dd

Este documento contiene distintos conceptos teóricos sobre el funcionamiento del Sistema Operativo Android. Asimismo se describen los mecanismos y componentes que ofrece su Framework para poder desarrollar aplicaciones que puedan ejecutarse en esta plataforma.

Apunte Teórico sobre el Sistema Operativo Android

Sistemas Operativos Avanzados

Ing. Esteban A. Carnuccio
Ing. Mariano Volker
Ing. Raúl Villlca
Ing. Matías Adagio

Apunte Teórico sobre el Sistema Operativo Android

Contenido

1. Introducción

En este documento se detallarán conceptos teóricos sobre el funcionamiento del Sistema Operativo Android y la creación de aplicaciones que funcionen en la plataforma. Existen muchas herramientas y lenguajes de programación que permiten desarrollar programas para Android. No obstante, se utilizará como herramienta de programación al IDE Android Studio y los ejemplos mostrados serán codificados en Java. Esto se debe a que son herramientas base, que Google sugiere utilizar con el fin de desarrollar programas para este Sistema Operativo.

2. Gradle y complemento de Android

Gradle es una herramienta flexible que es utilizada para gestionar dependencias, integrar con repositorios o módulos externos para el proyecto, y establecer la descripción del módulo empaquetado. A su vez, Android posee dos archivos de configuración para Gradle. El primero está a nivel de raíz del proyecto y, el otro, a nivel de aplicación. En la figura siguiente se muestra la estructuración de Archivos que componen un Proyecto de Android Studio de acuerdo a la configuración de Gradle.

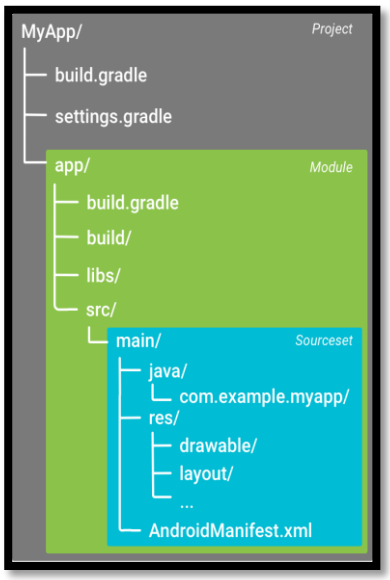


Fig. 1 Estructura de Archivos de un proyecto de Android Studio

A continuación se describen con mayor detalle los archivos de configuración que usa Gradle en un proyecto Android:

- **build.gradle (MyApp)**: Contiene el complemento de Android para Gradle llamado com.android.tools.build:Gradle, siendo una configuración global de todos los módulos de la aplicación. Cuando Android se actualiza, generalmente va a pedir que se actualice el complemento, pudiendo realizarlo de manera manual o automática. Posteriormente se permitirá actualizar los módulos sincronizando la aplicación.
- **build.gradle (app):** Abarca las configuraciones compiladas del programa tales como versión y plataforma de compilación, id de la aplicación, entre otros. A su vez, describe qué repositorios y/o dependencias de ese módulo utilizará durante la ejecución del proyecto.
- **settings.gradle:** Este archivo le indica a Gradle qué módulos debe incluir en el proyecto de la aplicación.

3. ADMINISTRADOR DE SDK

Permite instalar o actualizar versiones específicas de Android (API), siendo herramientas de compilación (sdk platform-tools, sdk platform) necesarias para comenzar el desarrollo. Se recomienda que sean por lo menos de dos versiones específicas de Android debido a que el proyecto exige por un lado, una versión mínima requerida y por el otro, una estable para el desarrollo que se quiera realizar. En la figura siguiente se muestra cómo se puede acceder al Administrador de SDK, desde la pantalla inicial de Android Studio.

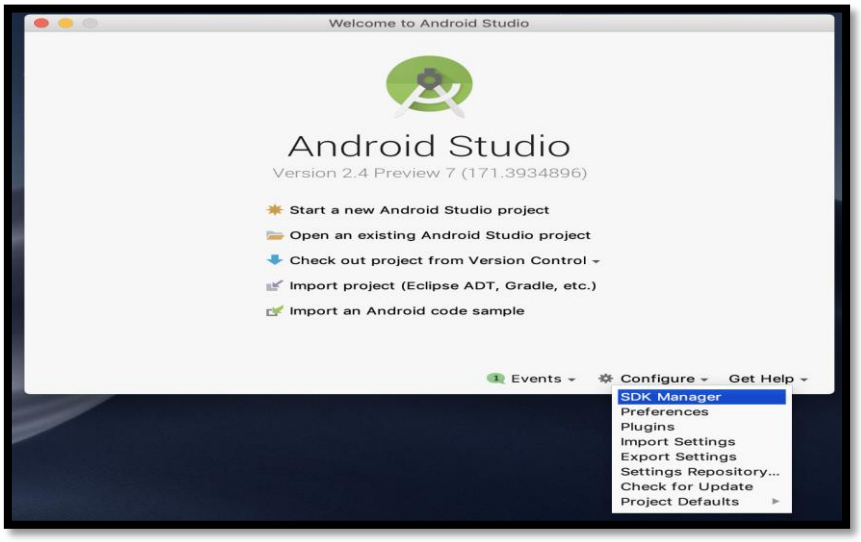


Fig. 2Acceso al SDK Manager por medio de la interfaz de Inicio de Android Studio

Una vez elegida la opción de menú, se abrirá una interfaz del SDK Manager, visible en la Fig. 3. En dicha pantalla se podrán elegir e instalar bibliotecas específicas de Android (API) en su PC, que les permitirán desarrollar aplicaciones ejecutables en las versiones del S.O seleccionadas.

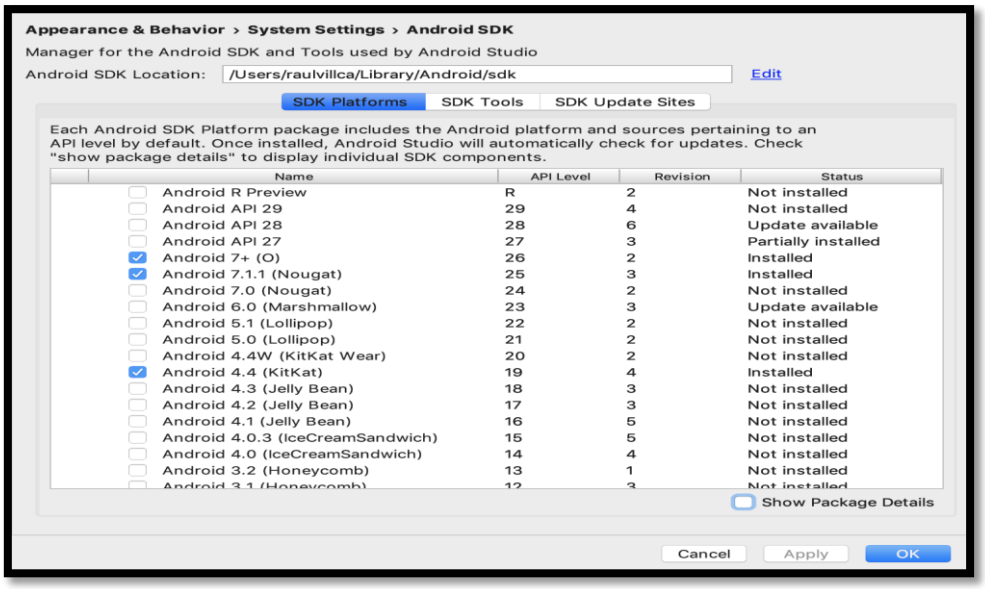


Fig. 3 Interfaz del Administrador de SDK

4. ACTIVITY

Es una pantalla de aplicación de Android. De esta manera, cada ventana que conforma una aplicación es una Activity. En este sentido, si un programa de Android tiene tres pantallas, estará formada por tres Activities. En la siguiente figura se muestra un ejemplo de una aplicación que posee varias Acvtities.

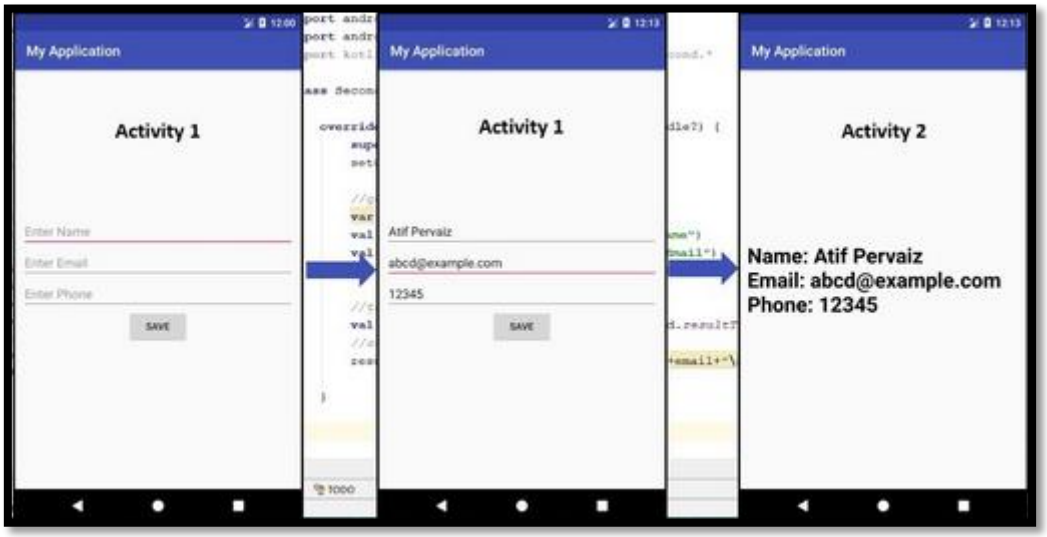


Fig. 4 Ejemplo de Aplicación que contiene 2 Activities

Al momento de programar una pantalla de una aplicación desde Android Studio, se puede observar que una Activity está divida en dos partes:

- **Front-End (Parte Gráfica):** Archivo XML
- **Back-End (Parte lógica) :** Archivo .Java

a. Front-End

Se refiere a la parte gráfica que el usuario va a visualizar cuando utilice la aplicación. Esta puede estar conformada por botones, caja de textos, checkbox, barra deslizantes, etc. Android Studio le

permite al programador crear la parte visual de las Activitys fácilmente arrastrando y soltando componentes gráficos con el mouse. En ese sentido, a medida que el programador vaya arrastrando dichos componentes a la interfaz gráfica de una Activity, Android Studio automáticamente va a ir creando un archivo XML asociada al mismo. En este archivo va a estar el código fuente de los componentes que vayamos creando con el mouse. En la Fig. 5 se puede visualizar la interfaz gráfica del archivo XML y sus componentes:

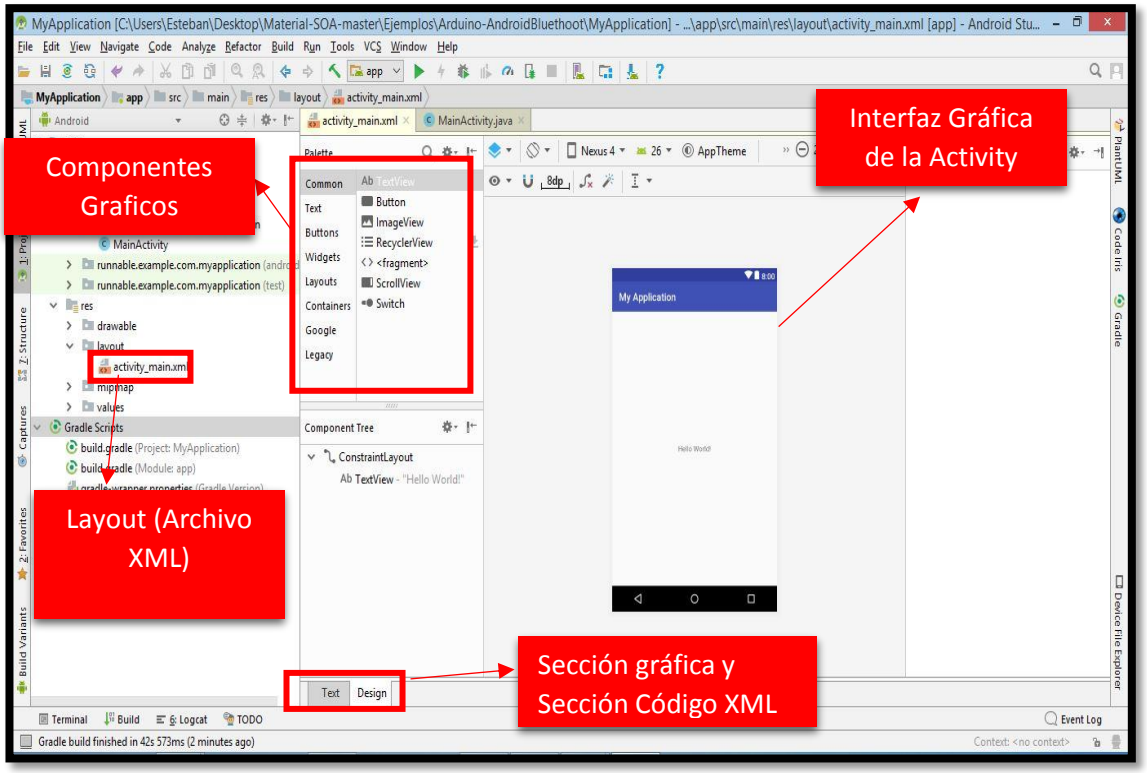


Fig. 5 Interfaz gráfica del archivo XML de una Activity (Front-End)

De esta forma, el programador tiene dos maneras de crear la parte gráfica de una Activity: la primera consiste en ir arrastrando y soltando los componentes con el mouse; la segunda, puede ser directamente desde código en un archivo XML.

b. Back-End

En el apartado anterior se mencionó que el Front-End permite armar la parte gráfica de las Activities. Pero para que pueda funcionar, debe haber un código fuente que indique qué acciones deben realizar cada uno de sus componentes gráficos cuando el usuario interactúe con ellos. Por ese motivo cuando un programador crea una Activity, Android Studio le asocia automáticamente a su parte gráfica, un archivo .Java que contiene toda su lógica. Este archivo pertenece al Back-End de las Activities. En este fichero se pueden poner las acciones que deberá realizar una Activity cuando el usuario interactúe con ella. En la figura siguiente se muestra el archivo Java asociado a la parte gráfica expuesta anteriormente.

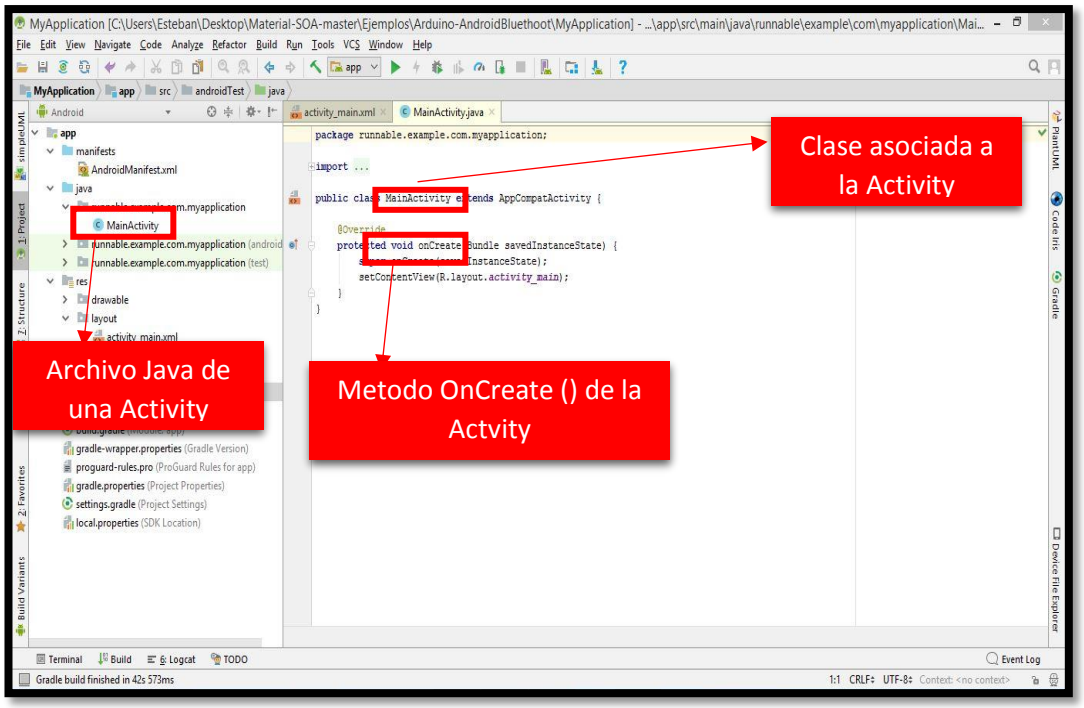


Fig. 6 Archivo Java asociado a una Activity

En la Fig. 6 se observa el archivo Java asociado a una Activity. Dentro de su código se halla la clase perteneciente, que lleva el mismo nombre. En ella el programador podrá escribir el código con las acciones que deberá realizar la Activity cuando interactúe con el usuario. Un punto importante de nombrar, es que en esta clase se encuentra escrito por defecto el OnCreate (), que es la primera función a ejecutarse cuando el usuario utilice por primera vez la Activity. Este método será descripto con mayor detalle en el apartado siguiente.

c. Ciclo de Vida de una Activity

Como se dijo anteriormente una aplicación puede estar compuesta por varias Activities (pantallas). Cuando se ejecuta una Activity en ella, la misma pasa por diversos estados a lo largo de su ciclo de vida. En cada uno de ellos, el Sistema Operativo de Android realiza distintas acciones. En toda aplicación existe una Activity Principal, que es la primera en ejecutarse cuando un usuario comienza el programa por vez inicial. Por ese motivo a continuación se describirán los distintos estados por lo que transcurre una Activity durante su ejecución. En la siguiente figura [1], se muestra el diagrama de estados perteneciente al ciclo de vida de una Activity.

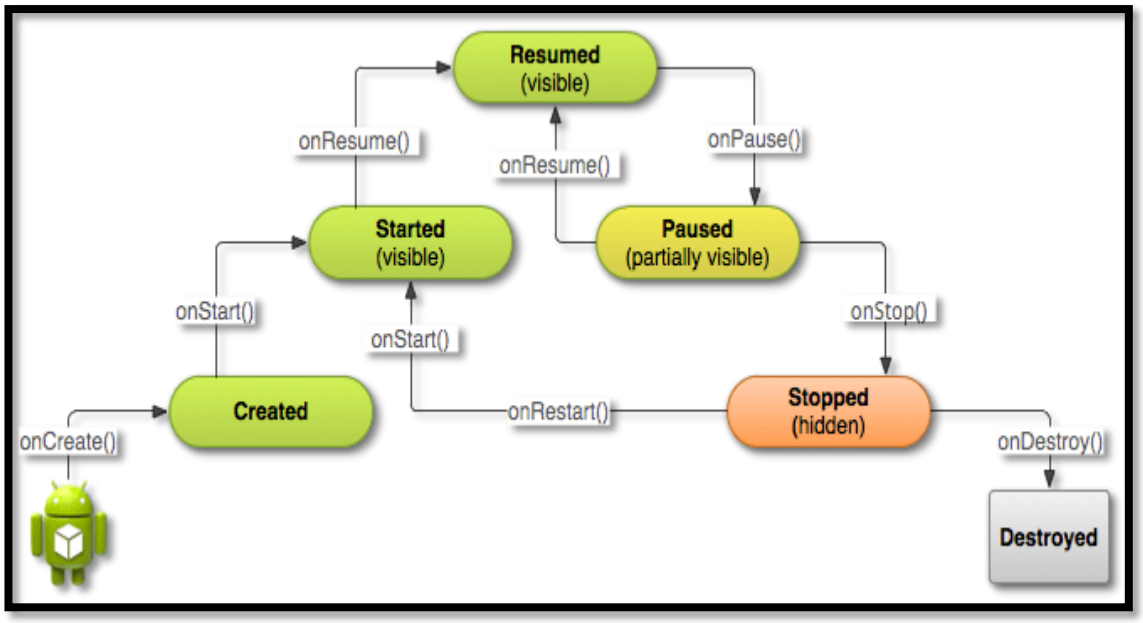


Fig. 7 Ciclo de Vida de una Activity

- **OnCreated:** Cuando el usuario utiliza por primera vez una Activity, se ejecuta automáticamente en su código Java el método **OnCreate()**. En consecuencia cuando se invoca esta función, la Activity pasa al estado *Created*. Es importante mencionar que en este estado la pantalla no estará visible para el usuario, por lo que no podrá interactuar con ella. Consecutivamente el Sistema Operativo crea y carga en memoria el objeto correspondiente a la Activity, con sus datos iniciales. Dentro de este método el programador puede colocar un código de inicialización, recibir parámetros que le envían otras Activities y recibir datos de sus estados previos. Se recuerda y advierte que el código de este método se ejecuta por única vez al iniciar la Activity, salvo que sea destruida. En dicho caso esta función se volverá a efectuar.

