La Clase `Sensor`

CONCEPTOS

Antecedentes.

La mayoría de los dispositivos con Android han incorporado en los sensores que miden el movimiento, la orientación y otras condiciones ambientales. Estos sensores proporcionan datos en bruto con una gran precisión y exactitud, y son útiles si se desea supervisar el movimiento del dispositivo en tres dimensiones o localización, o si desea monitorear los cambios en el entorno ambiental cerca de un dispositivo.

La plataforma Android es compatible con tres amplias categorías de sensores:

- Sensores de movimiento: Miden las fuerzas de aceleración y las fuerzas de rotación a lo largo de tres ejes. Esta categoría incluye los acelerómetros, sensores de gravedad, giroscopios y sensores de rotación del vector.
- Sensores ambientales: Miden diferentes parámetros ambientales, como la temperatura del aire ambiental, presión, iluminación, y humedad. Esta categoría incluye barómetros, fotómetros, y termómetros.
- Sensores de posición: Miden la posición física de un dispositivo. Esta categoría incluye sensores de orientación y magnetómetros.

Se puede acceder a los sensores disponibles en el dispositivo y adquirir datos crudos del sensor mediante el uso de la jerarquía de la clase Sensor de Android, la cual herramientas que ayudan a realizar una amplia variedad de tareas relacionados con los sensores. Por ejemplo:

- Determinar cuáles sensores están disponibles en un dispositivo.
- Determinar las capacidades de un sensor individual, alcance máximo, fabricante, los requisitos de energía y resolución.
- Adquirir datos crudos del sensor y definir la velocidad mínima a la que adquiere datos del sensor.
- Registrar y eliminar detectores de eventos de sensores que monitorizan los cambios de sensores.

El conjunto de recursos de los sensores de Android permite acceder a muchos tipos de sensores. Algunos de estos sensores están basados en hardware y algunos son basados en software. Los sensores basados en hardware son componentes físicos integrados en el dispositivo portátil o tableta. Sus datos se derivan midiendo directamente las propiedades ambientales específicas, como la aceleración, la fuerza del campo geomagnético, o el cambio angular. Los sensores basados en software no son dispositivos físicos, a pesar de que imitan a los sensores basados en hardware. Los sensores basados en software derivan sus datos de uno o más de los sensores basados en hardware y, a veces se llaman sensores virtuales o sensores sintéticos. El sensor de aceleración lineal y el sensor de la gravedad, son ejemplos de sensores basados en software.

La **Tabla 1** resume los sensores que son compatibles con la plataforma Android.

Sensor	Tipo	Descripción	Uso común
`TYPE_ACCELEROMETER`	Hardware	Mide la aceleración en m/s2 que se aplica al dispositivo en los tres ejes físicos (x, y, and z), incluyendo la fuerza de gravedad.	Detecta movimiento (agitado, inclinado, etc.)
`TYPE_AMBIENT_TEMPERATURE`	Hardware	Mide temperatura ambiental en grados Celsius (°C).	Monitorea temperatura del aire.
`TYPE_GRAVITY`	Software o Hardware	Mide la fuerza de gravedad en m/s2 que se aplica al dispositivo en los tres ejes físicos (x, y, z).	Detecta movimiento (agitado, inclinado, etc.)

TYPE_GYROSCOPE	Hardware	Mide la rotación en rad/s alrededor de cada uno de los ejes físicos (x, y, and z).	Detecta rotación (giro, vueltas, etc.)
TYPE_LIGHT	Hardware	Mide el nivel de luz ambiental (iluminación) en lx.	Controla el brillo de la pantalla.
TYPE_LINEAR_ACCELERATION	Software o Hardware	Mide la aceleración en m/s2 que se aplica a un dispositivo en cada uno de los tres ejes físicos (x, y, and z), excluyendo la fuerza de gravedad.	Monitorea la aceleración a lo largo de un sólo eje.
TYPE_MAGNETIC_FIELD	Hardware	Mide el campo geomagnético ambiental de los tres ejes físicos (x, y, z) en μT.	Crea una brújula.
TYPE_ORIENTATION	Software	Mide los grados de rotación que realiza alrededor de los tres ejes físicos (x, y, z). Pertenece a la API nivel 3 por lo que obtiene una matriz de inclinación y una matriz de rotación del dispositivo utilizando los sensores de gravedad y de campo electromagnético en conjunto con el método getRotationMatrix().	Determina la posición del dispositivo.
TYPE_PRESSURE	Hardware	Mide la presión del aire ambiental en hPa o mbar.	Monitorea los cambios de la presión del aire.
TYPE_PROXIMITY	Hardware	Mide la proximidad de un objeto en centímetros, relativos a la pantalla del dispositivo. Típicamente se utiliza para determinar sin un móvil se encuentra cerca del oído de una persona.	Mide la posición del móvil durante una llamada.
TYPE_RELATIVE_HUMIDITY	Hardware	Mide la humedad relativa ambiental porcentual (%).	Monitorea humedad mínima, absoluta y relativa.
TYPE_ROTATION_VECTOR	Software o Hardware	Mide la orientación del dispositivo proveyendo los tres elementos del vector de rotación del dispositivo.	Detección de movimiento y rotación.
TYPE_TEMPERATURE	Hardware	Mide la temperatura del dispositivo en grados Celsius (°C). Las características varían de uno a otro dispositivo y es remplazado con el sensor TYPE_AMBIENT_TEMPERATURE en API nivel 14.	Monitorea temperaturas.

