

# Desarrollo de aplicaciones e introducción a juegos con Android Studio y AndEngine.

## Tema 7: Manejo de sensores y localización en Android



**Jose Manuel Fornés Rumbao**

Departamento de Ingeniería Telemática  
Universidad de Sevilla

[jfornes@us.es](mailto:jfornes@us.es)

# Índice

1. Introducción
2. Base de los sensores
3. Categorías y características de los sensores
4. Sistema de coordenadas
5. Estudio de sensores:
  1. Acelerómetro
  2. Campo magnético
  3. Orientación
  4. Giroscopio
  5. Otros sensores
  6. Aplicaciones
6. API de sensores
  1. Pasos para la programación
  2. Sensores predefinidos
  3. Ejemplos de código para sensores
7. Localización
  1. Clases e Interfaces
  2. Localización y mapas
  3. Google Maps Library
  4. Ejemplo de código de localización

# Introducción

- Los sensores en los dispositivos móviles han abierto un abanico de nuevas aplicaciones. Miden movimiento, orientación y varias condiciones ambientales
- Los sensores más importantes son:
  - Acelerómetro
  - Orientación
  - Luz
  - Presión
  - Campo magnético
  - Giroscopio
  - Proximidad
  - Temperatura
- Se manejan de forma homogénea:
  - Todos devuelven valores escalares (algunos en 3 ejes)
  - Todos tienen un consumo, resolución, periodo de muestreo, etc.
- No se suelen incluir en esta línea, otros sensores como:
  - Micrófono, cámara o incluso GPS

# Base de los sensores

- Hay sensores basados en hardware:
  - Son componentes físicos integrados en el terminal
  - Obtienen los datos midiendo directamente propiedades ambientales, tales como aceleración, fuerza del campo geomagnético, o cambio angular
  - Ejemplo: Luz, proximidad, presión, temperatura interna, acelerómetro, giróscopo, campo magnético, humedad relativa, temperatura ambiental
- Hay sensores basados en software:
  - No disponen de elementos físicos aunque imitan a sensores basados en hardware
  - Obtienen sus datos de uno o más sensores basados en hardware y se les suele llamar sensores virtuales o sensores sintéticos
  - Ejemplo: sensor de aceleración lineal, vector de rotación, gravedad, orientación

# Categorías de sensores

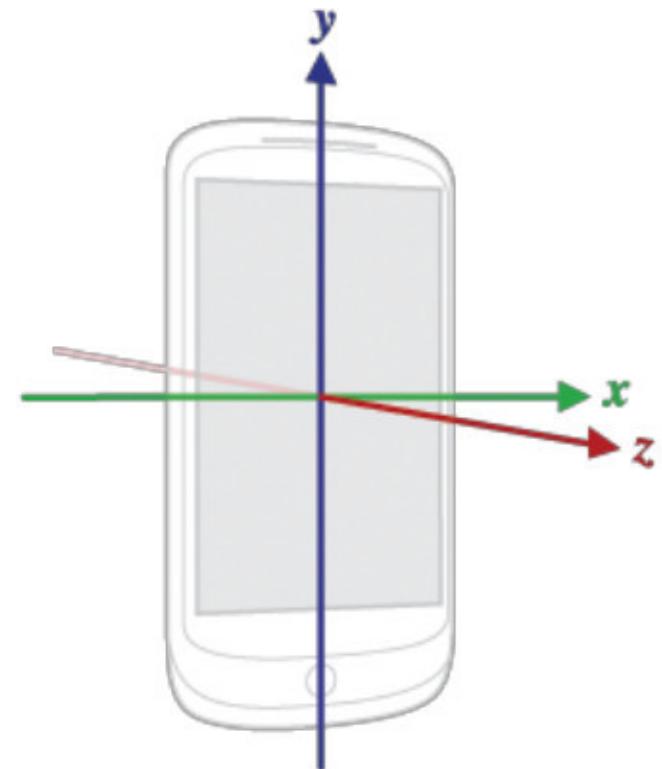
- Sensores de movimiento: Miden fuerzas de aceleración y fuerzas rotacionales alrededor de tres ejes
  - Acelerómetros, sensores de gravedad, giróscopos, etc.
- Sensores ambientales: miden varios parámetros ambientales, tales como temperatura y presión del aire ambiental, iluminación y humedad
  - Barómetros, fotómetros y termómetros
- Sensores de posición: miden la posición física de un dispositivo
  - Sensores de orientación y magnetómetros

