Groups should submit 1-slide on their final project (due next class)
Using Maps
MapView and MapActivity

- **MapView**: UI widget that displays maps
- **MapActivity**: Java class (extends Activity), handles map-related lifecycle and management for displaying maps.
7 Steps for using Google Maps Android API
https://developers.google.com/maps/documentation/android-api/start

1. Install Android SDK (Done!!)
2. Add Google Play services to Android Studio
3. Create a Google Maps project
4. Obtain Google Maps API key
5. Hello Map! Take a look at the code
6. Connect an Android device
7. Build and run your app
Step 2: Add Google Play Services to Android Studio

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

- Google Maps API v2 is part of Google Play Services SDK
- Use Android Studio SDK manager to download Google Play services

Open SDK Manager
Click on SDK Tools
Check Google Play Services, then Ok
Step 3: Create new Android Studio Project
https://developers.google.com/maps/documentation/android-api/start

- Select “Google Maps Activity, click Finish
Step 4: Get Google Maps API key

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

- To access Google Maps servers using Maps API, must add Maps API key to app
- Maps API key is free. E.g.

Your API key
A1zaSyCc0_lEEjP11TlnPkJvSx10YIY7oBa9XsXs

- Google uses API key to uniquely identify your app, track its resource usage, etc
Step 4a: Fast, Easy way to get Maps API Key

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

- Copy link provided in `google_maps_api.xml` of Maps template into browser
- Goes to Google API console, auto-fills form
- Creates API key

Register your application for Google Maps Android API in Google API Console

Google API Console allows you to manage your application and monitor API usage.

You have no existing projects. A new project named 'My Project' will be created.

Please email me updates regarding feature announcements, performance suggestions, feedback surveys and special offers.

- Yes  - No

I agree that my use of any services and related APIs is subject to my compliance with the applicable Terms of Service.

- Yes  - No

Agree and continue

The API is enabled

The project has been created and Google Maps Android API has been enabled.

Next, you'll need to create an API key in order to call the API.

Create API key
Step 4a: Fast, Easy way to get Maps API Key

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

- If successful, Maps API key generated
  
  API key created
  
  Use this key in your application by passing it with the \texttt{key=API\_KEY} parameter.
  
  Your API key
  
  \texttt{AIzaSyCc0\_lEEjPllTlnPkVsX1OYIY7oBa9XsXs}
  
  \textbf{Restrict your key to prevent unauthorized use in production.}

- Copy key, put it in \texttt{<string>} element in \texttt{google\_maps\_api.xml} file

\texttt{<string name=\textit{google\_maps\_key} templateMergeStrategy=\textit{preserve} translatable=false>AIzaSyCc0\_lEEjPllTlnPkVsX1OYIY7oBa9XsXs</string>
Step 4b: Longer (older) way to API key

- If easy way doesn’t work, older way to obtain a Maps API key
- Follow steps at:
  - See: https://developers.google.com/maps/documentation/android-api/signup
Step 5: Examine Code Generated by Android Studio Maps Template

- XML file that defines layout is in `res/layout/activity_maps.xml`

```xml
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
          xmlns:tools="http://schemas.android.com/tools"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:id="@+id/map"
          tools:context="MapsActivity"
          android:name="com.google.android.gms.maps.SupportMapFragment" />
```
Step 5: Examine Code Generated by Android Studio Maps Template

- Default Activity file is `MapActivity.java`

```java
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

public class MapActivity extends FragmentActivity implements OnMapReadyCallback {

    private GoogleMap mMap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }

    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;

        // Add a marker in Sydney, Australia, and move the camera.
        LatLng sydney = new LatLng(-34, 151);
        mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
    }
}
```
Steps 6, 7

- **Step 6:** Connect to an Android device (smartphone)

- **Step 7:** Run the app
  - Should show map with a marker on Sydney Australia

- More code examples at:
  - https://github.com/googlemaps/android-samples
AsyncTask API
AsyncTask API

- For compute intensive tasks, remote or tasks that take a long time, doing it in main activity blocks
- **AsyncTask**: spawn separate thread to offload such task, free up main Activity
Android Sensors
What is a Sensor?

- Converts physical quantity (e.g. light, acceleration, magnetic field) into a signal
- **Example:** accelerometer converts acceleration along X,Y,Z axes into signal
So What?