Por ejemplo, la mayoría de los dispositivos de teléfonos móviles y las tabletas tienen un acelerómetro y un magnetómetro, pero menos dispositivos tienen barómetros o termómetros. También, un dispositivo puede tener más de un sensor de un tipo dado. Por ejemplo, un dispositivo puede tener dos sensores de gravedad y cada uno tiene un intervalo diferente.

La jerarquía de la clase `Sensor`.

La clase `Sensor` representa un sensor. Utilizar `getSensorList(int)` para listar los sensores disponibles.

```
public final class Sensor extends Object
java.lang.Object -> android.hardware.Sensor

public abstract class SensorManager extends Object
java.lang.Object -> android.hardware.SensorManager
```

Uso de las clases de los sensores.

Se puede acceder a los sensores y adquirir sus datos crudos. El paquete `android.hardware` incluye las siguientes clases e interfaces:

`SensorManager`

Esta clase es para crear una instancia del servicio del sensor. Proporciona varios métodos para acceder y enumerar los sensores, registrar y anular el registro de los detectores de eventos del sensor, y la adquisición de información de orientación. También proporciona varias constantes de sensores que se utilizan para informar de la precisión del sensor, las tasas de adquisición de conjunto de datos, y calibrar sensores.

`Sensor`

Esta clase crea una instancia de un sensor específico. Proporciona varios métodos que permiten determinar las capacidades de un sensor.

`SensorEvent`

Esta clase para crea un objeto de evento sensor, que proporciona información acerca de un evento de sensor. Un objeto del evento sensor incluye la siguiente información: datos crudos del sensor, tipo de sensor que generó el evento, exactitud de los datos, y la marca de tiempo para el evento.

`SensorEventListener`

Esta interfaz crea dos métodos de devolución de llamada que reciben notificaciones de eventos (sensor) cuando se cambian los valores del sensor o cuando cambia la precisión del sensor.

En una aplicación típica utiliza estas API relacionadas con los sensores realizan dos tareas básicas:

- **Identificación de los sensores y capacidades de sensores**

La identificación de los sensores y capacidades de los sensores en tiempo de ejecución es útil si la aplicación tiene características que dependen de los tipos de sensores específicos o capacidades. Por ejemplo, es posible que desee identificar todos los sensores que están presentes en un dispositivo y desactivar funciones de las aplicaciones que se basan en sensores que no están presentes. Del mismo modo, es posible que desee identificar todos los sensores de un tipo dado para que pueda elegir la aplicación de sensor que tiene el rendimiento óptimo para la aplicación.

- **Supervisor de los eventos de sensores**

La supervisión de eventos de sensor es la forma de adquirir los datos crudos del sensor. Un evento de sensor se produce cada vez que un sensor detecta un cambio en los parámetros que está midiendo. Un evento de sensor proporciona cuatro piezas de información: nombre del sensor que activa el evento, marca de tiempo del evento, exactitud de la prueba, y datos crudos del sensor que desencadenó el evento.

Disponibilidad de los sensores.

La Tabla 2 resume la disponibilidad de cada sensor, plataforma por plataforma. Sólo cuatro plataformas se enumeran porque son las plataformas que involucran cambios de sensor. Los sensores indicados como obsoletos aún están disponibles en las plataformas subsiguientes (siempre que el sensor está presente en un dispositivo), en línea con la política de compatibilidad de Android.

Tabla 2. Disponibilidad de sensores, según la plataforma.

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`	Sí	Sí	Sí	Sí
`TYPE_AMBIENT_TEMPERATURE`	Sí	n/d	n/d	n/d
`TYPE_GRAVITY`	Sí	Sí	n/d	n/d
`TYPE_GYROSCOPE`	Sí	Sí	n/d[1]	n/d[1]
`TYPE_LIGHT`	Sí	Sí	Sí	Sí
`TYPE_LINEAR_ACCELERATION`	Sí	Sí	n/d	n/d
`TYPE_MAGNETIC_FIELD`	Sí	Sí	Sí	Sí

TYPE_ORIENTATION	Sí[2]	Sí[2]	Sí[2]	Sí
TYPE_PRESSURE	Sí	Sí	n/d[1]	n/d[1]
TYPE_PROXIMITY	Sí	Sí	Sí	Sí
TYPE_RELATIVE_HUMIDITY	Sí	n/d	n/d	n/d
TYPE_ROTATION_VECTOR	Sí	Sí	n/d	n/d
TYPE_TEMPERATURE	Sí[2]	Sí	Sí	Sí

Nota:
1 Se añadió en Android 1.5 (API Nivel 3), pero no estaba disponible para su uso hasta Android 2.3 (API Nivel 9).
2 Está disponible, pero ya no se utiliza.

