La mayoría de los dispositivos con Android tienen sensores incorporados que miden el movimiento, la orientación y varias condiciones ambientales. Estos sensores son capaces de proporcionar datos en bruto con alta precisión y precisión, y son útiles si desea monitorear el movimiento o posicionamiento tridimensional del dispositivo, o si desea monitorear cambios en el entorno ambiental cerca de un dispositivo. Por ejemplo, un juego puede rastrear las lecturas del sensor de gravedad de un dispositivo para inferir gestos y movimientos complejos del usuario, como inclinación, movimiento, rotación o balanceo. Del mismo modo, una aplicación meteorológica podría usar el sensor de temperatura y el sensor de humedad de un dispositivo para calcular e informar el punto de rocío, o una aplicación de viaje podría usar el sensor de campo geomagnético y el acelerómetro para informar el rumbo de una brújula.

La plataforma Android admite tres amplias categorías de sensores:

- Sensores de movimiento

Estos sensores miden fuerzas de aceleración y fuerzas de rotación a lo largo de tres ejes. Esta categoría incluye acelerómetros, sensores de gravedad, giroscopios y sensores de vector giratorio.

- Sensores ambientales

Estos sensores miden varios parámetros ambientales, como la temperatura y presión del aire ambiente, la iluminación y la humedad. Esta categoría incluye barómetros, fotómetros y termómetros.

- Sensores de posición

Estos sensores miden la posición física de un dispositivo. Esta categoría incluye sensores de orientación y magnetómetros.

Puede acceder a los sensores disponibles en el dispositivo y adquirir datos sin procesar del sensor utilizando la API de sensor de Android. Este

framework proporciona varias clases e interfaces que lo ayudan a realizar una amplia variedad de tareas relacionadas con el sensor. Por ejemplo, puedes usar el marco del sensor para hacer lo siguiente:

- Determinar qué sensores están disponibles en un dispositivo.
- Determinar las capacidades de un sensor individual, como su rango máximo, fabricante, requisitos de potencia y resolución.
- Adquirir datos sin procesar del sensor y defina la velocidad mínima a la que adquiere los datos del sensor.
- Registrar y anular el registro de escuchas de eventos del sensor que supervisan los cambios del sensor.

## Trabajando con sensores estándar

Dentro del paquete `Android.hardware` del SDK de Android encontramos varias clases para trabajar con los sensores del dispositivo, estas son: `SensorManager`, Sensor, `SensorEvent` y la interfaz `SensorEventListener`. La principal de todas ellas es la clase `SensorManager`, ya que es la que nos permite comprobar los sensores disponibles, acceder a sus datos y registrar o eliminar `listeners` de eventos, entre otras acciones. Por lo tanto, lo primero que vamos a hacer es obtener una instancia de esta clase. Para esto tenemos que solicitar el servicio de acceso llamando al método `getSystemService`() e indicando como argumento la constante SENSOR_SERVICE. Tal y como se muestra en el siguiente ejemplo:

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

A partir de esta clase podemos obtener una referencia al sensor con el que queramos trabajar. Para esto usaremos el método `getDefaultSensor` del `SensorManager` y una constante de la clase Sensor que identifique el sensor que queremos, por ejemplo:

```
        mSensor =
mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

En la siguiente lista se muestra un resumen de los principales sensores que podemos utilizar, incluyendo el nombre de la constante que nos da acceso, la versión de la API en la que apareció y una pequeña descripción:

- TYPE_ACCELEROMETER: mide la aceleración en m/s2 a la que se somete el dispositivo en los tres ejes, x, y, z, incluyendo la fuerza de gravedad. Permite medir cambios de movimiento y de orientación.
- TYPE_AMBIENT_TEMPERATURE: mide la temperatura ambiente en grados Celsius.

- TYPE_GRAVITY: mide la fuera de gravedad en m/s2 que se ejerce sobre el dispositivo en los ejes x, y, z.
- TYPE_GYROSCOPE: giroscopio que proporciona la orientación del dispositivo en los tres ejes a partir de los cambios de orientación que sufre el dispositivo. Es capaz de reconocer movimientos que no reconoce el acelerómetro, como por ejemplo los giros en el eje Y(vertical).
- TYPE_LIGHT: detecta la iluminación ambiental en lux (lx o lumen/m2), para así poder modificar de forma automática el brillo de la pantalla.
- TYPE_LINEAR_ACELERATION: mide la aceleración en m/s2 a la que se somete el dispositivo en los ejes x, y, z sin incluir la fuerza de gravedad.
- TYPE_ORIENTATION: se trata de un sensor virtual que combina información de varios sensores para darnos la orientación del dispositivo en los tres ejes. Su uso está desaconsejado, en su lugar se recomienda obtener la orientación combinando manualmente la información del acelerómetro y de la brújula.
- TYPE_PRESSURE: mide la presión atmosférica a la que está sometido el dispositivo en hectopascales (hPa o mbar). Este sensor en realidad es un barómetro y permite calcular la altura y ayuda al posicionamiento del GPS.
- TYPE_PROXIMITY: detecta la proximidad de la pantalla del dispositivo a otros objetos en centímetros. Se utiliza habitualmente para apagar la pantalla cuando situamos el móvil cerca de nuestra oreja para hablar.

- TYPE_RELATIVE_HUMIDITY: Mide la humedad ambiental en porcentaje (%). Proporciona información sobre la humedad absoluta, relativa y el punto de condensación.
- TYPE_ROTATION_VECTOR: Proporciona la orientación del dispositivo a partir del vector de rotación en los tres ejes (x, y ,z). Para su cálculo combina información de otros sensores como el acelerómetro, el magnetómetro y el giroscopio.
- TYPE_AMBIENT_TEMPERATURE: Mide la temperatura ambiente en grados Celsius.

Hay pocos dispositivos que tengan todos estos sensores. La mayoría cuenta con acelerómetro y magnetómetro, pero muy pocos disponen de barómetro. Además, como se puede ver en el listado anterior, su disponibilidad no solo depende del tipo de dispositivo sino también de la versión de Android que tenga. También hay dos sensores que están en desuso ya que han sido sustituidos por nuevas versiones. Por lo tanto, es importante tener todo esto en cuenta y comprobar si un sensor está disponible o no antes de utilizarlo esto lo podemos hacer de forma muy sencilla mediante el método `getDefaultSensor()` que hemos visto antes. En caso de que devuelva `null` significará que el sensor no está disponible. A continuación, se puede ver un ejemplo en el que se comprueba la disponibilidad del sensor de Luz:

```
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
if(mSensor != null){

}else{

}
```

También podemos utilizar el método `getSensorList()` con la constantes TYPE_ALL para obtener una lista de todos los sensores disponibles en el dispositivo:

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

for(Sensor sensor : deviceSensors){
    Log.i("TAG", sensor.getName());
}
```

Algunos dispositivos cuentan con varios sensores de un mismo tipo, en esos casos si usamos el método `getDefaultSensor` solo obtendremos uno, el que esté marcado como sensor por defecto para la clase indicada. Si queremos acceder a los otros podemos utilizar el método que hemos visto antes, `getSensorList()`, indicando que nos devuelva la lista de sensores del tipo que nos interese, por ejemplo:

manager.getSensorList(Sensor.TYPE_ACCELEROMETER).

La clase Sensor proporciona una serie de métodos para obtener información sobre el sensor seleccionado y sus capacidades. Como ya hemos visto en el ejemplo anterior, con `getName()` podemos obtener el nombre del sensor, pero además tenemos otros métodos como `getVendor()` para consultar el fabricante, `getVersion()` para la versión del sensor, `getType()` para el tipo de sensor, `getMaximumRange()` para consultar el rango máximo que nos puede devolver una lectura, `getResolution()`, para saber la resolución de las unidades y `getPower()` que nos devolverá la potencia en miliamperios(mA) consumida por el sensor mientras esté en uso.