- Raw sensor data can be processed into useful info
- **Example:** Raw accelerometer data can be processed/classified to infer user’s activity (e.g. walking, running, etc)
- Voice samples can be processed/classified to infer whether speaker is nervous or not

![Raw accelerometer readings](image1)

Machine learning
Feature extraction and classification

Walking
Running
Jumping
Step count
Calories burned
Falling
Android Sensors

- Microphone (sound)
- Camera
- Temperature
- Location (GPS, A-GPS)
- Accelerometer
- Gyroscope (orientation)
- Proximity
- Pressure
- Light

- Different phones do not have all sensor types!!
Android Sensor Framework

● Enables apps to:
  ● Access sensors available on device and
  ● Acquire raw sensor data

● Specifically, using the Android Sensor Framework, you can:
  ● Determine **which sensors** are available on phone
  ● Determine **capabilities of sensors** (e.g. max. range, manufacturer, power requirements, resolution)
  ● **Register and unregister** sensor event listeners
  ● **Acquire raw sensor data** and define data rate
Android Sensor Framework

- Android sensors can be either hardware or software

- **Hardware sensor:**
  - physical components built into phone,
  - **Example:** temperature

- **Software sensor (or virtual sensor):**
  - Not physical device
  - Derives their data from one or more hardware sensors
  - **Example:** gravity sensor
Sensor Types Supported by Android

- **TYPE_PROXIMITY**
  - Measures an **object’s proximity to device’s screen**
  - **Common uses:** determine if handset is held to ear

- **TYPE_GYROSCOPE**
  - Measures device’s **rate of rotation** around X,Y,Z axes in rad/s
  - **Common uses:** rotation detection (spin, turn, etc)
# Types of Sensors

<table>
<thead>
<tr>
<th>Sensor</th>
<th>HW/SW</th>
<th>Description</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_ACCELEROMETER</td>
<td>HW</td>
<td>Rate of change of velocity</td>
<td>Shake, Tilt</td>
</tr>
<tr>
<td>TYPE_AMBIENT_TEMPERATURE</td>
<td>HW</td>
<td>Room temperature</td>
<td>Monitor Room temp</td>
</tr>
<tr>
<td>TYPE_GRAVITY</td>
<td>SW/HW</td>
<td>Gravity along X,Y,Z axes</td>
<td>Shake, Tilt</td>
</tr>
<tr>
<td>TYPE_GYROSCOPE</td>
<td>HW</td>
<td>Rate of rotation</td>
<td>Spin, Turn</td>
</tr>
<tr>
<td>TYPE_LIGHT</td>
<td>HW</td>
<td>Illumination level</td>
<td>Control Brightness</td>
</tr>
<tr>
<td>TYPE_LINEAR_ACCELERIMATION</td>
<td>SW/HW</td>
<td>Acceleration along X,Y,Z – g</td>
<td>Accel. Along an axis</td>
</tr>
<tr>
<td>TYPE_MAGNETIC_FIELD</td>
<td>HW</td>
<td>Magnetic field</td>
<td>Create Compass</td>
</tr>
<tr>
<td>TYPE_ORIENTATION</td>
<td>SW</td>
<td>Rotation about X,Y,Z axes</td>
<td>Device position</td>
</tr>
<tr>
<td>TYPEPRESSURE</td>
<td>HW</td>
<td>Air pressure</td>
<td>Air pressure</td>
</tr>
<tr>
<td>TYPE_PROXIMITY</td>
<td>HW</td>
<td>Any object close to device?</td>
<td>Phone close to face?</td>
</tr>
<tr>
<td>TYPE_RELATIVE_HUMIDITY</td>
<td>HW</td>
<td>% of max possible humidity</td>
<td>Dew point</td>
</tr>
<tr>
<td>TYPE_ROTATION_VECTOR</td>
<td>SW/HW</td>
<td>Device’s rotation vector</td>
<td>Device’s orientation</td>
</tr>
<tr>
<td>TYPE_TEMPERATURE</td>
<td>HW</td>
<td>Phone’s temperature</td>
<td>Monitor temp</td>
</tr>
</tbody>
</table>
2 New Hardware Sensor introduced in Android 4.4

- **TYPE_STEP_DETECTOR**
  - Triggers sensor event each time user takes a step *(single step)*
  - Delivered event has value of 1.0 + timestamp of step

- **TYPE_STEP_COUNTER**
  - Also triggers a sensor event each time user takes a step
  - Delivers total *accumulated number of steps since this sensor was first registered by an app*,
  - Tries to eliminate false positives

- **Common uses**: step counting, pedometer apps
- Requires hardware support, available in Nexus 5
- Alternatively available through Google Play Services (more later)
Sensor Programming

- Sensor framework is part of `android.hardware`
- Classes and interfaces include:
  - `SensorManager`
  - `Sensor`
  - `SensorEvent`
  - `SensorEventListener`
- These sensor-APIs used for:
  1. Identifying sensors and sensor capabilities
  2. Monitoring sensor events
Sensor Events and Callbacks

- Sensors send events to sensor manager asynchronously, when new data arrives

General approach:
- App registers callbacks
- **SensorManager** notifies app of sensor event whenever new data arrives (or accuracy changes)
Sensor