Identificación de los sensores y sus capacidades.

Android proporciona varios métodos que hacen que sea fácil determinar, en tiempo de ejecución, cuáles sensores tiene un dispositivo. La API también proporciona métodos que permiten determinar las capacidades de cada sensor, su alcance máximo, resolución, y sus requerimientos de energía. Para identificar los sensores que se encuentran en un dispositivo primero se tiene que obtener una referencia al servicio del sensor. Para ello, se crea una instancia de la clase `SensorManager` llamando al método `getSystemService`() y pasando el argumento `SENSOR_SERVICE` . Por ejemplo:

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

A continuación, obtener una lista de todos los sensores en un dispositivo llamando al método `getSensorList`() y utilizando la constante `TYPE_ALL`. Por ejemplo:

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

Si se desea una lista de todos los sensores de un tipo determinado, se puede usar otra constante en lugar de `TYPE_ALL`, como `TYPE_GYROSCOPE`, `TYPE_LINEAR_ACCELERATION`, o `TYPE_GRAVITY`.

También se puede determinar si un tipo específico de sensor existe en un dispositivo mediante el método getDefaultSensor() y pasando la constante de un sensor de tipo específico. Si un dispositivo tiene más de un sensor de un tipo dado, uno de los sensores debe ser designado como el sensor de forma predeterminada. Si un sensor predeterminado no existe para un tipo de sensor, la llamada al método devuelve un valor nulo, lo que significa que el dispositivo no tiene ese tipo de sensor. Por ejemplo, el siguiente código comprueba si hay un magnetómetro en un dispositivo:

```
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.
  }
```

Nota: Android no requiere que los fabricantes de dispositivos construyan un tipo particular de sensor en sus dispositivos Android, por lo que los dispositivos pueden tener una amplia gama de configuraciones de sensor.

Además de enumerar los sensores que se encuentran en un dispositivo, se pueden utilizar los métodos públicos de la clase Sensor para determinar las capacidades y atributos de los sensores individuales. Esto es útil si desea que la aplicación se comporte de manera diferente en base al cual los sensores o las capacidades del sensor está disponible en un dispositivo. Por ejemplo, se pueden utilizar los métodos `getResolution`() y `getMaximumRange`() para obtener la resolución de un sensor y el rango máximo de medición. También se puede utilizar el método `getPower`() para obtener los requisitos de energía de un sensor.

Dos de los métodos públicos son particularmente útiles si se desea optimizar su aplicación para sensores de diferentes fabricantes o versiones distintas de un sensor. Por ejemplo, si la aplicación necesita controlar los gestos del usuario, como la inclinación y agitado, se puede crear un conjunto de reglas de filtrado de datos y optimizaciones para los nuevos dispositivos que tienen sensor de gravedad de un proveedor específico, y otro conjunto de reglas de filtrado de datos y optimizaciones para los dispositivos que no cuentan con un sensor de gravedad y sólo tienen un acelerómetro. El ejemplo siguiente muestra cómo se pueden utilizar los métodos `getVendor`() y `getVersion`() para hacer esto. En el código, se busca un sensor de gravedad que enumera Google Inc. como el proveedor y tiene un número de versión de 3. Si ese sensor en particular no está presente en el dispositivo, se intenta de utilizar el acelerómetro.

```
private SensorManager mSensorManager;
private Sensor mSensor;
      :
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = null;
if (mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) != null){
  List<Sensor> gravSensors = mSensorManager.getSensorList(Sensor.TYPE_GRAVITY);
  for(int i=0; i<gravSensors.size(); i++) {
    if ((gravSensors.get(i).getVendor().contains("Google Inc.")) &&
       (gravSensors.get(i).getVersion() == 3)){
      // Use the version 3 gravity sensor.
      mSensor = gravSensors.get(i);
    }
  }
}
if (mSensor == null){
  // Use the accelerometer.
  if (mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) != null){
    mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
  }
  else{
    // Sorry, there are no accelerometers on your device.
    // You can't play this game.
  }
}
```

Otro método útil es el método `getMinDelay`(), que devuelve el intervalo de tiempo mínimo (en microsegundos) que un sensor puede utilizar para detectar datos. Cualquier sensor que devuelve un valor distinto de cero para el método `getMinDelay`() es un sensor de streaming (flujo). Los sensores de Streaming detectan datos a intervalos regulares y se introdujeron en Android 2.3 (API Nivel 9). Si un sensor devuelve cero cuando se llama al método `getMinDelay`(), significa que el sensor no es un sensor de streaming, porque reporta datos sólo cuando hay un cambio en los parámetros que se encuentra detectando.