- **Started:** Al finalizar la ejecución del onCreate(), automáticamente se invoca al método **onStart()**. Cuando se realiza dicha función, la Activity pasa a estar en el estado Started. En él, la pantalla gráfica de la misma pasa a ser visible para el usuario, quien no podrá interactuar con ella. Por ese motivo cualquier acción que quiera realizar en ella no tendrá efecto.

- **Resumed:** Concluido **onStarted()**, automáticamente se invoca al método onResume(). La Activity pasará a estar en el estado Resumed. Lugar del que será visible (en primer plano), y estará habilitada para que el usuario pueda interactuar con ella. Se quedará esperando acciones del usuario, y no saldrá de esa circunstancia a menos que el usuario lo indique, o que suceda algún evento que lo modifique.

- **Paused:** El método que se invoca para realizar este cambio de estado es **onPaused()**. Una Activity pasa a él cuando pierde el foco, dejando de estar en primer plano. No obstante sigue siendo visible en forma parcial. Un ejemplo de ello sería al iniciar una Activity que no ocupa la pantalla completa, superponiéndose parcialmente a otra que se encontraría en primer plano, de forma tal que ésta no se pudiera ver en su totalidad. Cuando una Activity se encuentra pausada, no podrá interactuar con el usuario, debido a que no se encuentra en primer plano. Así, ella se mantendrá en este estado esperando que suceda algún evento que lo saque del mismo.

- **Stoped:** Una Activity pasará a Stoped cuando deje de estar completamente en primer plano, de forma tal que el usuario no pueda visualizarla en su totalidad. En consecuencia se ejecutará el método **onStop().** La Activity se mantendrá en este estado esperando que suceda algún evento que lo saque del mismo.

- **Destroyed:** El método onDestroy() se ejecuta antes de que la Activity finalice o sea destruida. Cuando finalice la ejecución de esta función, pasará al estado Destroyed. Con lo cual el S.O libera los recursos que fueron asignados a ella, eliminando su objeto de la memoria.

d. Ejemplos de distintas situaciones de los estados del ciclo de vida de una Activity

La siguiente tabla muestra ejemplos de una serie de situaciones que puede llegar atravesar una Activity a lo largo de su ciclo de vida, mostrando cuáles serían su estado final e intermedios. Se recomienda ver los ejemplos de la tabla observando transiciones de la Fig. 7.

Situación	Transición de llamadas estados	Comentarios
Se abre la Activity1 por primera vez	1°. onCreate() 2°. Onstrart() 3°. OnResume()	La Activity se queda en el estado Resumed esperando interacciones del usuario
Estando visible la Activity1 (estado Resumed), se cierra presionando el botón de volver en la barra de navegación	1°. OnPause() 2°. OnStop() 3°. OnDestroy()	La Activity se destruye al cerrarla
Estando visible la Activity1 (estado Resumed), se presiona el botón de inicio o el botón de aplicaciones recientes en la barra de navegación	1°. OnPause() 2°. OnStop()	La Activity estando visible se oculta completamente, por lo que se queda en el estado Stoped a la espera de algún evento.
Estando visible la Activity1 (estado Resumed), se abre una Activity2 que oculta parcialmente a la primera actividad	1°. OnPause()	La Activity estando visible se oculta parcialmente, por lo que se queda en el estado Stoped a la espera de algún evento.
La Activity1 se encuentra en el estado Paused, debido a que está parcialmente oculta por la Activitiy2. Luego el usuario selecciona la Activity1, dándole foco, ubicándola en primer plano.	1°. OnResume()	Al estar en el estado Paused y pasar nuevamente a tener foco, la Activity pasa nuevamente al estado Resumed
La Activity1 se encuentra en el estado Stoped, debido a que está oculta totalmente. Luego el usuario selecciona la Activity1, dándole foco, ubicándola en primer plano.	1°. OnRestart() 2°. OnResume()	Al estar en el estado Stoped y pasar nuevamente a tener foco, la Activity pasa nuevamente al estado Resumed
El usuario abre una Activity1 que anteriormente había sido destruida.	1°. onCreate() 2°. Onstrart() 3°. OnResume()	Como la Activity había sido destruida, al abrirla

		nuevamente se vuelve a crear otra vez.
Estando visible la Activity1 (estado Resumed), el usuario gira 90° el Smartphone.	1°. OnPause() 2°. OnStop() 3°. OnDestroy() 4°. OnCreate() 5°. OnStart() 6°. OnResume()	Cuando una Activity se encuentra visible en primer plano y el usuario gira la pantalla 90°, el sistema la destruirá y la volverá a crear. Debido a que le resulta mucho más rápido al S.O crear nuevamente todos los componentes de la Activitys en sus nuevas posiciones, que reordenarlos gráficamente.

Tabla 1 Ejemplos del ciclo de vida de una Activity

En el siguiente repositorio se encuentra el código fuente de un proyecto Android que permitirá observar los distintos estados que adquiere una Activity durante su Ciclo de Vida, a través de mensajes Logs.

- https://gitlab.com/so-unlam/Material-SOA/-/tree/master/Ejemplos/Android/CV-Activity/CV-Activity-Bundle

5. INTENTS

Es un objeto que internamente posee un mensaje. Se lo puede ejecutar para solicitar que otro componente de la aplicación, sea nuesta o externa, realice una determinada acción. Un Intent nos permite comunicar distintos componentes. A su vez, existen dos situaciones comunmente utilizadas en él:

a. Para iniciar Activities, Services o Broadcast
b. Para solicitar a una aplicación externa la realización de una tarea

En los apartados siguientes se describirán cada una de estas situaciones. No se mostrará en detalle el uso de los Intents, sino que se explicarán sus conceptos básicos. Para mayor detalle sobre ellos, ver la documentación oficial de Android [2]

a. Tipos de Intents

De acuerdo a su forma de utilización, se clasifican en:

- **Intent Explícitas:**
  Se usa para iniciar una acción a través de un determinado componente. En estos casos, cuando se crea el Intent, se debe especificar el nombre del componente. Este tipo de mensajes es utilizado principalmente para comenzar Activities, Services o Broadcast, en donde se conoce la denominación con que se lo identifica. Un ejemplo de Intent Explícita, es cuando se quiere que desde una Activity se abra a otra de nuestra aplicación. Por lo que se le debe especificar a la Intent el nombre de la Activitiy que deseamos abrir.

- **Intent Implícitas:**
  Cuando se utiliza no se debe especificar el nombre del componente que se desea iniciar. Al crearse el Intent, lo único que se debe indicar es la acción que se quiere realizar. Por lo que

una vez ejecutado, el Sistema Operativo buscará todas las aplicaciones externas que se encuentren instaladas en el Smartphone y que puedan realizar dicha operación. En el caso de que el S.O halle alguna aplicación externa capaz de cumplir con esa tarea, automáticamente la ejecutará. En cambio, si encontrara que más de una aplicación externa puede realizar la acción solicitada, el sistema le mostrará al usuario una pantalla de selección de app, para que pueda elegir la que desea utilizar. La interfaz mencionada se muestra en la figura siguiente [2].

Fig. 8 Selector de Aplicaciones para Intent Implícitas

Un ejemplo de Intent Implícita es cuando se crea una para que desde el programa se pueda una página web. En este caso se debe especificar con qué aplicación se desea verla, sino que se solicita al S.O que busque con qué programa pueda abrirla. Entonces cuando se ejecuta el Intent, el S.O se fijará qué aplicaciones tiene registradas para realizar dicha acción. Posteriormente cuando encuentre alguna app capaz de llevarla a cabo, el S.O lo abrirá automáticamente y mostrará la sitio web.

b. Intents para iniciar Activities, Services y Broadcast

- **Iniciar Activities**

Para iniciar una Activity desde otra Activity y pasar datos entre ellas. Se debe realizar los siguientes pasos:

En el código de la Activity llamadora (Origen)

1°. Se debe crear un Intent desde la Activity llamadora (Origen) indicándole de forma explícita, cuál es la Activity que se quiere invocar:

```
Intent intentA = new Intent (Activity Origen, Activity Destino);
```

2°. Si se desean pasar datos a la segunda Activity se deben guardar en el Intent en una estructura “Etiqueta”-> valor, los parámetros que se quieren enviar:

```
intentA.putExtra (“Etiqueta1”,valor);
intentA.putExtra (“Etiqueta2”,valor);
```

```
intentA.putExtra (“Etiqueta3”,valor);
```

Este punto se debe repetir dependiendo de la cantidad de parámetros que deseamos enviar.