- A class that can be used to create instance of a specific sensor
- Has methods used to determine a sensor’s capabilities
- Included in sensor event object
SensorEvent

- Android system sensor event information as a sensor event object

- **Sensor event object** includes:
  - **Sensor**: Type of sensor that generated the event
  - **Values**: Raw sensor data
  - **Accuracy**: Accuracy of the data
  - **Timestamp**: Event timestamp

Sensor value depends on sensor type
<table>
<thead>
<tr>
<th>Sensor</th>
<th>Sensor event data</th>
<th>Description</th>
<th>Units of measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_ACCELEROMETER</td>
<td>SensorEvent.values[0]</td>
<td>Acceleration force along the x axis (including gravity).</td>
<td>m/s²</td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[1]</td>
<td>Acceleration force along the y axis (including gravity).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[2]</td>
<td>Acceleration force along the z axis (including gravity).</td>
<td></td>
</tr>
<tr>
<td>TYPE_GRAVITY</td>
<td>SensorEvent.values[0]</td>
<td>Force of gravity along the x axis.</td>
<td>m/s²</td>
</tr>
<tr>
<td>TYPE_GYROSCOPE</td>
<td>SensorEvent.values[0]</td>
<td>Rate of rotation around the x axis.</td>
<td>rad/s</td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[1]</td>
<td>Rate of rotation around the y axis.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[2]</td>
<td>Rate of rotation around the z axis.</td>
<td></td>
</tr>
<tr>
<td>TYPE_GYROSCOPE_UNCALIBRATED</td>
<td>SensorEvent.values[0]</td>
<td>Rate of rotation (without drift compensation) around the x axis.</td>
<td>rad/s</td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[1]</td>
<td>Rate of rotation (without drift compensation) around the y axis.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[2]</td>
<td>Rate of rotation (without drift compensation) around the z axis.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[3]</td>
<td>Estimated drift around the x axis.</td>
<td></td>
</tr>
</tbody>
</table>
## Sensor Values Depend on Sensor Type

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Sensor event data</th>
<th>Description</th>
<th>Units of measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_LINEAR_ACCELERATION</td>
<td>SensorEvent.values[0]</td>
<td>Acceleration force along the x axis (excluding gravity).</td>
<td>m/s²</td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[1]</td>
<td>Acceleration force along the y axis (excluding gravity).</td>
<td></td>
</tr>
<tr>
<td>TYPE_ROTATION_VECTOR</td>
<td>SensorEvent.values[0]</td>
<td>Rotation vector component along the x axis ($x \times \sin(\theta/2)$).</td>
<td>Unitless</td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[1]</td>
<td>Rotation vector component along the y axis ($y \times \sin(\theta/2)$).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[2]</td>
<td>Rotation vector component along the z axis ($z \times \sin(\theta/2)$).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[3]</td>
<td>Scalar component of the rotation vector ($\cos(\theta/2)$).$^1$</td>
<td></td>
</tr>
<tr>
<td>TYPE_SIGNIFICANT_MOTION</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TYPE_STEP_COUNTER</td>
<td>SensorEvent.values[0]</td>
<td>Number of steps taken by the user since the last reboot while the sensor was activated.</td>
<td>Steps</td>
</tr>
<tr>
<td>TYPE_STEP_DETECTOR</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
SensorEventListener

- Interface used to create 2 callbacks that receive notifications (sensor events) when:
  - Sensor values change \( \text{onSensorChange}(\ ) \) or
  - When sensor accuracy changes \( \text{onAccuracyChanged}(\ ) \)
Sensor API Tasks

- **Sensor API Task 1: Identifying sensors and their capabilities**
  - Why identify sensor and their capabilities at runtime?
    - Disable app features using sensors not present, or
    - Choose sensor implementation with best performance

- **Sensor API Task 2: Monitor sensor events**
  - Why monitor sensor events?
    - To acquire raw sensor data
    - Sensor event occurs every time sensor detects change in parameters it is measuring
Sensor Availability

- Different sensors are available on different Android versions

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Android 4.0 (API Level 14)</th>
<th>Android 2.3 (API Level 9)</th>
<th>Android 2.2 (API Level 8)</th>
<th>Android 1.5 (API Level 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_ACCELEROMETER</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TYPE_AMBIENT_TEMPERATURE</td>
<td>Yes</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>TYPE_GRAVITY</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>TYPE_GYROSCOPE</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a(^1)</td>
<td>n/a(^1)</td>
</tr>
<tr>
<td>TYPE_LIGHT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TYPE_LINEAR_ACCELERATION</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>TYPE_MAGNETIC_FIELD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TYPE_ORIENTATION</td>
<td>Yes(^2)</td>
<td>Yes(^2)</td>
<td>Yes(^2)</td>
<td>Yes</td>
</tr>
<tr>
<td>TYPE_PRESSURE</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a(^1)</td>
<td>n/a(^1)</td>
</tr>
<tr>
<td>TYPE_PROXIMITY</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TYPE_RELATIVE_HUMIDITY</td>
<td>Yes</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>TYPE_ROTATION_VECTOR</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>TYPE_TEMPERATURE</td>
<td>Yes(^2)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Identifying Sensors and Sensor Capabilities