El método `getMinDelay`() es útil porque le permite determinar la velocidad máxima a la que un sensor puede adquirir datos. Si ciertas funciones de la aplicación requieren tasas altas de velocidades de adquisición de datos o un sensor de streaming, se puede utilizar este método para determinar si un sensor cumple con esos requisitos y luego, en consecuencia, activar o desactivar las características relevantes de la aplicación.

Precaución: La tasa máxima de velocidad de adquisición de datos de un sensor no es necesariamente la velocidad a la que el sensor suministra los datos para la aplicación. El sensor reporta datos a través de eventos del sensor, y varios factores influyen en la tasa de velocidad a la que la aplicación recibe eventos de sensor.

Monitoreando los eventos del sensor.

Para controlar los datos crudos del sensor se necesitan implantar dos métodos de la interfaz de `SensorEventListener`: `onAccuracyChanged`() y `onSensorChanged`(). El sistema Android llama a estos métodos cuando se produce lo siguiente:

- **Cambios de la precisión de un sensor.**

En este caso, el sistema invoca el método `onAccuracyChanged` (), que le proporciona una referencia al objeto de sensor que cambió y la nueva precisión del sensor. La precisión está representado por una de las cuatro constantes de estado: `SENSOR_STATUS_ACCURACY_LOW`, `SENSOR_STATUS_ACCURACY_MEDIUM`, `SENSOR_STATUS_ACCURACY_HIGH`, o `SENSOR_STATUS_UNRELIABLE`.

- **Un sensor informa de un nuevo valor.**

En este caso, el sistema invoca al método `onSensorChanged`(), que le proporciona un objeto `SensorEvent`. Un objeto `SensorEvent` contiene información sobre los nuevos datos de sensor, incluyendo: exactitud de los datos, el sensor que genera los datos, la marca de tiempo en el que se generaron los datos, y los nuevos datos que el sensor registra.

El código siguiente muestra cómo utilizar el método `onSensorChanged` () para supervisar los datos del sensor de luz. Este ejemplo muestra los datos crudos del sensor en una `TextView` que se define en el archivo main.xml como `sensor_data`.

```
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.
  }
  @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);
  }
}
```

En este ejemplo, el retardo predeterminado de datos (SENSOR_DELAY_NORMAL) se especifica cuando se invoca al método `registerListener` (). El retraso de datos (o velocidad de muestreo) controla el intervalo en el que se envían los eventos de sensor para su aplicación mediante el método `onSensorChanged` (). El retraso de datos predeterminado es adecuado para el seguimiento de los cambios típicos orientación de la pantalla y utiliza un retraso de 200.000 microsegundos. Se pueden especificar otros retrasos de datos, como `SENSOR_DELAY_GAME` (20.000 microsegundos de

retardo), SENSOR_DELAY_UI (60.000 microsegundos de retardo), o SENSOR_DELAY_FASTEST (0 microsegundos de retraso). A partir de Android 3.0 (API Nivel 11) también se puede especificar el retardo como un valor absoluto (en microsegundos).

El retraso que se especifica es sólo un retraso sugerido. El sistema Android y otras aplicaciones pueden alterar este retraso. Como práctica recomendada, se debe especificar el retraso más grande que se pueda porque el sistema utiliza normalmente un retraso menor que el que se especifique (es decir, se debe elegir la frecuencia de muestreo más lento que todavía satisfaga las necesidades de su aplicación). El uso de un retardo más largo impone una carga más baja en el procesador y por lo tanto consume menos energía.

No existe un método público para determinar la tasa de velocidad a la que el sensor envía los de eventos de sensores para la aplicación. Sin embargo, se pueden utilizar las marcas de tiempo que están asociados con cada evento de sensor para calcular la velocidad de muestreo lo largo de varios eventos. No se debería tener que cambiar la frecuencia de muestreo (retardo) una vez que se establece. Si por alguna razón se necesita cambiar el retraso, se tendrá que anular el registro y volver a registrar el escucha del sensor.

También es importante tener en cuenta que en este ejemplo se utilizan los métodos onResume() y onPause() para registrar y eliminar el detector de eventos del sensor. Para una mejor práctica siempre se deben desactivar los sensores que no se necesitan, sobre todo cuando está en pausa la actividad. De no hacerlo, se puede agotar la batería en tan sólo unas pocas horas debido a que algunos sensores tienen requisitos considerables de energía y pueden usar energía de la batería rápidamente. El sistema no se desactivará automáticamente cuando los sensores de la pantalla se apaguen.

Manejo de las diferentes configuraciones de los sensores.