3°. Se ejecuta el Intent, iniciando para ello la Activity con la siguiente instrucción:

```
startActivty(intentA);
```

En el caso de que se desee que la Activity Destino retorne un resultado a la Activity Origen, se debe usar la siguiente instrucción en lugar de la anterior:

```
startActivityForResult(intentA,1);
```

En el código de la Activity llamada (Destino)

Cuando en tiempo de ejecución se invoque a la instrucción startActivity, automáticamente se abrirá la Activity llamadora y se ejecutará su código. Por lo que si se desea recuperar los parámetros que fueron enviados en forma de etiqueta, estos se deben recuperar de la siguiente forma:

4°. Dentro de la clase de la Activity Destino, se pueden recuperar los parámetros enviados a través de un objeto Bundle:

```
Intent intentB= getIntent(); //se recupera el Intent que envió la Activity de Origen
Bundle extras = intentB.getExtras(); //se guardan los parámetros en un objeto Bundle
String Texto1 = extras.getString (“Etiqueta1”); //se recupera el valor de la etiqueta1 y se
                                                 //lo guarda en una variable
String Texto2 = extras.getString (“Etiqueta2”); //se recupera el valor de la etiqueta2 y se
                                                 //lo guarda en una variable
```

- **Iniciar Services**

Para iniciar un services desde el código de una Activity en forma explícita usando un Intent, se deben seguir los siguientes pasos

1°. Se crea un Intent desde la Activity, indicándole de forma explícita cuál es la clase del Service que se quiere invocar.

```
Intent intentS = new Intent (Activity Origen, clase del Service);
```

2°. Si desde la Activity se desean pasar parámetros al Services cuando se inicia, se deberá usar el método **putExtra** del Intent como se explicó en el apartado para iniciar Activities.

3°. Posteriormente, para poder iniciar el servicio se debe ejecutar el Intent usando la instrucción **startService**()

```
startService (intentS);
```

- **Iniciar un Broadcast**

  Para iniciarlo, se deben realizar los siguientes pasos:

  1°. Se crea un Intent desde la Activity, indicándole de forma explícita cuál es el Broadcast que se quiere invocar:

  IntentB = new Intent (nombre de la acción del Broadcast);

  2°. Si desde la Activity se desean pasar parámetros al Broadcast cuando este inicia, se deberá usar el método **putExtra** del Intent como se explicó en el apartado para iniciar Activities.

  3°. Posteriormente para poder iniciar el Broadcast, se debe ejecutar el Intent usando la instrucción **sendBroadcast()**:

  sendBroadcast (IntentB);

- **Ejemplo de Intent para iniciar y pasar datos a una Activity**

  A continuación, se muestra un ejemplo de Intent Explícita donde se inicia una Activity desde otra y se le envían parámetros:

```
//se genera un Intent para poder lanzar la activity principal
intent=new Intent(MainActivity.this,DialogActivity.class);

//Se le agrega al intent los parametros que se le quieren pasar a la activyt principal
//cuando se lanzado
intent.putExtra("textoOrigen",txtOrigen.getText().toString());

//se inicia la activity principal
startActivity(intent);
```

Fig. 9 Código de la Activity Origen

```
//se crea un objeto Bundle para poder recibir los parametros
//enviados por la activity Inicio
//al momeento de ejecutar stratActivity
Intent intent=getIntent();
Bundle extras=intent.getExtras();
String texto= extras.getString("textoOrigen");
txtDestino.setText(texto);
```

Fig. 10 Código de la Activity Destino

En el siguiente repositorio se encuentra el código fuente del proyecto Android de este ejemplo:

- https://gitlab.com/so-unlam/Material-SOA/-/tree/master/Ejemplos/Android/CV-Activity/CV-Activity-Bundle

c. Pasos a seguir para solicitar a una aplicación externa la realización de una tarea mediante un Intent Implícito.

1°. Crear un Intent indicando la acción y la **Uri** que se desea realizar por medio de una aplicación externa. El parámetro Uri es utilizado para determinadas acciones y se refiere a los datos con que va a trabajar específicamente la misma:

```
Intent IntentE = new Intent (Accion,URI);
```

2°. Opcionalmente puede incluirse al Intent parámetros adicionales que se desean enviar a la aplicación a ejecutar, a través del método **puteEtxra()**

3°. Finalmente se debe iniciar el Intent a través de la instrucción startActivity():

```
startActivty(intentE);
```

- **Ejemplos de Intents para solicitar una acción a una aplicación externa**

A continuación se muestran dos ejemplos de Intent Implícitos que invocan a una aplicación externa:

```
public void llamadaTelefono(View view) {
    Intent intent = new Intent(Intent.ACTION_DIAL,
                        Uri.parse("tel:962849347"));
    startActivity(intent);
}
```

Fig. 11 Código de un Intent que ejecuta una aplicación externa que llama a un número de teléfono determinado

```
public void googleMaps(View view) {
    Intent intent = new Intent(Intent.ACTION_VIEW,
                        Uri.parse("geo:41.656313,-0.877351"));
    startActivity(intent);
}
```

Fig. 12 Código de un Intent que muestra en un mapa de Google Maps la ubicación de las coordenadas pasadas en el parámetro URI

Se recomienda leer el tutorial de la siguiente página, si se desea conocer más sobre Intent Implícitas, para poder ejecutar aplicaciones externas.

- http://www.androidcurso.com/index.php/130

d. Intent Filters (Filtro de Intents)

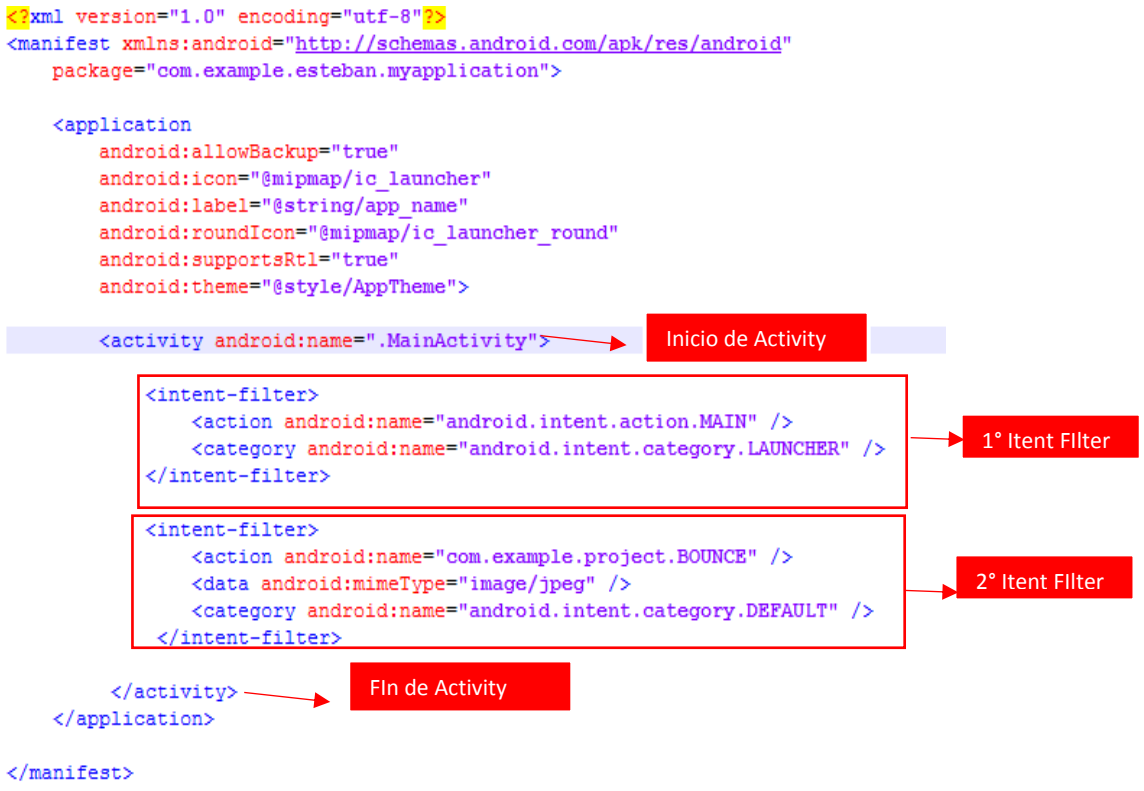
En un apartado anterior, mencionamos que usando un tipo de Intent Implícita se podría solicitar a una aplicación externa realizar una acción. Por ejemplo, si desde la aplicación se desea sacar una foto con la cámara del Smartphone, se puede ejecutar un Intent implícita que le solicite al S.O

obtenerla por medio de una de las aplicaciones que estén instaladas en el teléfono y posteriormente nos devuelva la imagen capturada. Pero, ¿qué pasa si se quiere hacer el proceso inverso? ¿Qué aplicaciones externas puedan solicitarle a nuestro programa para que realice determinadas acciones? Para realizarlo el Framework de Android tiene una herramienta que se llama Intent Filter. Este instrumento permite al programador indicarle al S.O, qué tipos de Intent Implícitas puede administrar la aplicación para realizar una determinada acción que solicite una aplicación externa. Gracias a esto, se puede por ejemplo, desarrollar un programa de mensajería similar a WhatsApp que se registrará en el S.O a través de un Filtro de Intent. Concluyendo que cuando una aplicación externa solicite enviar un mensaje de texto usando un Intent Implícito determinado, el S.O le ofrecerá al usuario poder utilizar nuestra aplicación mediante el selector de app de la Fig. 8.

Si se quiere registrar un Intent Filter en el S.O, se debe realizar en el archivo AndroidManifest.XML de nuestra aplicación. Para eso se seguirá un formato específico de inscripción, en donde se detallarán ciertos campos del Intent. Si se desea conocer de manera minuciosa dicho formato, se recomienda leer los tutoriales:

- https://desarrollador-android.com/desarrollo/formacion/empezar-formacion/interactuar-con-otras-apps/permitir-otras-apps-inicien-actividad/
- https://developer.android.com/training/basics/intents/filters?hl=es

En la figura siguiente, se muestra un ejemplo de dos Intent Filters que son registrados en el archivo Manifest de una aplicación.




