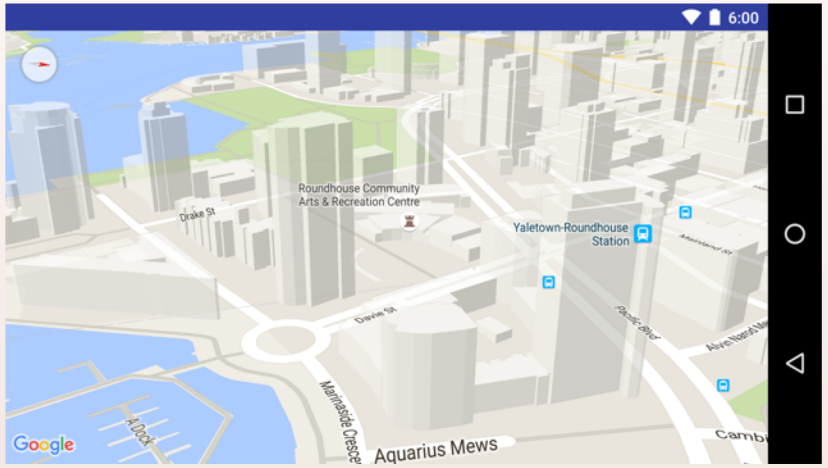
- Controls and gestures:
  - You can use the lite mode for minimal user interaction
- UI controls include:
  - Zoom → UiSettings.setZoomControlsEnabled(true)
  - Compass → UiSettings.setCompassEnabled(boolean)
  - “My location button”
  - Level picker
- Map toolbar:
  - Enable/disable → UiSettings.setMapToolbarEnabled(boolean)
- Map gestures:
  - Zoom, scroll, tilt, rotate…
    - uiSettings.set{Zoom,Scroll,Tilt} GesturesEnabled(boolean)

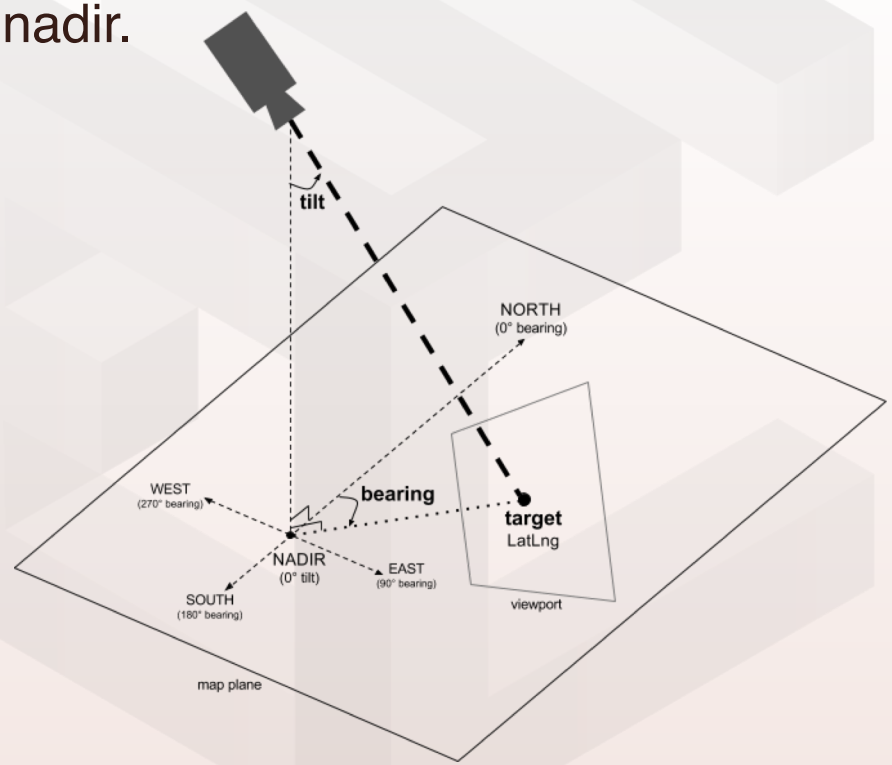


Camera and View

- You can change the user's viewpoint of the map by modifying the map's **camera**.
  - Location → expressed in forms of latitude/longitude coordinates.
  - Zoom → defines the scale levels of the map.
  - Bearing → defines the map orientation, i.e. the direction in which a vertical line on the map points, measured in degrees clockwise from north.
  - Tilt → viewing angle, measured as degrees from the nadir.

- You can enable 3D buildings
  - GoogleMap.setBuildingsEnabled(true)





Drawing on the map: Markers

- Markers can be used to identify locations on the GoogleMap.