# Características de los sensores

- **Sensor binario:** reporta sólo uno de dos valores posibles.
  - La mayoría de los sensores de proximidad y algunos sensores de luz sólo informan de una medida lejos o cerca.
- **Sensor continuo:** mide cualquier rango de valores entre su mínimo y su máximo
- **Rango dinámico:** es el rango de valores que un sensor puede medir:
  - Sensor de luz: entre 1 y 10.000 lux
- **Saturación:** sucede cuando la medición es mayor que el mayor valor medible. Cuando no hay estímulo, el valor vuelve a cero
- **Frecuencia de muestreo:** es la inversa del tiempo entre muestras medidas en Hertzios. Se puede obtener cuál es el mínimo valor entre muestras y puede variar de un dispositivo a otro

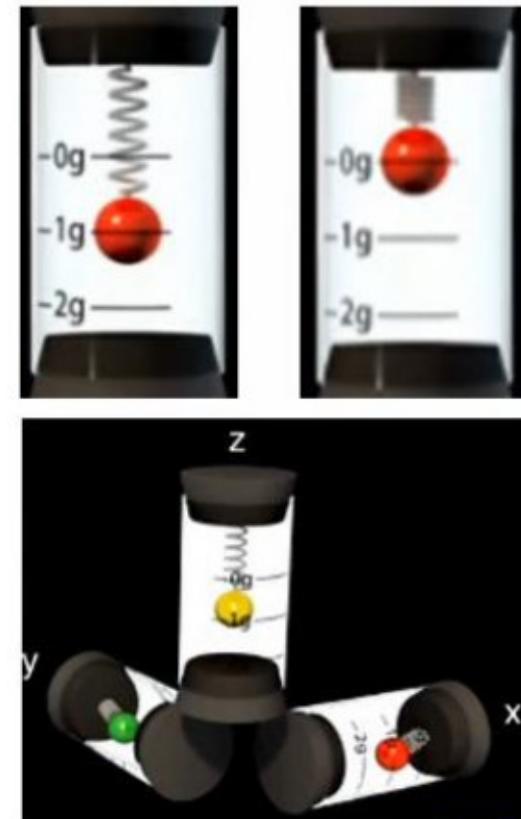
# Sistema de coordenadas de los sensores

- Cuando un dispositivo se mantiene en su orientación por defecto
  - El eje X es horizontal y apunta a la derecha
  - El eje Y es vertical y apunta hacia arriba
  - El eje Z apunta hacia afuera del plano de la pantalla



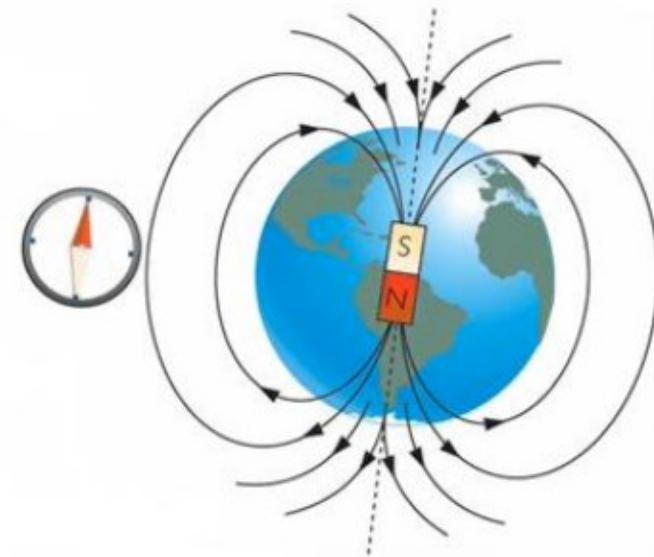
# Acelerómetro

- Detecta: las fuerzas de aceleración a la que se somete una masa. Esta fuerza se ve alterada por los cambios de velocidad de la masa
- Utilidades prácticas:
  - Medir cambios de velocidad de un objeto
  - Cuando el teléfono está en reposo, medir la inclinación del teléfono con respecto al centro de masa de la tierra.
- Cómo funciona: midiendo el desplazamiento de una masa
  - Ojo. Este dispositivo no mide aceleraciones (cambios de velocidad) sino fuerzas de aceleración.
  - Por ejemplo: un objeto en reposo en la superficie de la tierra se ve sometido a una fuerza ( $g$ ), sin embargo en caída libre esta fuerza sería 0.
  - Se construye en 3 ejes
  - Inconveniente: Si giro a velocidad constante sobre el eje z no se apreciará cambio en este sensor