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

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

        <activity android:name=".MainActivity">

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

            <intent-filter>
                <action android:name="com.example.project.BOUNCE" />
                <data android:mimeType="image/jpeg" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>

        </activity>
    </application>

</manifest>
```

Fig. 13Registración de Intent Filter en el Archivo AndoridManifest.XML

En la figura anterior, se muestran dos IntentFilters declarados dentro de la Activity llamada MainActivity. El primer Intent indica que esta Activity será el punto de entrada principal de la aplicación. En otras palabras, será la pantalla inicial del programa, la cual se abrirá cuando el

usuario lance inicialmente la app con el ícono de selector. Por otra parte, el segundo Intent Filter está indicando que esta Activity puede manejar imágenes jpeg, las cuales pueden ser solicitadas por una aplicación externa a través de un Intent Implícito.

6. Sensores en Android

a. Tipo de Sensores

Los Smartphone poseen incorporados físicamente varios sensores que varían en cantidad, según su marca y modelo. Asimismo, estos deben poder ser controlados por el S.O que se instale en el dispositivo, debiendo tener instaladas las bibliotecas drivers que permitirán utilizar dichos sensores. En ese sentido, la versión de sus drivers que tenga instalados en el teléfono, dependerá de la versión del S.O Android que se esté utilizando y determinará la cantidad y tipo de sensores que pueda manejar el S.O. No sirve de nada tener un Smartphone que venga de fábrica con demasiados sensores, si no tiene instalado el S.O con los drivers adecuado para poder utilizarlos.

Desde la tienda oficial de Android de Google Playstore, se pueden descargar diferentes aplicaciones que le permitirán al lector conocer qué sensores puede utilizar en su Smartphone y cuáles no. En este documento se recomienda la aplicación **Z-Device Test,** debido a que es bastante útil para cumplir dicha tarea.

Hay sensores que vienen físicamente con el hardware correspondiente incorporados en el dispositivo móvil. Por ejemplo, el acelerómetro y el sensor de luz, entre otros. A su vez, existen algunos que son virtuales debido a que son generados en forma lógica por el S.O que, por lo general, son una derivación de uno o más sensores físicos. Este es el caso del que es de aclaración lineal y el de gravedad.

Para continuar, se hará una lista de los sensores que traen los Smartphone que son más utilizados, y se mostrará para qué se emplean cada uno de ellos.

Sensor	Descripción	Ejemplo de Uso
Acelerómetro	Mide la aceleración, con respecto a la gravedad, que se le aplica al teléfono móvil	-Detectar movimiento -Detectar vibración -Detectar en qué posición se encuentra el Smartphone, para poder determinar si se debe girar la pantalla del móvil o no. -Detectar cuándo se produce un shake[1] con el dispositivo.
Giroscopio	Mide la aceleración angular que adquiere el Smartphone en un determinado momento.	-Detectar movimiento -Detectar vibración -Detectar en qué posición se encuentra el Smartphone -Sirve para usar Realidad Aumentada, Realidad Virtual, ver videos en 360°, -Jugar al Pokemon Go

[1] Agitar el SmartPhone

Sensor de Luz	Mide la cantidad de Luz que hay en el ambiente en donde se encuentre el teléfono	-Sirve para determinar el brillo de la pantalla de acuerdo a la luz del ambiente. -Sirve para determinar si es necesario usar el flash de la cámara al tomar una fotografía.
Sensor de Proximidad	Mide la distancia en que la se encuentra el Smartphone enfrente de un objeto.	-Sirve para apagar la pantalla cuando guardamos el teléfono en el bolsillo de un pantalón -Sirve para bajar automáticamente el volumen de los mensajes de audio de WhatsApp cuando ubicamos el teléfono cerca de nuestro oído.
Magnetómetro	Mide el campo magnético en los 3 ejes de coordenadas (x,y,z)	-Sirve para determinar la orientación en que se encuentra el dispositivo con respecto a los polos magnéticos - Se usa para crear una brújula
Sensor de temperatura	Mide la temperatura ambiente en donde se encuentre el teléfono	-Control de temperatura ambiente y del Smartphone
GPS	Entrega las coordenadas de localización en donde se encuentra el Smartphone	-Lo usan aplicaciones de geolocalización como Google Maps, Uber, Waze, entre otras.

Tabla 2 Sensores más comunes que traen los Smartphone

Existen muchísimos más sensores que los descriptos en la tabla anterior, y por ese motivo si desea conocer otros tipos de los que se puede encontrar en el Teléfono móvil, se recomienda leer el siguiente enlace:

- https://developer.android.com/guide/topics/sensors/sensors_overview?hl=es-419#java

b. Funcionamiento teórico del Acelerómetro y del Giróscopo.

El Acelerómetro es un sensor que poseen todos los teléfonos inteligentes debido a que es importante para la usabilidad del dispositivo. Su empleo más común es determinar la posición en que se encuentra el Smartphone en todo momento, para que el S.O decida si debe girar 90° la pantalla activa. El Acelerómetro captura la aceleración con respecto a la gravedad que adquiere el dispositivo en función a los tres ejes cartesianos: x, y, z.

El Giroscopio es un sensor que no se encuentra en todos los dispositivos y generalmente se halla en los Smartphone de gama media-alta. Permite obtener la aceleración angular que adquiere el dispositivo en un determinado momento, también en función de los tres ejes cartesianos. Es importante mencionar, que al igual que el Acelerómetro, utilizando el Giroscopio se puede determinar la posición en que se encuentra el Smartphone en un determinado momento. Sin embargo, si la posición del dispositivo se puede obtener usando uno u otro, ¿por qué se necesita que el teléfono tenga incorporado el Giroscopio para poder ver videos en 360° y usar realidad aumentada o virtual? La respuesta se contestará y detallará en los siguientes apartados, pero básicamente el Giroscopio permite detectar ciertos movimientos que el Acelerómetro no.

- **Acelerómetro**

Para comprender su funcionamiento, se considera a fines didácticos que el Smartphone contiene una masa sujetada por un resorte en el lado Positivo de cada eje (X+, Y+ , Z+), como podemos observar en la figura siguiente [3]:

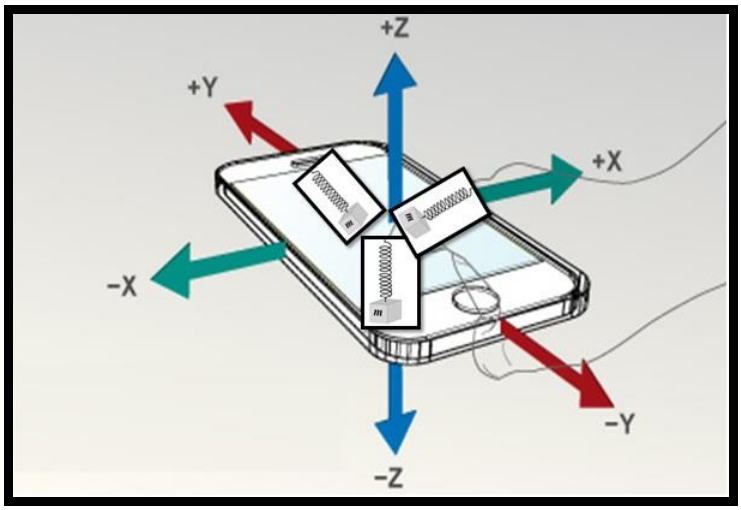


Fig. 14 Ejes de coordenadas del acelerómetro

De acuerdo a conceptos físicos, todo cuerpo independientemente de su masa experimenta una aceleración hacia el centro de la tierra de 9.81 m/s2, sin importar cuál sea su peso y posición [4]. En la física a este valor se lo conoce como 1G.

Fig. 15 aceleración de un cuerpo con respecto a la gravedad

En la Fig. 16 se muestra una masa muy pequeña unida a un resorte que se encuentra apoyado sobre una mesa. En este caso, el cuerpo se ve afectado por la fuerza de gravedad, pero esta no produce ningún movimiento sobre él debido a que se halla apoyado sobre la mesa. Por ende, se dice que la aceleración del cuerpo con respecto a la gravedad es de 0 G.

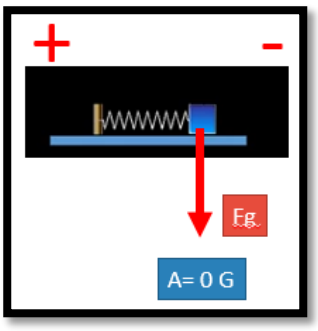


Fig. 16 Resorte apoyado sobre una mesa

Se considera al plano de la mesa como el eje X, y se asume que en su lado izquierdo se encuentran los valores positivos. Mientras que en su lado derecho están los negativos.

Ahora se gira la mesa junto con el resorte, de manera tal que la masa quede colgando hacia los valores negativos del eje x, como se muestra en la Fig. 17. Esto producirá que el resorte se estire una distancia determinada (Ax), producto de la fuerza de gravedad. No obstante también existe la fuerza del resorte que trata de contrarrestar dicho estiramiento.

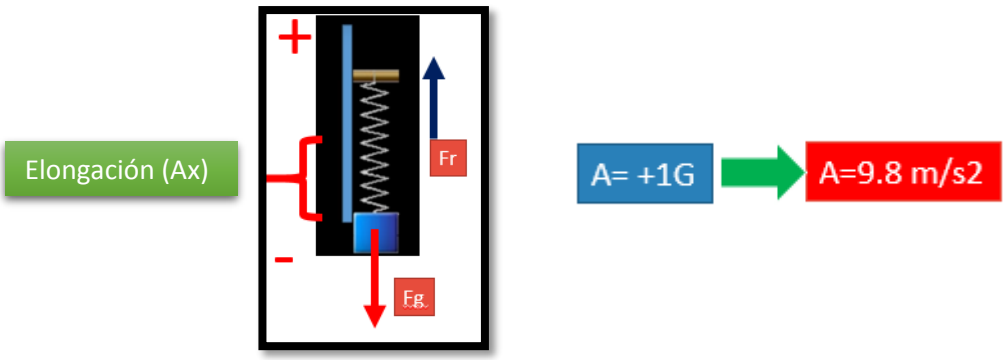


Fig. 17 Elongación del resorte con la masa que produce una aceleración de +1G

La fuerza de resorte (Fr) tratará de equilibrar hacia el eje +**,** contrarrestándolo aplicando una fuerza similar a la gravedad (Fg). Si no se aplica ninguna otra fuerza, esto produce como resultado una aceleración igual a la fuerza de gravedad de +9.8 m/s2. **Por ese motivo la aceleración es +1G**

A continuación se gira la mesa 180°, de manera tal que la masa comprima el resorte como se muestra en el Fig. 18. Esto producirá que este se comprima a una distancia determinada (Ax), producto de la fuerza de gravedad. No obstante también existe la fuerza del resorte que trata de contrarrestar la compresión.

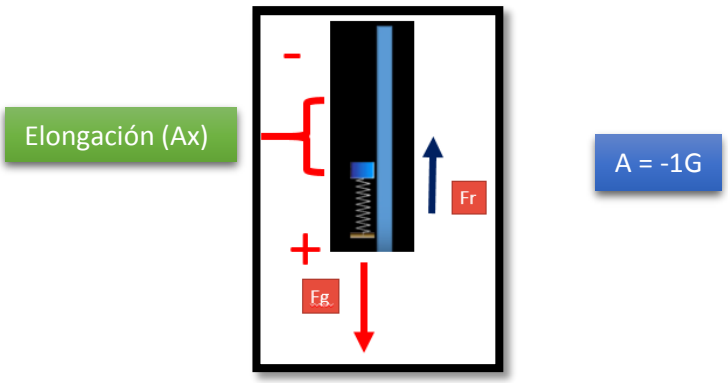


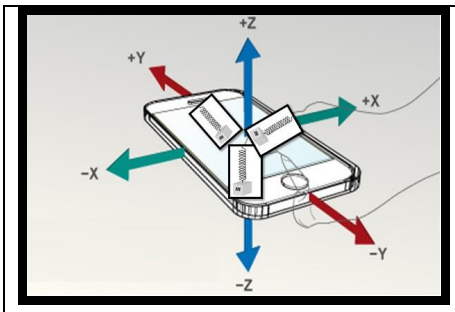
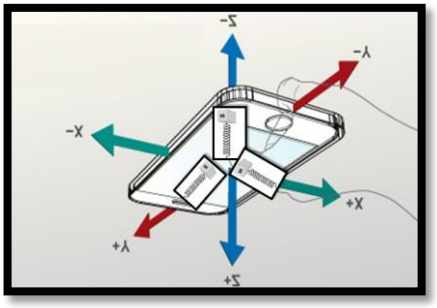
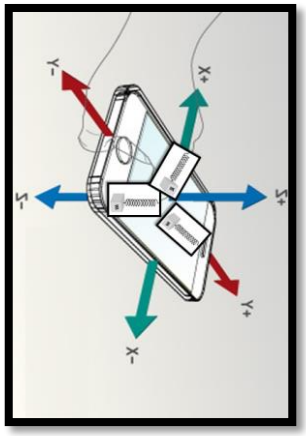
Fig. 18 Elongación del resorte con la masa que produce una aceleración de -1G

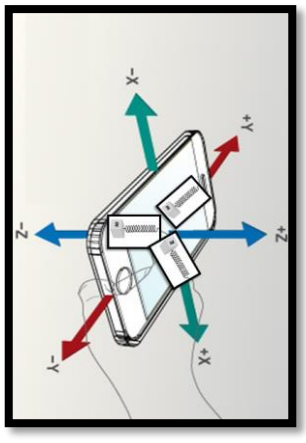
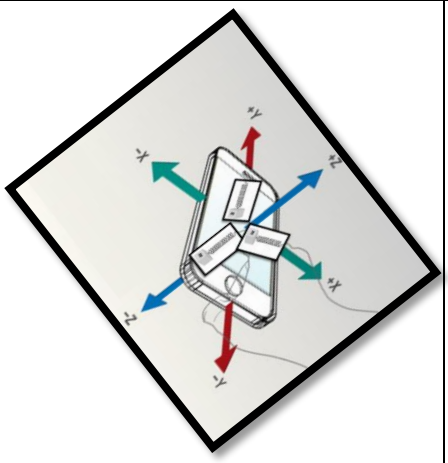
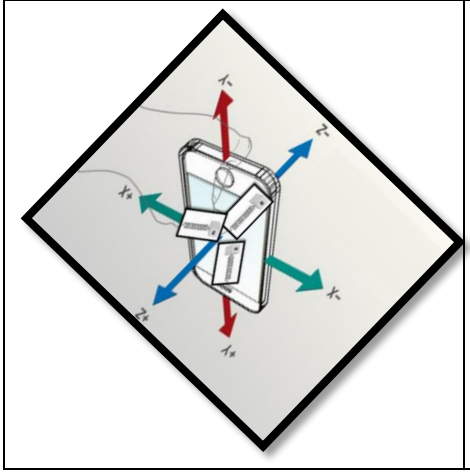