Android no especifica una configuración estándar de sensores, lo que significa que los fabricantes de dispositivos pueden incorporar cualquier configuración de sensores que quieren en sus dispositivos Android. Como resultado, los dispositivos pueden incluir una variedad de sensores en una amplia gama de configuraciones. Por ejemplo, el Motorola Xoom tiene un sensor de presión, pero el Samsung Nexus S no lo hace. Del mismo modo, el Xoom y Nexus S tienen giroscopios, pero el HTC Nexus One no lo hace. Si la aplicación se basa en un tipo específico de sensor, se tiene que asegurar que el sensor está presente en un dispositivo para que la aplicación se pueda ejecutar correctamente. Se tienen dos opciones para asegurar que un determinado sensor está presente en un dispositivo:

- Detectar los sensores en tiempo de ejecución y activar o desactivar las características de aplicación, según proceda.
- Utilizar filtros de Google Play de los dispositivos de destino con configuraciones específicas de sensores.

Detección de sensores en tiempo de ejecución.

Si la aplicación utiliza un tipo específico de sensor, pero no se confía en él, se puede detectar el sensor en tiempo de ejecución y luego desactivar o activar las funciones de la aplicación, según proceda. Por ejemplo, una aplicación de navegación podría usar el sensor de temperatura, sensor de presión, sensor de GPS, y el sensor de campo geomagnético para mostrar la temperatura, la presión barométrica, la ubicación y rumbo de la brújula. Si un dispositivo no dispone de un sensor de presión, se puede detectar la ausencia del sensor de presión en tiempo de ejecución y luego desactivar la parte de la interfaz de usuario de la aplicación que muestra la presión. Por ejemplo, el siguiente código comprueba si hay un sensor de presión en un dispositivo:

```
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.
  }
```

Uso de filtros de Google Play para apuntar configuraciones específicas de sensores.

Si va a publicar la aplicación en Google Play se puede utilizar el elemento `<uses-feature>` en el archivo de manifiesto para filtrar la aplicación de los dispositivos que no cuentan con la configuración del sensor apropiado para la aplicación. El elemento `<uses-feature>` tiene varios descriptores de hardware que permiten filtrar las aplicaciones basadas en la presencia de sensores específicos. Los sensores que se pueden mostrar incluyen: acelerómetro, barómetro, brújula (campo geomagnético), giroscopio, la luz, y la proximidad. El siguiente es un ejemplo de entrada en el manifiesto que filtra las aplicaciones que no tienen un acelerómetro:

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

Si agrega este elemento y el descriptor al manifiesto de la aplicación, los usuarios podrán ver la aplicación en Google Play sólo si el dispositivo tiene un acelerómetro.

Se debe establecer el descriptor con `android:required="true"` sólo si la aplicación se basa por completo en un sensor específico. Si la aplicación utiliza un sensor para algunas funciones, pero todavía funciona sin el sensor, hay que enumerar el sensor en el elemento `<uses-feature>`, pero establecer el descriptor con `android:required="true"`. Esto ayuda a asegurar que los dispositivos pueden instalar la aplicación, incluso si no tienen ese sensor en particular. Esto también es una buena práctica de gestión de proyectos que ayuda a mantener un registro de las características que utiliza la aplicación. Tener en cuenta, si la aplicación utiliza un sensor en particular, pero que si aún se ejecuta sin el sensor, entonces se debería detectar el sensor en tiempo de ejecución y activar o desactivar las características de aplicación, según proceda.

Sistema de coordenadas del sensor.

En general, se utiliza un sistema estándar de coordenadas de 3 ejes para expresar los valores de datos. Para la mayoría de los sensores, el sistema de coordenadas se define en relación a la pantalla del dispositivo cuando el dispositivo se mantiene en su orientación predeterminada (ver figura 1). Cuando un dispositivo se mantiene en su orientación predeterminada, el eje X es horizontal y apunta a la derecha, el eje Y es vertical y apunta hacia arriba, y los puntos del eje Z hacia el exterior de la cara de la pantalla. En este sistema, las coordenadas detrás de la pantalla tienen valores Z negativos. Este sistema de coordenadas es utilizado por los siguientes sensores:

- Sensor de aceleración.
- Sensor de gravedad.
- Giroscopio.
- Sensor de aceleración lineal.
- Sensor de campo geomagnético.

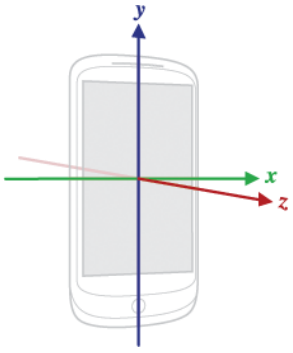

Figura 1. Sistema de coordenadas (en relación con un dispositivo) que utiliza la API del sensor.