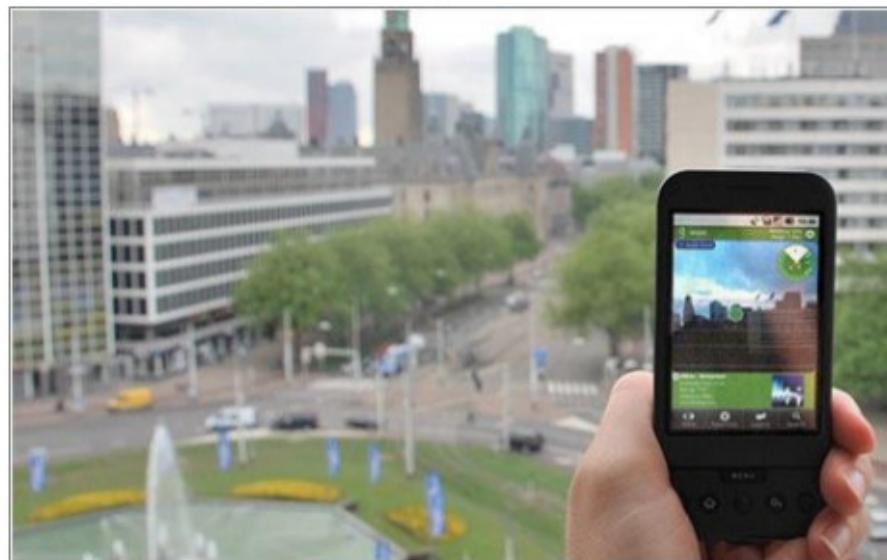
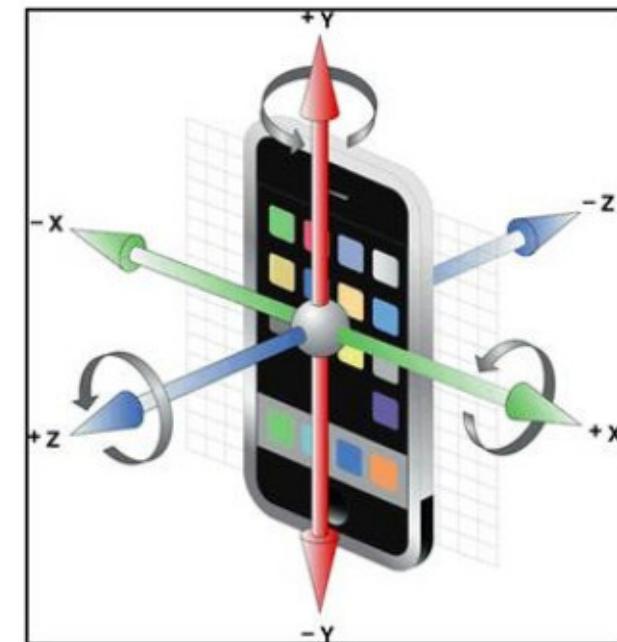
# Campo magnético

- Detecta: La fuerza y dirección de un campo magnético
- Utilidades prácticas:
  - Brújula
  - Detector de metales o corrientes eléctricas
- Cómo funciona:
  - utilizando materiales magnetoresistivos
  - Su resistencia eléctrica cambia según el ángulo de incidencia de un campo magnético
  - Se construye en 3 ejes
- Inconveniente: Si el dispositivo gira tomando como eje una línea de campo magnético no se apreciará cambio en este sensor



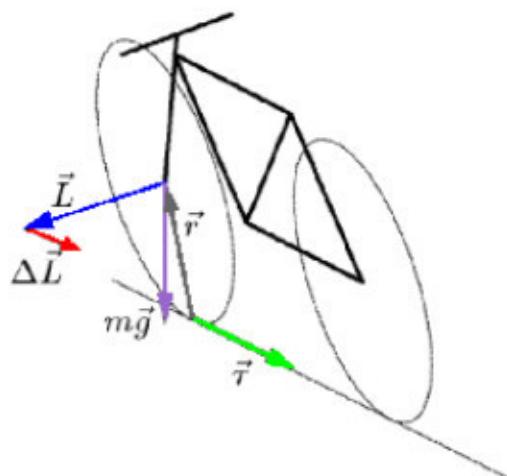
# Orientación

- Detecta: La orientación del dispositivo
- Utilidades prácticas:
  - Realidad aumentada
- Cómo funciona:
  - Combinando acelerómetro y magnetómetro
  - Un acelerómetro permite obtener un vector al centro de masa de la tierra, mientras que un magnetómetro permite obtener un vector al polo norte magnético.
  - Combinando estas dos informaciones podemos conocer a dónde apunta el terminal en sus 3 ejes