La fuerza de resorte (Fr) trata de equilibrar **hacia el eje -,** contrarrestándolo aplicando una fuerza similar a la gravedad (Fg). Si no se aplica ninguna otra fuerza, esto produce como resultado una aceleración igual a la fuerza de gravedad de -9.8 m/s2. **Por ese motivo la aceleración es -1G.**

En conclusión, se podría hacer la siguiente tabla como resumen nemotécnico del concepto didáctico del acelerómetro con el resorte:

Estado del resorte sobre el eje	Aceleración sobre el eje
Si el resorte se encuentra en reposo, no hay aceleración con respecto a la gravedad sobre ese eje	0 G
Si el resorte se encuentra estirado hacia los valores negativos, el valor de la aceleración con respecto a la gravedad sobre ese eje tiene un resultado positivo	+1G
Si el resorte se encuentra comprimido hacia los valores positivos, el valor de la aceleración con respecto a la gravedad sobre ese eje tiene un resultado negativo	-1 G

A continuación se aplicará el mismo concepto visto, pero ahora sobre el Smartphone. Para ello se usara una masa sujetada por un resorte en el lado Positivo de cada eje de coordenadas (X+, Y+, Z+), como se mostró en la Fig. 14. En consecuencia, analizaremos el valor que muestra el acelerómetro del teléfono en cada uno de los ejes. Para verificar dichos valores se puede usar la aplicación **Z-Device Test,** descargable desde Google Playstore.

	Si el Smartphone los apoyamos sobre una mesa, estarán inmóviles los resortes que se encuentran sobre el eje X e Y, por lo que no habrá aceleración. En cambio, el eje Z si detectará +1G de aceleración, debido a que el resorte se ubica estirado sobre ese eje y paralelo con respecto a la Fg.	**Acel X**= 0G **Acel Y**= 0G **Acel Z**= +1G
	Si al Smartphone los giramos 180° sobre la mesa, estarán inmóviles los resortes que se encuentran sobre el eje X e Y, y no habrá aceleración. En cambio, el eje Z si detectará -1G de aceleración, debido a que el resorte se ubica comprimido sobre ese eje y paralelo con respecto a la Fg.	**Acel X**= 0G **Acel Y**= 0G **Acel Z**= -1G
	Ahora giramos 90° al Smartphone, de forma tal que el eje X se encuentre paralelo a la Fg y su resorte, estirado. De esta manera el único eje que detectará aceleración será al X, con un valor de +1 G. Esto se debe a que los resortes de los ejes Z e Y se ubican inmóviles	**Acel X**=+1G **Acel Y**= 0G **Acel Z**= 0G

	Ahora giramos 180° al Smartphone, de forma tal que el eje X se encuentre paralelo a la Fg y su resorte, comprimido. De esta manera el único eje que detectará aceleración será el X, con un valor de -1 G. Esto se debe a que los resortes de los ejes Z e Y se ubican inmóviles	**Acel X**=-1G **Acel Y**= 0G **Acel Z**= 0G
	Ahora giramos el Smartphone, de forma tal que el eje Y se encuentre paralelo a la Fg y su resorte estirado. De esta manera el único eje que detectará aceleración será el Y, con un valor de +1 G. Esto se debe a que los resortes de los ejes Z e Y se ubican inmóviles	**Acel X**= 0G **Acel Y**= +1G **Acel Z**= 0G
	Ahora giramos el Smartphone, de forma tal que el eje Y se encuentre paralelo a la Fg y su resorte, comprimido. De esta manera el único eje que detectará aceleración será al Y, con un valor de -1 G. Esto se debe a que los resortes de los ejes Z e Y se ubican inmóviles	**Acel X**= 0G **Acel Y**= -1G **Acel Z**= 0G

Con esta información ya se tendría el conocimiento para saber el funcionamiento del Acelerómetro y, por lo tanto, se puede entender cómo el S.O Android determina cuándo debe girar la pantalla del dispositivo. Al mismo tiempo, se sería capaz de crear programas que permitan usar este sensor para determinar en qué posición se encuentra el celular. A su vez, usando trigonometría se podría saber con qué ángulo se encuentra ubicado con respecto al suelo.

- **Limitaciones del Acelerómetro**

No mide aceleración Lineal, sino que mensura la aceleración de un cuerpo con respecto a la Gravedad, como se explicó anteriormente. Por eso **NO** es muy viable calcular la velocidad y aceleración cinética de un cuerpo utilizando solamente el Acelerómetro.

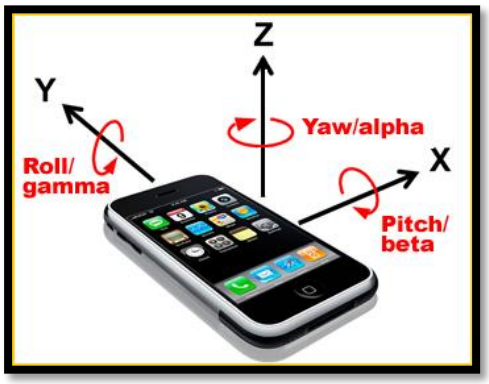


Fig. 19 Rotaciones del Smartphone sobre los ejes

Como se puede observar en la Fig. 19 [5], se conoce con el nombre de **Roll** al movimiento de girar el teléfono únicamente sobre su eje Y. Se llama **Pitch,** al movimiento de virar al móvil sobre su eje X y, **Yaw** cuando se lo hace sobre su eje Z. En este sentido, el acelerómetro puede detectar cambios de aceleración en sus ejes cuando se produce un movimiento Roll o Pitch. No obstante el acelerómetro no puede detectar cambios de aceleración cuando se produce el movimiento Yaw, siendo una gran limitación. Esto se puede comprobar si apoyamos el Smartphone sobre una mesa y posteriormente lo giramos sobre su eje, sin levantarlo. Al hacer esto observaremos que no detecta ningún valor. Para resolver dicha limitación se usa el Giroscopio, el cual permite detectar cambios de aceleración cuando se producen movimientos Pitch, Roll o Yaw. Por ese motivo es que las aplicaciones de realidad aumentada o Virtual, y los videos de 360° necesitan del Giroscopio para funcionar.

- **Giroscopio**

  Es un sensor que permite medir la velocidad angular con que se gira el Smartphone dentro de un intervalo de tiempo. Para poder comprender su funcionamiento se considera inicialmente al Smartphone que se muestra Fig. 20. En esta imagen el dispositivo se encuentra estático en la Posición A, por lo que el Giróscopo no detecta movimiento en ninguno de los tres ejes de coordenadas. Por ese motivo en la imagen se muestra que los valores de la velocidad angular que mostraría el giróscopo en los ejes X,Y, Z, valen 0 grados/seg.

  En la Fig. 21 se muestra que se rota al Smartphone desde el Punto A hasta el Punto B. En todo ese trayecto el dispositivo adquiere una determinada velocidad angular, que es capturada por el Giróscopo. En ese instante, el sensor devolverá valores distintos de cero en los tres ejes. Esta situación se representa en la figura con los valores z=3 °/s, x=2°/s, z=2°/s, que corresponde a la velocidad angular que adquiere el dispositivo en un determinado punto del trayecto. Finalmente cuando el Smartphone se ubica en la Posición B, ya no habrá más movimiento, debido a que llegó a su destino. Con lo cual el giróscopo no detectará más velocidad angular, y mostraría el valor de 0 °/s en los tres ejes.

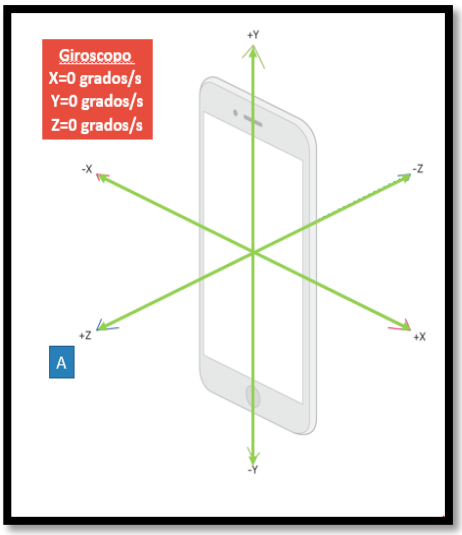


Fig. 20 Giróscopo detectando la velocidad angular del Smartphone en la Posición A

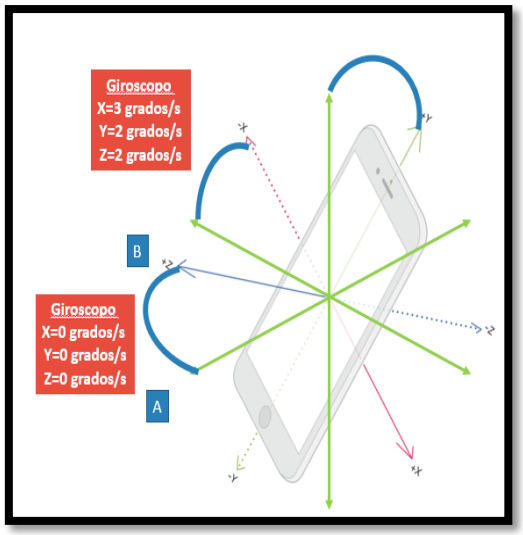


Fig. 21 Giróscopo detectando la velocidad angular del Smartphone desde la Posición A hasta la B

En resumen, el Giróscopo sensa la variación de velocidad que tiene el dispositivo cuando se gira, desde un punto A hasta un punto B. Durante el giro va a detectar la variación de velocidad, pero cuando el Smartphone llegue al punto B, el valor del giróscopo será de 0 grados/seg**.** Esto es muy importante porque en la práctica, cuando el teléfono se encuentra sin movimiento, el Giróscopo mostrará 0°/s.

- **Limitaciones del Giróscopo**

Con él, se podría detectar en qué posición se encuentra el Smartphone en un determinado momento, debido a que por medio de cálculos matemáticos se sabría en qué ángulo se encuentra. Pero si es lo mismo que hace el acelerómetro, ¿por qué en los Smartphone siempre vienen con él? ¿Por qué no vienen solamente con el giróscopo? La respuesta es que el giróscopo trae una limitación importante llamada Deriva [6]. La cual consiste en que a medida que se vaya capturando mediciones, se va ir generando un pequeño desplazamiento en la valores que se irán acumulando con el tiempo. Este pequeño error inicialmente será mínimo e imperceptible, pero con el tiempo será cada vez mayor. Con lo cual los valores finales no serán correctos. Por ese motivo, para solucionar las limitaciones que traen los acelerómetros y los giróscopos, muchas aplicaciones realizan una fusión de los datos de ambos sensores, obteniendo valores precisos. Existen distintos métodos para realizar la mencionada unión,siendo los más utilizados el Filtro Complementario [7] y el Filtro de Kalman [8]. El primero es el más sencillo de implementar, mientras que el segundo es el más preciso.

c. Utilización de sensores en Android

a. Clase del Framework para usar los Sensores

El S.O Android le ofrece al programador un conjunto de herramientas a través de su Framework, que le permite acceder a los sensores del Smartphone y adquirir los datos que capten del ambiente. Por consiguiente, a continuación se describen las clases del Framework para poder utilizar los sensores en Android [9]:

- **SensorManager:**
  Se utiliza para crear una instancia (objeto) del servicio del administrador de los sensores del sistema. Es el punto de entrada para realizar la gestión sobre los mismos. Empleándolo