El punto más importante es entender en este sistema de coordenadas que los ejes no se intercambian cuando cambia la orientación de la pantalla del dispositivo, es decir, el sistema de coordenadas del sensor no cambia a medida que el dispositivo se mueve. Este comportamiento es el mismo que el comportamiento del sistema de coordenadas OpenGL.

Otro punto a entender es que la aplicación no debe asumir que la orientación natural (predeterminada) de un dispositivo es vertical. La orientación natural para muchos dispositivos de tableta es horizontal. Y el de sistema de coordenadas del sensor siempre se basa en la orientación natural de un dispositivo.

Por último, si en la aplicación coinciden los datos del sensor con lo mostrado en pantalla, es necesario utilizar el método `getRotation`() para determinar la rotación de la pantalla, y luego usar el método `remapCoordinateSystem`() para asignar el sensor en coordenadas de pantalla. Es necesario hacer esto incluso si el manifiesto especifica sólo pantalla vertical.

Nota: Algunos sensores y métodos utilizan un sistema de coordenadas que está relacionado con el marco mundial de referencia (lo contrario al marco de referencia del dispositivo). Estos sensores y métodos devuelven datos que representan el movimiento del dispositivo o posición del dispositivo respecto a la Tierra. Consultar el método `getOrientation`(), el método `getRotationMatrix`(), sensor de orientación, y el sensor del vector de rotación.

Las mejores prácticas para el acceso y uso de los sensores.

A medida que se diseña la aplicación con sensores, asegurarse de seguir las indicaciones que se describen en esta sección. Esta guía recomienda las mejores prácticas para quien utilice las herramientas de sensores para accederlos y adquirir sus datos.

Eliminar escuchas de los sensores.

Asegurarse de eliminar el registro de escucha de un sensor cuando se haya terminado de usar el sensor o cuando la actividad del sensor se detiene. Si el escucha del sensor se ha registrado y su actividad está en pausa, el sensor continuará adquiriendo los datos y usando la batería, a menos que se elimine el registro del sensor. El código siguiente muestra cómo utilizar el método `onPause`() para eliminar el registro de un escucha:

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

No probar el código en el emulador

No se puede probar el código del sensor en el emulador ya que el emulador no puede emular sensores. Se debe probar el código del sensor en un dispositivo físico. Hay, sin embargo, simuladores de sensores que se pueden utilizar para simular la salida del sensor.

No bloquear el método `onSensorChanged` ()

Los datos de los sensores pueden cambiar a una alta velocidad, lo que significa que el sistema puede llamar al método `onSensorChanged(SensorEvent)` con bastante frecuencia. Se recomienda debe hacerlo lo menos posible dentro del método `onSensorChanged (SensorEvent)` para no bloquearlo. Si la aplicación requiere que se realice el filtrado de datos o la reducción de los datos del sensor, se debe realizar ese trabajo fuera del método `onSensorChanged (SensorEvent)`.

Evitar el uso de métodos, o tipos de sensores, obsoletos.

Varios métodos y constantes han quedado en desuso. En particular, el tipo de sensor `TYPE_ORIENTATION` ya no se utiliza. Para obtener los datos de la orientación en su lugar se debe utilizar el método `getOrientation`(). Del mismo modo, el tipo de sensor `TYPE_TEMPERATURE` ya no se utiliza. En cambio, se debe utilizar el tipo de sensor `TYPE_AMBIENT_TEMPERATURE` en los dispositivos que ejecutan Android 4.0.

Verificar los sensores antes de usarlos

Siempre comprobar que existe determinado sensor en un dispositivo antes de intentar adquirir datos de él. No se asuma que existe un sensor, simplemente porque es un sensor de uso frecuente. Los fabricantes de dispositivos no están obligados a proporcionar cualquier sensor particular en los dispositivos.

Elegir cuidadosamente los retardos del sensor.
Cuando se registre un sensor con el método `registerListener`(), asegurarse de elegir una tasa de velocidad de entrega que sea adecuada para la aplicación o uso concreto. Los sensores pueden proporcionar datos a velocidades muy altas. Permitiendo que el sistema envíe datos adicionales que no son necesarios lo que desperdicia recursos y energía del sistema.