# Giroscopio

- Detecta: La rotación a la que se somete el dispositivo
- Utilidades prácticas:
  - Mejorar las prestaciones de un sensor de orientación
  - Medir la velocidad de giro (navegador automóvil)
- Cómo funciona:
  - Aprovechando el principio de conservación de la cantidad de movimiento de un cuerpo que puede girar sobre sí mismo



# Otros sensores (I)

- Sensor de luz
  - Detecta: La luz ambiental
  - Utilidades prácticas:
    - Ajustar el brillo de la pantalla
  - Cómo funciona:
    - Fotorresistencias
- Sensor de proximidad
  - Detecta: si hay un objeto a menos de 5 cm
  - Utilidades prácticas:
    - Apagar la pantalla cuando hablamos por teléfono
  - Cómo funciona: Emite luz infrarroja que mide con el sensor de luz



# Otros sensores (II)

- Sensor de presión
  - Detecta: La presión atmosférica
  - Utilidades prácticas:
    - Altímetro, barómetro
  - Cómo funciona:
    - Piezoresistencias
- Sensor de temperatura ambiental
  - Detecta: la temperatura ambiental
  - Utilidades prácticas:
    - Previsión meteorológicas
  - Cómo funciona: Transistor integrado. Aprovecha la dependencia de la unión NP con la temperatura



# Aplicaciones para experimentar

- Android: Z- Device Test



- iOS: Sensor Monitor



# API de sensores (I)

- La API ofrece varias clases e interfaces para ayudar a desarrollar una amplia variedad de tareas relacionadas con los sensores:
  - Determinar qué sensores están disponibles en un dispositivo
  - Determinar las capacidades individuales de un sensor tales como su rango máximo, fabricante, requerimientos de potencia y resolución
  - Adquirir datos brutos de sensores y definir la tasa mínima a la cual se adquieren los datos de los sensores
  - Registrar y desregarstrar a los Listeners de sensores de eventos que monitorizan los cambios en los sensores
- El punto de entrada a la API es la clase `SensorManager`, que permite a una aplicación solicitar información de los sensores y registrarse para recibir datos de los sensores.
- Esta clase también ofrece varias constantes que se usan para informar de la precisión del sensor, tasas de adquisición de datos y calibración de sensores.

## API de sensores (II)

- La clase `Sensor` se utiliza para crear una instancia de un sensor específico.
- Esta clase ofrece varios métodos para determinar las capacidades del sensor:
  - Rango máximo
  - Retardo mínimo
  - Nombre
  - Potencia
  - Resolución
  - Tipo
  - Fabricante
  - Versión

## API de sensores (III)

- La clase `SensorEvent`, la utiliza el sistema para crear un objeto de evento de un sensor que ofrece información sobre un evento del sensor
- El objeto de evento de sensor incluye la siguiente información:
  - Los datos brutos de la medida del sensor
  - El tipo de sensor que generó el evento
  - La precisión de los datos
  - Una marca de tiempo del evento
- El interfaz `SensorEventListener`, se utiliza para recibir las notificaciones (eventos de sensor) cuando cambia los valores de los sensores o cuando cambia su precisión. Esto se hace mediante dos funciones de callback:
  - `onAccuracyChanged(Sensor sensor, int accuracy)` :Se llama cuando cambia la precisión del sensor
  - `onSensorChanged(SensorEvent event)` :Se llama cuando el valor del sensor ha cambiado

## API de sensores (IV)

- Cuando se registra un Listener, se especifica el retardo o la tasa de medidas para el Listener. Los valores predefinidos son:
  - SENSOR\_DELAY\_FASTEST (0 ms)
  - SENSOR\_DELAY\_GAME (20ms)
  - SENSOR\_DELAY\_UI (67 ms. Adecuado para funciones de interfaz de usuario normales como la rotación de la pantalla)
  - SENSOR\_DELAY\_NORMAL (200 ms. El valor por defecto)
- Se pueden pasar otros valores distintos si no se usan estas constantes
- Los valores temporales son simplemente intenciones de Android. El hardware de los sensores puede limitarlo o incluso las capacidades del terminal. Los eventos pueden recibirse tanto más rápidos como más lentos
- Para determinar el mínimo retardo permitido entre dos eventos en microsegundos, se usa el método `Sensor.getMinDelay()`. Devuelve cero cuando los datos sólo se miden cuando hay cambios como en el sensor de proximidad