podremos manejar métodos para acceder a ellos, obtener una lista de los que tiene el dispositivo y, registrar y cancelar el registro a un listener, que recogerá las datos que capture un sensor determinado cuando estos sufran una variación.

- **Sensor:**
  Se utiliza para crear un objeto asociado a un determinado sensor y, por medio de este, acceder a las características que posea. Cada sensor deberá estar asignado a un objeto Sensor. Así por ejemplo, si quisiéramos usar el acelerómetro y el sensor de luz del Smartphone, deberíamos crear dos objetos de la clase Sensor. Uno para el acelerómetro y otro para el sensor de luz.

- **SensorEventListener:** Es una clase interfaz que posee los métodos (listener), que detectan cuando se produce una variación en los valores de los sensores a los que se encuentra suscripto a través del SensorManager. Esta suscripción se debe realizar a través de la función *registerListner*. Como toda Interfaz en Java, dispone de dos métodos abstractos que deberá implementar el programador para capturar los eventos de los sensores [10].

- **SensorEvent:**
  Cuando se invoca un listener, en el momento en que el sistema detecta un cambio en un sensor que se encuentre registrado a través del SensorManager, se puede utilizar un objeto de la clase SensorEvent para obtener las características que tenga en ese momento. Así cuando se ejecute el listener, a través de un objeto SensorEvent se puede obtener el valor leído por el sensor.

b. Métodos de la Interfaz SensorEventListener

Una vez que se registran los sensores que se quieren supervisar, a través del **SensorManager**, las variaciones en sus valores serán capturados por dos listeners distintos. Estos dos métodos corresponden a la Interfaz **SensorEventListener**. Al ser esta una clase Interfaz, los dos escuchadores deberán ser implementados por el programador en el código de su programa. En consecuencia el S.O Android llama a estos métodos cuando ocurre lo siguientes eventos:

- **OnAccuracyChanged():** será invocado cuando se produzca un cambio de precisión en los valores que captura un determinado sensor. En otras palabras, este método se ejecutará cuando un sensor, de los que fueron registrados para escuchar a través del SensorManager, cambie su fiabilidad en los valores que sensa. De esta manera se podrá saber si esos valores son exactos. En caso de que no lo fueran, el programador podrá determinar si conviene recalibrarlo o si está funcionando mal. [11]

- **OnSensorChanged():** Será invocado cuando un sensor, que fue registrado a través del SensorManager, detecte un cambio en los valores que sensa. Por lo tanto se ejecutará este método cada vez que varié el valor capturado.

Si el programador registra varios sensores, a través del SensorManager, entonces los métodos OnAccuracyChanged y OnSensorChanged van a ser invocados por cualquier cambio que sufran

dichos elementos. Por lo tanto para poder identificar qué sensor fue el que invocó al listener, estos métodos reciben como parámetro un objeto que identifica al mismo.

c. Pasos a seguir para usar el Framework de Sensores en Android

A continuación se explican los pasos a seguir para la creación de un programa que pueda utilizar los sensores a través del Framework de Android. Para ello seguiremos un ejemplo de cómo usar el sensor de Luz del Smartphone [9], el cual se muestra en la siguiente figura.

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

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

        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mLight = sensorManager.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();
        sensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
    }

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

(1) (2) (3) (4)

Fig. 22 Ejemplo de uso del sensor de luz usando el Framework de Android

Primeramente se obtiene una instancia del Servicio del Administrador de Sensores del S.O. Para ello, se crea un objeto SensorManager, como se muestra en el punto **(1)** de la figura anterior. Luego se debe registrar en el SensorManager, el sensor que se va a controlar. Convenientemente se realiza al comenzar a ejecutar la Activity. Es el caso del método onResume(), como se muestra en el punto **(2)**. Posteriomente cuando se oculta la Activity, se debe desregistrar el sensor del SensorManager, como se muestra en el punto **(3)** con el método OnPause(). Finalmente en el punto **(4),** se deben implementar los métodos de la Interfaz SensorEventListener.

Es importante mencionar que dentro del método OnSensorChanged(), se pueden obtener los valores capturados por los sensores en un determinado momento. En el ejemplo mostrado se

obtiene el valor del sensor de Luz, almacenándolo en la variable lux. De esta forma este dato podría ser posteriormente utilizado en el programa.

d. Ejemplo de proyecto de Android que utiliza Sensores

En el siguiente repositorio se encuentra el código fuente de un proyecto Android que utiliza los distintos sensores que posee un Smartphone:

- https://gitlab.com/so-unlam/Material-SOA/-/tree/master/Ejemplos/Android/Sensores/sensores1

7. Ejecución en Background en Android

El S.O Android crea un proceso por cada aplicación nueva que se ejecute. Esto quiere decir que cada aplicación Android correrá dentro de su propio proceso del sistema operativo. Por defecto, este proceso está conformado por un único hilo, que es llamado **Hilo Principal (Main Thread)** [12] [13]**.** Este punto es importante, porque en el Hilo principal se ejecutan todos los componentes que conforman esa aplicación: el código de la parte gráfica y lógica de todas las Activitys, todos los componentes que conforman a la aplicación, servicios, Broadcast, etc. El S.O administra a las aplicaciones de esta manera, por una cuestión de eficiencia en su funcionamiento. Por ese motivo si un programador creara una aplicación que realizará una operación en su Hilo Principal que tardase mucho tiempo en completarse, por ejemplo dentro del método OnCreate() de una Activity, el programa se bloqueará totalmente, dado que se estará ejecutando todo el código en el mismo hilo. Como consecuencia el usuario no podrá interactuar con la Interfaz gráfica de la Activity que se encuentra activa en ese momento, ya que no verá respuesta alguna [14]. Por ese motivo, si luego de 5 segundos el S.O ve que la aplicación sigue sin responder, automáticamente mostrará el mensaje de error de la Fig. 23. En caso de que el usuario acepte forzar el cierre, a través del cuadro de alerta, el S.O terminará la aplicación y eliminará su proceso de la memoria.

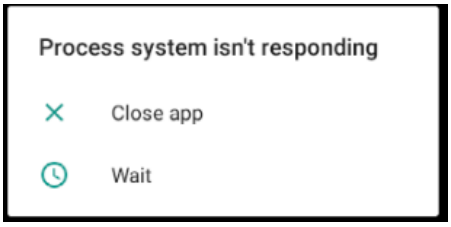


Fig. 23 Mensaje de APP NO Responde

Por las razones expuestas, se recomienda que las tareas bloqueantes o que requieran mucho tiempo de procesamiento no se realicen en el Hilo Principal de las aplicaciones. Operaciones bloqueantes pueden ser: realizar bucles, esperas activas, etc. Lo recomendado es que estos tipos de tareas sean ejecutadas en segundo plano a través de algunas de las siguientes herramientas:

- A) Thread
- B) AsyncTask
- C) Service

A continuación se detallarán cada una de ellas:

a. Thread

Es un hilo de ejecución de un programa de Android. La máquina Virtual de Java del S.O Android permite que una aplicación pueda contener varios hilos ejecutándose en forma simultánea. Como se mencionó previamente, cuando se abre una aplicación por primera vez, el S.O crea su proceso con un único Thread, llamado Hilo Principal. Este es el encargado de actualizar las interfaces graficas de las Activities. En otras palabras, es el responsable de mostrar en pantalla lo que se quiere informar al usuario. Por consiguiente, si se creara otro Thread, cuya tarea sea realizar una operación que requiera mucho procesamiento, el S.O no permitirá que desde ese Hilo Secundario se muestre el resultado de la operación directamente por pantalla. Esto es debido a que Android tiene esta restricción por motivos de seguridad. El S.O pone como condición que Hilo Principal sea el único que puede actualizar las interfaces gráficas. De esta manera, si un Hilo secundario quisiera mostrar datos por pantalla, se los deberá enviar al Hilo Principal para que posteriormente se los muestre al usuario.

El Main Thread de Android utiliza una cola de mensaje, que permite a los Hilos secundarios poder comunicarse de manera asincrónica con él. Pero para que puedan utilizar dicho buffer, deberán utilizar un manejador llamado **Handler**. De esta forma un Hilo Secundario podrá enviarle datos al Hilo Principal en cualquier momento, colocándolos en la cola de mensajes, y luego el Main Thread los leerá cuando no este ocupado. En la Fig. 24 se expone este concepto.

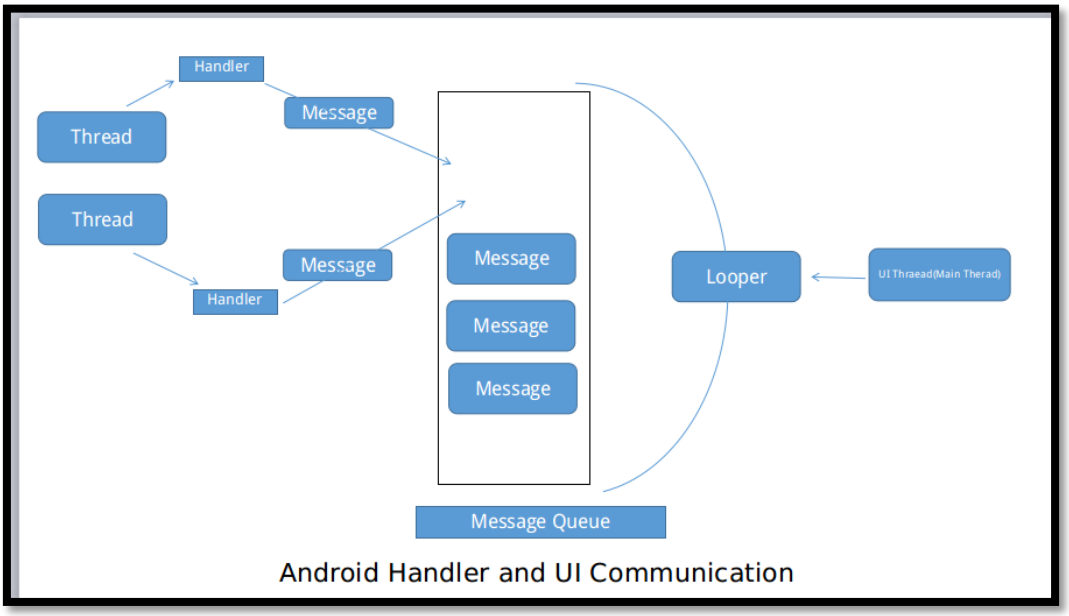


Fig. 24 Comunicación de Hilos Secundarios con el Hilo Principal

En la Fig. 25 se muestra un ejemplo de cómo comunicar un Hilo Secundario con el Main Thread a través de su cola de mensajes usando un Handler.

MainActivity.java