- Markers can be customized in terms of:
  - Icon to be displayed
  - Position of the marker on the map
  - Title and text to be displayed
  - Events to be managed

```
@Override
public void onMapReady(GoogleMap map) {
    map.addMarker(new MarkerOptions()
        .position(new LatLng(10, 10))
        .title("Hello world"));
}
```

Interacting with the map: Events

- Developers can handle the events on the Google Map.
- Events are managed through the listener mechanism seen so far…
  - **Click events:** Implement the onMapClickListener() interface and the onMapLongClickListener method.
  - **Camera events**: Implement the OnCameraChangeListener interface and the onCameraChange(CameraPosition) method.

For a `MapView`:

```
MapView view;
...
view.setClickable(false);
```

For a `MapFragment`:

```
MapFragment fragment;
...
fragment.getView().setClickable(false);
```


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

StreetView

- Panoramic 360-degree views from designated roads throughout its coverage area.
- Use StreetViewPanorama class:
  1. Add a Fragment to the Activity
  2. Implement onStreetViewPanoramaReadyCallback
  3. Use the onStreetViewPanoramaReady() callback
  4. Call getStreetViewPanoramaAsync() on the fragment

1.

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

2.

```
public class MainActivity extends FragmentActivity
    implements OnStreetViewPanoramaReadyCallback {
...
}
```

3.

```
@Override
public void onStreetViewPanoramaReady(StreetViewPanorama panorama) {
    panorama.setPosition(new LatLng(-33.87365, 151.20689));
}
```

4.

```
StreetViewPanoramaFragment streetViewPanoramaFragment =
    (StreetViewPanoramaFragment) getFragmentManager()
        .findFragmentById(R.id.streetviewpanorama);
streetViewPanoramaFragment.getStreetViewPanoramaAsync(this);
```

Maps API on Android Wear

- Adding dependencies to the Wear module
  - build.gradle script
- Using the onMapReady(GoogleMap) callback to handle the GoogleMap object
- You can enable AmbientMode → activated when the user is not actively using the app
- You can enable StreetView

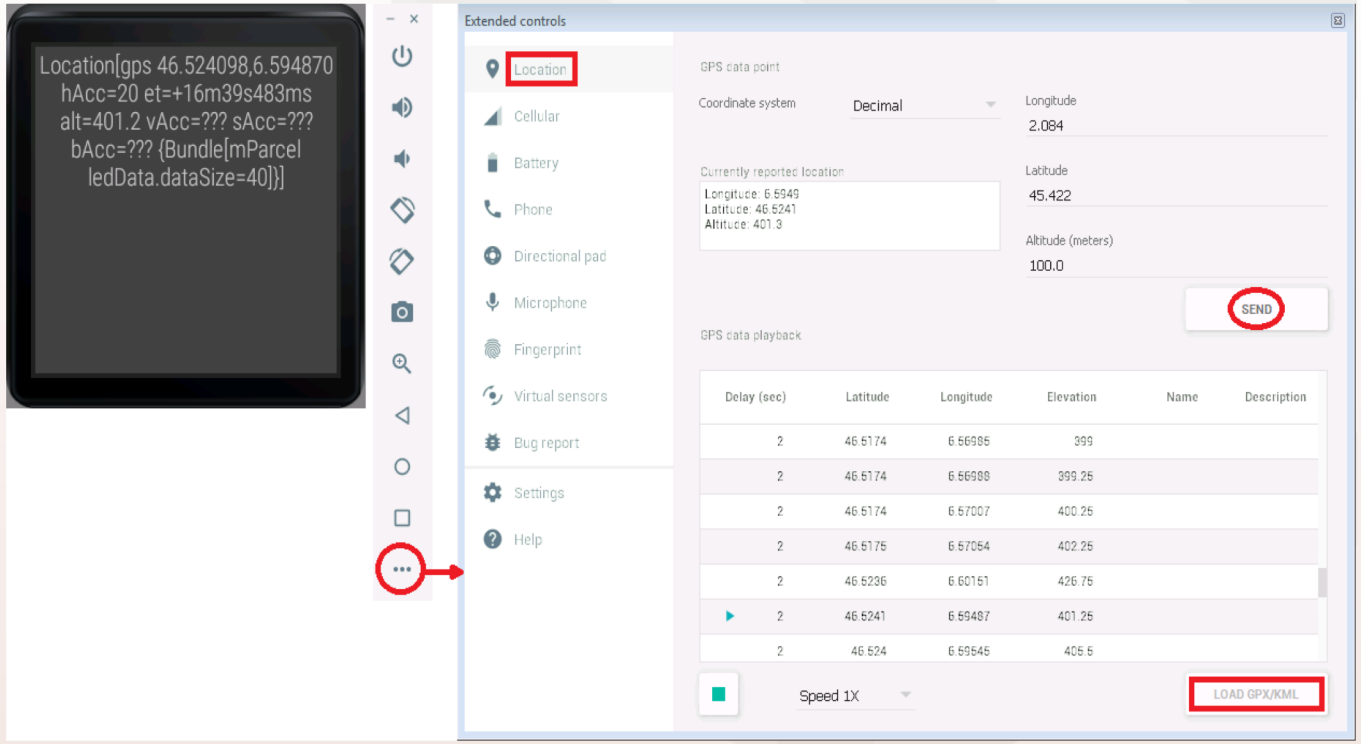

```
dependencies {
  compile fileTree(dir: 'libs', include: ['*.jar'])
  compile 'com.google.android.support:wearable:1.2.0'
  provided 'com.google.android.wearable:wearable:1.0.0'
  compile 'com.google.android.gms:play-services-maps:11.4.2'
}
```

```
public class MainActivity extends WearableActivity
        implements OnMapReadyCallback, GoogleMap.OnMapLongClickListener {

    private static final LatLng SYDNEY = new LatLng(-33.85704, 151.21522);
    private GoogleMap mMap;

    ...

    @Override
    public void onMapReady(GoogleMap map) {
        mMap = map;
        mMap.addMarker(new MarkerOptions().position(SYDNEY)
            .title("Sydney Opera House"));
        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(SYDNEY, 10));
        mMap.setOnMapLongClickListener(this);
    }
}
```