# API de sensores (V)

- La precisión de un SensorEvent viene determinada por SensorEvent.accuracy:
  - SensorManager.SENSOR\_STATUS\_ACCURACY\_HIGH
  - SensorManager.SENSOR\_STATUS\_ACCURACY\_MEDIUM
  - SensorManager.SENSOR\_STATUS\_ACCURACY\_LOW
  - SensorManager.SENSOR\_STATUS\_UNRELIABLE

El estado UNRELIABLE hace referencia a que no puede dar precisión debido a por ejemplo se necesita una calibración o es un sensor binario que simplemente informa de que se ha llegado al umbral
- Los datos van en SensorEvent.values
- La marca de tiempo en milisegundos va en SensorEvent.timestamp

# Pasos para la programación

1. Se puede indicar en el fichero AndroidManifest la intención de usar algún sensor. Es sólo informativo para declarar qué sensores se usan y así poder saber si es o no compatible con un determinado dispositivo:

```
<uses-feature android:name="android.hardware.sensor.accelerometer"
    android:required="true" />
<uses-feature android:name="android.hardware.sensor.compass"
    android:required="false" />
```

2. Acceder al SensorManager mediante  
getSystemService (SENSOR\_SERVICE)
3. Acceder al Sensor mediante el método  
sensorManager.getDefaultSensor ()
4. Una vez adquirido el sensor, registramos el objeto  
SensorEventListener en él. Este Listener será informado  
si hay cambios en los eventos
5. Para evitar el uso innecesario de batería, se registra el  
Listener en el método onResume () y se desregistra en el  
método onPause (), de manera que sólo funcionen cuando el  
usuario puede interactuar con la aplicación.

# Sensores predefinidos

Sensor	Android 4.0 (API Level 14)	Android 2.3 (API Level 9)	Android 2.2 (API Level 8)	Android 1.5 (API Level 3)
TYPE_ACCELEROMETER	Yes	Yes	Yes	Yes
TYPE_AMBIENT_TEMPERATURE	Yes	n/a	n/a	n/a
TYPE_GRAVITY	Yes	Yes	n/a	n/a
TYPE_GYROSCOPE	Yes	Yes	n/a <sup>1</sup>	n/a <sup>1</sup>
TYPE_LIGHT	Yes	Yes	Yes	Yes
TYPE_LINEAR_ACCELERATION	Yes	Yes	n/a	n/a
TYPE_MAGNETIC_FIELD	Yes	Yes	Yes	Yes
TYPE_ORIENTATION	Yes <sup>2</sup>	Yes <sup>2</sup>	Yes <sup>2</sup>	Yes
TYPE_PRESSURE	Yes	Yes	n/a <sup>1</sup>	n/a <sup>1</sup>
TYPE_PROXIMITY	Yes	Yes	Yes	Yes
TYPE_RELATIVE_HUMIDITY	Yes	n/a	n/a	n/a
TYPE_ROTATION_VECTOR	Yes	Yes	n/a	n/a
TYPE_TEMPERATURE	Yes <sup>2</sup>	Yes	Yes	Yes

<sup>1</sup> This sensor type was added in Android 1.5 (API Level 3), but it was not available for use until Android 2.3 (API Level 9).

<sup>2</sup> This sensor is available, but it has been deprecated.

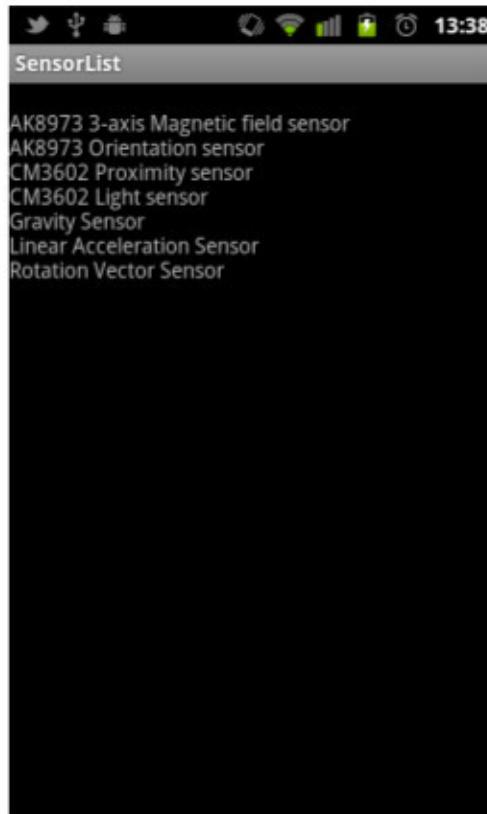
# Ejemplo: listado de todos los sensores