DESARROLLO

EJEMPLO 1.
Este ejemplo muestra una lista de los sensores detectados en el dispositivo actual.
Paso 1. Crear un nuevo proyecto Sensores1 en Android Studio. En la carpeta java/com.example.mipaquete, abrir y modificar el archivo predeterminado MainActivity.java con el siguiente código:

```
import java.util.*;
import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.hardware.*;
import android.widget.TextView;
public class MainActivity extends Activity{
    TextView        jtv;
    Sensor          s;
    SensorManager   sm;
    List <Sensor>   l;
    String          c, v;
    int             n, t;
    float           p, r, d;
    public void onCreate(Bundle b) {
        super.onCreate(b);
        setContentView(R.layout.activity_main);
        jtv = (TextView) findViewById(R.id.xtv);
        sm  = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        l = sm.getSensorList(Sensor.TYPE_ALL);
        n   = l.size();
        jtv.append("\nSensores detectados: " + n + "\n");
        for(int i=0; i<n; i++){
            s = l.get(i);
            t = s.getType();
            c = s.getName();
            v = s.getVendor();
            p = s.getPower();
            r = s.getResolution();
            d = s.getMaximumRange();
            jtv.append("\nTipo de sensor: " + t + ", " + c);
            jtv.append("\nProveedor: " + v);
            jtv.append("\nPotencia (ma): " + p);
            jtv.append("\nMáxima resolución: " + r);
            jtv.append(", rango: " + d + "\n");
        }
    }
}
```

Paso 2. En la carpeta res/layout, abrir y modificar el archivo activity_main.xml con el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:id="@+id/xtv"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/lista" />
</LinearLayout>
```

Paso 3. En la carpeta res/values, abrir y modificar el archivo strings.xml con el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">MisSensores</string>
    <string name="action_settings">Settings</string>
    <string name="lista">Lista de Sensores:</string>
</resources>
```

Paso 4. En la carpeta res/menu, abrir y modificar el archivo menu_main.xml con el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MainActivity">
    <item
        android:id="@+id/action_settings"
        android:title="@string/action_settings"
        android:orderInCategory="100"
        app:showAsAction="never" />
</menu>
```

Paso 5. Por último, ejecutar la aplicación. Se muestra una lista de los sensores predeterminados del dispositivo actual. El número de sensores varía de acuerdo a la marca y modelo del fabricante del dispositivo. Si la lista es muy larga y no se visualizan todos los sensores, se debe modificar el archivo activity_main.xml agregando el par de etiquetas `<ScrollView>` antes y después de la etiqueta `<LinearLayout>` para efectuar el desplazamiento en la pantalla y visualizar el resto de sensores de la lista. En este ejemplo, se utilizó un emulador virtual modelo Nexus 5, API 23 x86, y solamente posee 5 sensores. En un dispositivo real, la apariencia de la lista debe ser similar a la siguiente:

EJERCICIO 1.

El siguiente ejercicio muestra los valores generados por el sensor de aceleración.

Paso 1. Crear un nuevo proyecto Sensores2 en Android Studio. En la carpeta java/com.example.mipaquete, abrir y modificar el archivo predeterminado MainActivity.java con el siguiente código:

```
import android.app.Activity;
import android.content.Context;
import android.hardware.*;
import android.os.*;
import android.widget.TextView;
public class MiAceleradorActivity extends Activity implements SensorEventListener{
    SensorManager   sm;
    Sensor          s;
    int             n;
    double          x, y, z, a, m, g;
    TextView        jtvX, jtvY, jtvZ, jtvA, jtvM, jtvG;
    public void onCreate(Bundle b){
        super.onCreate(b);
        setContentView(R.layout.content_mi_acelerador);
        sm = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        s = sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        sm.registerListener(this, s, SensorManager.SENSOR_DELAY_FASTEST);
        x=0; y=0; z=0; a=0; m=0; n=0;
        g = SensorManager.STANDARD_GRAVITY;
        jtvX = (TextView) findViewById(R.id.xtvX);
        jtvY = (TextView) findViewById(R.id.xtvY);
        jtvZ = (TextView) findViewById(R.id.xtvZ);
        jtvA = (TextView) findViewById(R.id.xtvA);
        jtvM = (TextView) findViewById(R.id.xtvM);
        jtvG = (TextView) findViewById(R.id.xtvG);
        new MiAsincronia().execute();
    }
    public void onSensorChanged(SensorEvent se){
        x = se.values[0];
```

```
        y = se.values[1];
        z = se.values[2];
        a = Math.sqrt(x*x + y*y + z*z);
        if(a>m)
            m = a;
    }
    public void onAccuracyChanged(Sensor s, int i){}
    class MiAsincronia extends AsyncTask<Void, Void, Void>{
        protected Void doInBackground(Void... x){
            while(true){
                try{
                    Thread.sleep(100); // 100 milisegundos
                } catch(InterruptedException e){
                    e.printStackTrace();
                }
                n++;
                publishProgress();
            }
        }
        protected void onProgressUpdate(Void... progress){
            jtvX.setText(" " + x + "\n");
            jtvY.setText(" " + y + "\n");
            jtvZ.setText(" " + z + "\n");
            jtvA.setText(" " + a + "\n");
            jtvM.setText(" " + m + "\n");
            jtvG.setText(" " + g + "\t\tActualización: " + n + "(ms)");
        }
    }
}
```

Paso 2. En la carpeta res/layout, abrir y modificar el archivo activity_main.xml con el siguiente código:

```
<?xml versión="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="*** Acelerómetro ***" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="En X: " />
    <TextView
        android:id="@+id/xtvX"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Aceleracion en X" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="En Y: " />
    <TextView
        android:id="@+id/xtvY"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Aceleracion en Y" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="En Z: " />
    <TextView
        android:id="@+id/xtvZ"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Aceleracion en Z" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Módulo (|A|): " />
    <TextView
        android:id="@+id/xtvA"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Aceleracion: " />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Aceleración máxima: " />
    <TextView
        android:id="@+id/xtvM"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Gravedad estándar: " />
    <TextView
        android:id="@+id/xtvG"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Paso 3. Por último, ejecutar la aplicación. Si se requieren, agregar los archivos necesarios para la adecuada ejecución de la aplicación. Capturar las imágenes que muestran los datos generados por el sensor de aceleración del dispositivo. Interpretar los resultados para justificarlos en un reporte.

EJERCICIO 2.

El siguiente ejercicio muestra los valores generados por el sensor magnético.

Paso 1. Crear un nuevo proyecto Sensores3 en Android Studio. En la carpeta java/com.example.mipaquete, abrir y modificar el archivo predeterminado MainActivity.java con el siguiente código:

```
import android.os.*;
import android.app.*;
import android.content.*;
import android.hardware.*;
import android.widget.*;
public class MainActivity extends Activity implements SensorEventListener{
    int             n = 0;
    boolean         c = true;
    double          x=0, y=0, z=0, a=0, bM=0;
    double          mfeM = SensorManager.MAGNETIC_FIELD_EARTH_MAX;
    double          mfem = SensorManager.MAGNETIC_FIELD_EARTH_MIN;
    SensorManager   sm;
    Sensor          s;
    TextView jtvx, jtvy, jtvz, jtva, jtvM, jtvG;
    @Override
    protected void onCreate(Bundle b) {
        super.onCreate(b);
        setContentView(R.layout.activity_main);
        jtvx = (TextView) findViewById(R.id.xtvX);
        jtvy = (TextView) findViewById(R.id.xtvY);
        jtvz = (TextView) findViewById(R.id.xtvZ);
        jtva = (TextView) findViewById(R.id.xtvB);
        jtvM = (TextView) findViewById(R.id.xtvM);
        jtvG = (TextView) findViewById(R.id.xtvT);
        sm   = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        s    = sm.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
        sm.registerListener(this, s, SensorManager.SENSOR_DELAY_NORMAL);
        new Asincronia().execute();
    }
```

```
    public void onResume(){
        super.onResume();
        c = true;
        new Asincronia().execute();
    }
    public void onPause(){
        super.onPause();
        c = false;
    }
    public void onAccuracyChanged(Sensor s, int i){}
    public void onSensorChanged(SensorEvent e){
        x = e.values[0];
        y = e.values[1];
        z = e.values[2];
        a = Math.sqrt(x*x + y*y + z*z);
        if(a > bM)
            bM = a;
    }
    class Asincronia extends AsyncTask<Void, Void, Void>{
        protected Void doInBackground(Void... v){
            while (c){
                try{
                    Thread.sleep(200);
                }catch(InterruptedException e){ e.printStackTrace(); }
                n++;
                publishProgress();
            }
            return null;
        }
        protected void onProgressUpdate(Void... v){
            jtvx.setText(x + "");
            jtvy.setText(y + "");
            jtvz.setText(z + "");
            jtva.setText(a + "");
            jtvM.setText(bM + "");
            jtvG.setText(mfem + " - " + mfeM  + "");
            jtvG.append(n  + "");
        }
        protected void onPostExecute(Void result){
            super.onPostExecute(result);
            Context c = getApplicationContext();
            Toast.makeText(c, "Fin de Campo Magnetico.", Toast.LENGTH_LONG).show();
        }
    }
}
```

Paso 2. En la carpeta res/layout, abrir y modificar el archivo activity_main.xml con el siguiente código

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:text="*** Sensor Magnético ***\n"/>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="B en X:" />
    <TextView
        android:id="@+id/xtvX"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Campo X"/>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="\nB en Y:"/>
    <TextView
        android:id="@+id/xtvY"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Campo Y"/>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="\nB en Z:" />
    <TextView
        android:id="@+id/xtvZ"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Campo Z"/>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="\nB módulo:" />
    <TextView
        android:id="@+id/xtvB"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Campo total" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="\nCampo máximo:" />
    <TextView
        android:id="@+id/xtvM"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Campo terrestre" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="\nCampo de la Tierra mínimo y máximo:" />
    <TextView
        android:id="@+id/xtvT"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Campo tierra"/>
</LinearLayout>
```

Paso 3. Por último, ejecutar la aplicación. Agregar los archivos necesarios para la adecuada ejecución de la aplicación. Capturar las imágenes que muestran los datos generados por el sensor magnético del dispositivo. Interpretar los resultados para justificarlos en un reporte.