• First create instance of `SensorManager` by calling `getSystemService()` and passing in `SENSOR_SERVICE` argument

```java
private SensorManager mSensorManager;

mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

• Then list sensors available on device by calling `getSensorList()`

```java
List<Sensor> deviceSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);
```

• To list particular type, use `TYPE_GYROSCOPE, TYPE_GRAVITY`, etc

Checking if Phone has at least one of particular Sensor Type

- Device may have multiple sensors of a particular type.
  - E.g. multiple magnetometers
- If multiple sensors of a given type exist, one of them must be designated “the default sensor” of that type
- To determine if specific sensor type exists use `getDefaultSensor()`
- **Example:** To check whether device has at least one magnetometer

```java
private SensorManager mSensorManager;
...

mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if (mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null){
    // Success! There's a magnetometer.
}
else {
    // Failure! No magnetometer.
}
```
Example: Monitoring Light Sensor Data

- **Goal:** Monitor light sensor data using `onSensorChanged()`, display it in a `TextView` defined in `main.xml`

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

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

        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.
    }
}
```

- Create instance of Sensor manager
- Get default Light sensor
- Called by Android system when accuracy of sensor being monitored changes
Example: Monitoring Light Sensor Data (Contd)

```java
@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.
}

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

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

Called by Android system to report new sensor value
Provides SensorEvent object containing new sensor data
Get new light sensor value
Register sensor when app becomes visible
Unregister sensor if app is no longer visible to reduce battery drain
Handling Different Sensor Configurations

- Different phones have different sensors built in
  - E.g. Motorola Xoom has pressure sensor, Samsung Nexus S doesn’t
- If app uses a specific sensor, how to ensure this sensor exists on target device?
- Two options
  - **Option 1:** Detect device sensors at runtime, enable/disable app features as appropriate
  - **Option 2:** Use AndroidManifest.xml entries to ensure that only devices possessing required sensor can see app on Google Play
    - E.g. following manifest entry in AndroidManifest ensures that only devices with accelerometers will see this app on Google Play

```
<uses-feature android:name="android.hardware.sensor.accelerometer"
              android:required="true" />
```
Option 1: Detecting Sensors at Runtime

- Following code checks if device has at least one pressure sensor

```java
private SensorManager mSensorManager;
...

mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if (mSensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE) != null) {
    // Success! There's a pressure sensor.
}
else {
    // Failure! No pressure sensor.
}
```
Example Step Counter App

- **Goal:** Track user’s steps, display it in TextView
- **Note:** Phone hardware must support step counting

```java
package com.starboardland.pedometer;

import android.app.Activity;
import android.content.Context;
import android.hardware.*;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;

public class CounterActivity extends Activity implements SensorEventListener {
    private SensorManager sensorManager;
    private TextView count;
    boolean activityRunning;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        count = (TextView) findViewById(R.id.count);

        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    }
```
Example Step Counter App (Contd)

```java
@Override
protected void onResume() {
    super.onResume();
    activityRunning = true;
    Sensor countSensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
    if (countSensor != null) {
        sensorManager.registerListener(this, countSensor, SensorManager.SENSOR_DELAY_UI);
    } else {
        Toast.makeText(this, "Count sensor not available!", Toast.LENGTH_LONG).show();
    }
}

@Override
protected void onPause() {
    super.onPause();
    activityRunning = false;
    // if you unregister the last listener, the hardware will stop detecting step events
    // sensorManager.unregisterListener(this);
}
```

https://theelfismike.wordpress.com/2013/11/10/android-4-4-kitkat-step-detector-code/
Example Step Counter App (Contd)

```java
@Override
public void onSensorChanged(SensorEvent event) {
    if (activityRunning) {
        count.setText(String.valueOf(event.values[0]));
    }
}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
}
```
References

- John Corpuz, 10 Best Location Aware Apps
- Liane Cassavoy, 21 Awesome GPS and Location-Aware Apps for Android,
- Head First Android
- Android Nerd Ranch, 2nd edition
- Busy Coder’s guide to Android version 6.3
- CS 65/165 slides, Dartmouth College, Spring 2014
- CS 371M slides, U of Texas Austin, Spring 2014