1. Obtener la referencia del servicio de sensores:

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

2. Obtener la lista de todos los sensores de un dispositivo

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



# Ejemplo: uso del acelerómetro (I)

1. Obtener la referencia del servicio de sensores y una instancia de un sensor:

```
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);  
mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

2. Registrar un Listener

```
mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_GAME);
```

3. Sobrescribir la función de callback que se llama en el Listener para obtener los datos

```
@Override  
public void onSensorChanged(SensorEvent event) {  
    float x = event.values[0];  
    float y = event.values[1];  
    float z = event.values[2];  
}
```

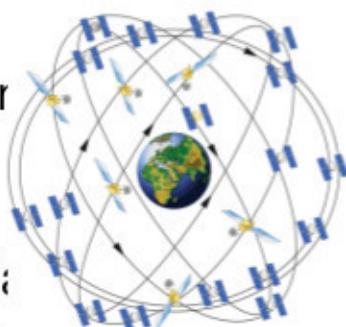
# Ejemplo: uso del acelerómetro (II)

- Código

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_acc_main);  
    mytx=(TextView)this.findViewById(R.id.tx);  
    myty=(TextView)this.findViewById(R.id.ty);  
    mytz=(TextView)this.findViewById(R.id.tz);  
    sm=(SensorManager)this.getSystemService(Context.SENSOR_SERVICE);  
    mysensor=sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);  
}  
private final SensorEventListener mysensorlistenr=new SensorEventListener(){  
  
    public void onAccuracyChanged(Sensor sensor, int accuracy) {  
        //No hace falta inicialmente  
    }  
    public void onSensorChanged(SensorEvent event) {  
        mytx.setText(String.valueOf(event.values[0]));  
        myty.setText(String.valueOf(event.values[1]));  
        mytz.setText(String.valueOf(event.values[2]));  
    }  
};  
protected void onResume() {  
    super.onResume();  
    sm.registerListener(mysensorlistenr, mysensor,SensorManagerSENSOR_DELAY_NORMAL);  
}  
protected void onPause() {  
    super.onPause();  
    sm.unregisterListener(mysensorlistenr);  
}
```

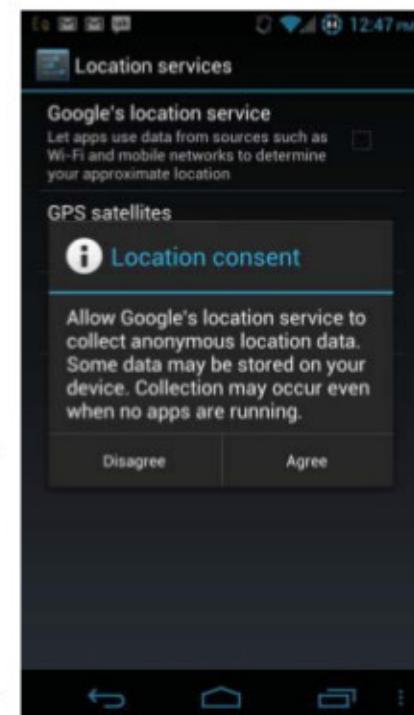
# Localización (I)

- Es el mecanismo para obtener la ubicación del dispositivo
- Permite a las aplicaciones ser más inteligentes y entregar información más interesante al usuario
- Android hace uso de diferentes métodos para ofrecer información de localización a las aplicaciones. Estos mecanismos se llaman Proveedores de localización y la aplicación puede seleccionarlo
- Proveedor GPS
  - El receptor GPS calcula la posición en función de la información que obtiene de los 27 posibles satélites que orbitan la tierra.
  - Tiene inconvenientes sobre todo para entorno móvil: sólo algunos satélites están visibles en cada momento, antes de calcular la posición deben verse varios satélites y conocer su posición. En consecuencia lleva mucho tiempo el cálculo de la posición
  - A-GPS (Assisted GPS) es una mejora en móviles ya que le llega por la red información relativa de dónde están los satélites en ese momento, acelerando el cálculo
  - S-GPS (Simultaneous GPS) son dispositivos con hardware adicional que permiten comunicar con el satélite a la vez que funcionan con la red terrestre. Esto acelera la adquisición de datos
  - Limitaciones: No funciona en interiores y tiene problemas cuando hay muchos obstáculos a la señal como árboles, edificios altos, etc.