## Especificación de sensores

Dependiendo del tipo de aplicación que estemos desarrollando, es posible que el uso de un determinado sensor sea indispensable o, por el contrario, que lo podamos suplir de alguna forma. Por ejemplo, si estamos haciendo una aplicación para medir la temperatura ambiental necesitaremos dicho sensor, por lo que en los dispositivos que no lo tengan, la aplicación directamente no funcionará y no servirá para nada. En estos casos, para evitar que haya usuarios que instalen la aplicación y no les funcione, lo que además dará mala imagen y provocará que la puntúen mal, tenemos que filtrar los dispositivos que no cuenten con el sensor requerido. Esto lo podemos conseguir mediante el uso de la etiqueta `<uses-feature>` del `Manifest`, por ejemplo:

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

En esta etiqueta tenemos que indicar el sensor que necesitamos mediante el atributo name (que podemos encontrar dentro del paquete `Android.hardware.sensor`) y además establecer también el atributo `Android:required=”true”.` Al añadir esta línea al `Manifest` conseguimos que la aplicación solo se muestre en Google Play para los dispositivos que cuenten con el sensor indicado.

Es recomendable añadir también al `Manifest` los sensores no requeridos que utilice la aplicación, pero marcándolos como no requeridos. `(required=”false”).` Esto no es obligatorio, y si no e pone funcionará igual, pero es una buena práctica para realizar un seguimiento de todas las características que utiliza la aplicación. Es importante recordar que, en este último caso, tendremos que comprobar mediante código si el sensor está disponible o no.

## Escuchando eventos del sensor

Una vez tenemos una referencia al sensor, el siguiente paso es crear un `Listener` que compruebe de forma periódica sus lecturas. Para esto podemos hacer que la clase actual implemente la interfaz `SensorEventListener` o crear una nueva clase que lo haga como en el siguiente ejemplo:

```
@Override
 public void onSensorChanged(SensorEvent sensorEvent) {



}

@Override
public void onAccuracyChanged(Sensor sensor, int i) {

}
```

Esta interfaz nos obligue a implementar dos métodos: `onSensorChanged` que se llamará para notificar una nueva lectura del sensor y `onAccuracyChanged` que nos indicará un cambio en la precisión de las lecturas. La precisión podrá ser una de las siguientes constantes de la clase `SensorManager`: SENSOR_STATUS_ACCURACY_LOW, SENSOR_STATUS_ACCURACY_MEDIUM, SENSOR_STATUS_ACCURACY_HIGH o SENSOR_STATUS_UNRELIABLE.

Solo nos falta registrar el `listener` (mediante el método `registerListener` de `SensorManager`) para que reciba los eventos del sensor. A continuación, podemos ver un ejemplo de cómo realizar esto:

El método de registro recibe tres parámetros: la instancia del `Listener`, el sensor y la periodicidad con la que se solicitarán los datos. Podemos utilizar una de las siguientes cuatro constantes de la clase `SensorManager` para indicar la frecuencia de actualización, pero debemos tener en cuenta que cuanto mayor sea la frecuencia, más recursos consumirá nuestra aplicación:

- SENSOR_DELAY_NORMAL: Esta es la tasa de actualización definida por defecto (0.2 segundos).
- SENSOR_DELAY_UI: Los datos se actualizan a una velocidad suficiente para utilizarlos en la interfaz de usuario (0.06 segundos).
- SENSOR_DELAY_GAME: Los datos se actualizan a una velocidad suficiente para ser utilizados en videojuegos (20000 microsegundos o 0.02 segundos).
- SENSOR_DELAY_FASTER: Los datos se actualizan tan rápido como pueda el dispositivo (0 microsegundos)

Una vez definido y registrado el `listener`, al ejecutar la aplicación comenzaremos a recibir actualizaciones con las lecturas del sensor a través del parámetro `SensorEvent` del método `onSensorChanged`. En la propiedad `values` (de tipo array) de la clase `SensorEvent` encontraremos los valores recogidos. Este array podrá contener entre uno y tres elementos dependiendo del tipo de sensor. Todos los sensores de orientación y aceleración nos devolverán tres elementos, y los sensores de luz, proximidad, presión atmosférica, temperatura y humedad, que devolverán un único valor.

Cuando hayamos terminado de trabajar con el sensor es importante que desconectemos el `Listener` mediante el método `unregisterListener` para evitar que se malgasten recursos:

```
mSensorManager.unregisterListener(this);
```

Además, **es recomendable** quitar y volver a poner los `listeners` de los sensores cada vez que se pause y se reanude la aplicación, utilizando para ello los métodos `onPause` y `onResume` de nuestra actividad.