How to test if everything works?

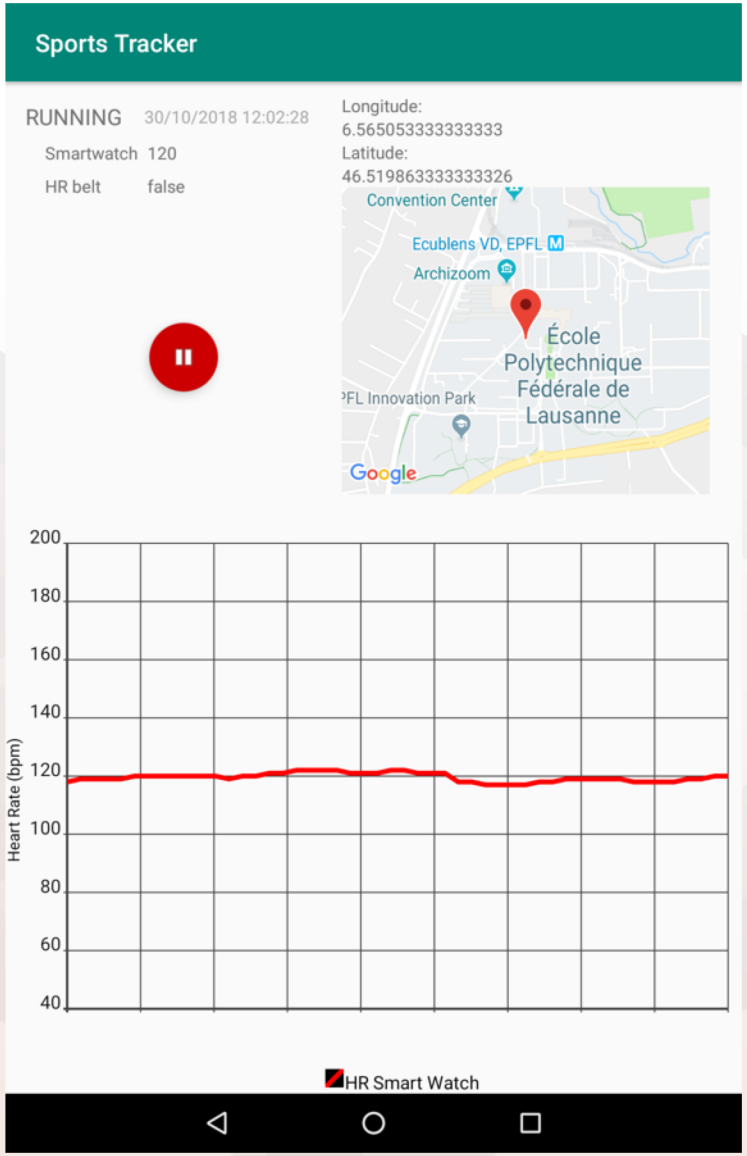
- The Android emulator provides ways of mocking the values of sensors
  - Simply click on the bottommost icon in the emulator
  - You will be able to fake values of sensors

- Specially useful for LocationService!



Today's lab

- Reading from Heart Rate (HR) Sensor!
  - Saving data to firebase
- Using the location service
  - Displaying live location in GoogleMaps



Questions?