# Localización (II)

- Proveedor de Red
  - Puntos de acceso WIFI
    - Necesita que la WIFI esté activada en el terminal. Consumo menor batería que el GPS
    - El terminal mide la potencia de la red detectada y la MAC del punto de acceso y le consulta a Google.
    - Google mantiene una base de datos de estos puntos de acceso por el mundo. Esta base de datos la obtiene mediante la colaboración de los terminales que tienen habilitada esta opción
    - Funciona en sitios donde el GPS no llega
    - Limitaciones: Hay puntos de acceso que no permiten el mapeo. Además si se cambia el punto de acceso de sitio puede dar lugar a imprecisiones
  - Cell-ID
    - Se usa la red móvil celular. El móvil debe estar conectado a una estación base con un ID único. Cuando se mueve puede cambiar de estación base.
    - El terminal colabora enviando el ID y la posición GPS a Google que mantiene una base de datos

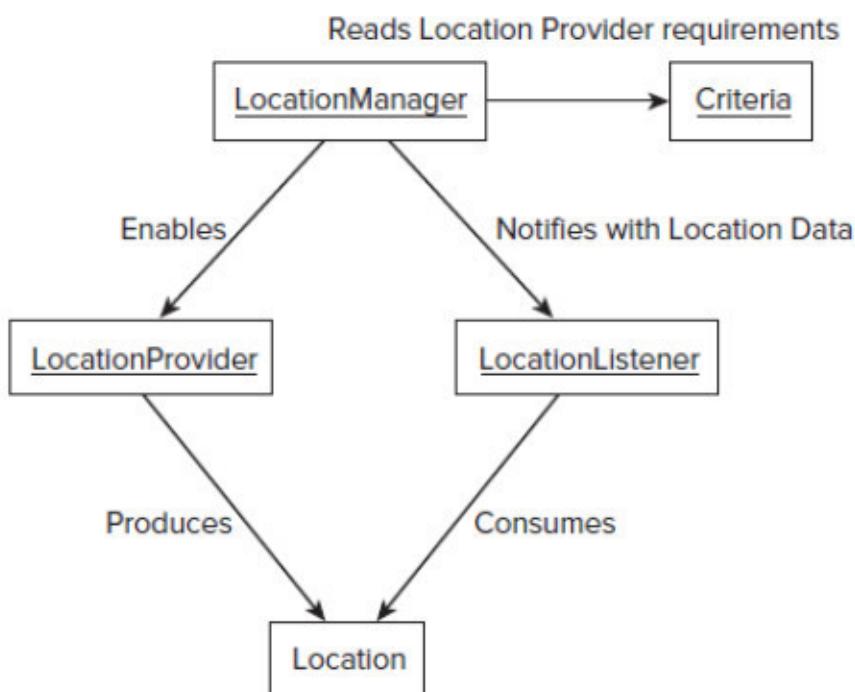


# Localización (III)

- Se deben dar los correspondientes permisos a las aplicaciones según el tipo de proveedor a usar: (GPS y Redes respectivamente)

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

- Las clases usadas son: LocationManager, LocationProvider, Location, Criteria, y el interfaz usado es: LocationListener



# Clases e interfaces (I)

- LocationManager
  - Es el punto de entrada principal al servicio. Permite a una aplicación indicar cuándo quiere recibir información actualizada de localización, cuándo ya no quiere más información, conocer y determinar proveedores de localización, conocer el estado del GPS, etc.
  - Se invoca utilizando:  
`getSystemService(Context.LOCATION_SERVICE)`
  - Este método devuelve una instancia de LocationManager
  - A partir de aquí se puede:
    - Lanzar consultas a los LocationProviders para conocer la última posición del dispositivo
    - Registrar un componente para que reciba actualizaciones periódicas de la ubicación de un LocationProvider
    - Registrar un Intent para que se lance si el dispositivo entra en un radio de proximidad de una petición

# Clases e interfaces (II)

- LocationProvider
  - Es una abstracción de las diferentes fuentes de información de localización.
  - De esta forma, aunque las fuentes y mecanismos de información sean distintos, a la aplicación le llega de forma homogénea
- Location
  - Encapsula los datos de localización actuales ofrecidos a la aplicación desde un proveedor de localización
  - Contiene datos cuantificables como latitud, longitud y altitud.
  - Sin embargo, hay proveedores de localización que no suministran todos los datos como por ejemplo la altitud. En este caso, este objeto tiene métodos para saber qué datos tiene contenidos

# Clases e interfaces (III)

- Criteria
  - Esta clase permite hacer consultas al LocationManager por proveedores de localización que contengan algunas características. De esta forma la aplicación no se preocupa de un proveedor en concreto sino que pide aquellos que cumplan unos criterios como: precisión, altitud, sin coste, potencia requerida, necesidad de velocidad, etc.
- LocationListener
  - Este interfaz contiene un grupo de métodos callback que son llamados en reacción a cambios en la posición actual del dispositivo o cambios en el estado del servicio de localización.
  - El LocationManager permite registrar/desregar un LocationListener para la aplicación
- PendingIntent
  - Es otro mecanismo para recibir información de localización. En este caso, en lugar de registrar un LocationListener, se registra un BroadcastReceiver con un filtro determinado, que funciona en background. Suele ser el mecanismo utilizado para aplicaciones que necesitan continuamente información de localización para realizar una navegación o seguimiento

# Localización y mapas

- Android utiliza la API de Google Maps para el desarrollo de aplicaciones basadas en localización
- Las clases de la API están ubicadas en el paquete com.google.android.maps
- Estas librerías deben instalarse de forma separada aunque hay un SDK con las librerías Android+Google Maps integradas
- Una forma de activarlas en el proyecto es seleccionando Google APIS en el SDK en lugar de usar sólo Android
- Los servicios de localización (GPS y redes) los aporta el dispositivo
- El componente principal de la localización es el LocationManager

# Google Maps Library

- Permiten que las aplicaciones puedan descargar, renderizar y cachear zonas de mapas, así como añadir capas, marcadores, etc.
- La clase principal es un MapView, subclase de ViewGroup, que dibuja un mapa con datos obtenidos del servicios de Google Maps
- Esta vista puede capturar eventos de la pantalla y realizar marcados, zoom, etc.
- Para utilizar los mapas será necesario obtener una API MAPS key.
- Es necesario registrarse en el servicio de Google Maps y aceptar las condiciones de servicio (indistintamente si es un entorno de pre o producción)
- Se realiza en dos partes:
  - Se registra la huella MD5 del certificado con el que vamos a firmar nuestra aplicación
  - Con ese MD5 el servicio proporciona una API KEY
  - Se debe introducir esa API KEY en cada MapView que utilicemos

# Ejemplo de localización (I)

```
public class MainLocalizacion extends Activity {

    private Button btnActualizar;
    private Button btnDesactivar;
    private TextView lblLatitud;
    private TextView lblLongitud;
    private TextView lblPrecision;
    private TextView lblEstado;
    private LocationManager locManager;
    private LocationListener locListener;

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

        btnActualizar = (Button) findViewById(R.id.BtnActualizar);
        btnDesactivar = (Button) findViewById(R.id.BtnDesactivar);
        lblLatitud = (TextView) findViewById(R.id.LblPosLatitud);
        lblLongitud = (TextView) findViewById(R.id.LblPosLongitud);
        lblPrecision = (TextView) findViewById(R.id.LblPosPrecision);
        lblEstado = (TextView) findViewById(R.id.LblEstado);

        btnActualizar.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                comenzarLocalizacion();
            }
        });
        btnDesactivar.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                locManager.removeUpdates(locListener);
            }
        });
    }
}
```

# Ejemplo de localización (II)

```
private void comenzarLocalizacion()
{
    //Obtenemos una referencia al LocationManager
    locManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    //Obtenemos la Última posición conocida
    Location loc = locManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
    //Mostramos la última posición conocida
    mostrarPosicion(loc);
    //Nos registramos para recibir actualizaciones de la posición
    locListener = new LocationListener() {
        public void onLocationChanged(Location location) {
            mostrarPosicion(location);
        }
        public void onProviderDisabled(String provider) {
            lblEstado.setText("Provider OFF");
        }
        public void onProviderEnabled(String provider) {
            lblEstado.setText("Provider ON ");
        }
        public void onStatusChanged(String provider, int status, Bundle extras) {
            Log.i("", "Provider Status: " + status);
            lblEstado.setText("Provider Status: " + status);
        }
    };
    locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 30000, 0, locListener);
}
```

# Ejemplo de localización (III)

```
private void mostrarPosicion(Location loc) {
    if(loc != null)
    {
        lblLatitud.setText("Latitud: " + String.valueOf(loc.getLatitude()));
        lblLongitud.setText("Longitud: " + String.valueOf(loc.getLongitude()));
        lblPrecision.setText("Precision: " + String.valueOf(loc.getAccuracy()));
        Log.i("", String.valueOf(loc.getLatitude()) + " - " +
String.valueOf(loc.getLongitude())));
    }
    else
    {
        lblLatitud.setText("Latitud: (sin_datos)");
        lblLongitud.setText("Longitud: (sin_datos)");
        lblPrecision.setText("Precision: (sin_datos)");
    }
}
```