```
public class MainActivity extends Activty
{
    ConnectedThread mConnectedThread ;
    Handler bluetoothIn;

    protected void onCreate()
    {
          mConnectedThread = new ConnectedThread(btSocket);
          bluetoothIn = Handler_Msg_Hilo_Principal();

          mConnectedThread.start();
    }

   private Handler Handler_Msg_Hilo_Principal ()
   {
       return new Handler() {
             public void handleMessage(msg)
             {
                   Toast.MakeText(msg);
             }
         }
  }
.....
....
   private class ConnectedThread extends Thread
   {
      .........
      public void run()
      {
         .........
         bluetoothIn.obtainMessage(msg).sendToTarget();
      }

  }
```

(3)

(2)

(1)

Los datos se muestran en pantalla

Fig. 25 Ejemplo de comunicación entre Hilo secundario y Thread Principal

En el ejemplo, se puede observar la clase MainActivity que corresponde al Hilo Principal del programa. En el punto **(1)** se define la clase del Thread secundario. Cuando se ejecuta su método run (), se inicia su ejecución en segundo plano. Como esta función pertenece a un Hilo secundario, dentro de ella el S.O no permite mostrar datos por pantalla al usuario. Por ese motivo se envía un mensaje al Hilo principal a través del Handler llamado **bluetoothIn**. De esta manera se colocan los datos, que se desean informar al usuario, en la cola de mensajes del Hilo Principal. Una vez hecho esto, automáticamente se ejecuta en forma asincrónica el método **Handler_message() (2).** Dentro de esta función se definió el Handler que permite mostrar por pantalla los datos fueron encolados. Por otra parte en el punto **(3)**, se muestra de qué forma se debe iniciar la ejecución del segundo Thread y el Handler mencionado.

b. AsyncTask

Permite generar Hilos secundarios, en segundo plano, que pueden mostrar resultados en la pantalla fácilmente a través del Hilo Principal. AsyncTask evita tener que utilizar Handler y Thread, evadiendo así problemas de sincronización. Esto se debe a que un hilo AsyncTask se crea en cuatro pasos bien definidos, a través de sus métodos:

- **onPreExecute():** se ejecuta en el Hilo Principal, siendo el primero en ejecutarse cuando se invoca a la instrucción **execute().** En esta función el programador puede colocar código para inicializar el Hilo secundario.
- **doInBackground():** Es el único que se ejecuta en un Hilo Secundario. En esta función el programador debe colocar todo el código que desee que se ejecute en segundo plano, en un hilo aparte. Como el código de este método se inicia en un hilo secundario, desde aquí el S.O no permite mostrar datos por pantalla en forma directa. Para realizar dicha tarea, la clase AsyncTask posee dos métodos que permiten al Hilo Secundario comunicarse con el Principal en forma indirecta a través de ellos. Esto se debe a que dichos métodos de la Clase AsyncTask se ejecutan dentro del contexto del Hilo Principal. Los dos métodos mencionados son onProgressUpdate() y onPostExecute().

- **onProgressUpdate():** debe ser llamado dentro de doInBackground, para poder mostrar datos por pantalla a medida que se ejecuta el Hilo Secundario. Para poder invocar a este método desde doInBackground(), se debe usar la instrucción publishProgress().
- **onPostExecute():**Este método es llamado tras finalizar doInBackground(), o sea cuando finaliza la ejecución del Hilo secundario. Recibe como parámetro el resultado devuelto por doInBackground(), permitiéndolo mostrar por pantalla a través del hilo Principal.

Es importante destacar nuevamente que de los cuatro métodos mencionados de la clase AsyncTask, doInBackground() es el único que se ejecuta en un Hilo secundario. Mientras que onPreExecute(), onProgressUpdate() y onPostExecute() lo hacen en el hilo Principal.

Cuando desde el Hilo principal se crea un objeto de la clase AsyncTask, debe iniciar la ejecución del Hilo Secundario con el comando **execute()**. De esta manera las ejecuciones de una misma AsyncTask se realizan de forma secuencial, esto es, hasta que no finalice la ejecución en curso no se realizará la siguiente ejecución solicitada. Cada vez que se ejecute el comando execute, se irán apilando las invocaciones al hilo. Con lo cual, si deseamos poder realizar varias ejecuciones en paralelo del mismo hilo deberíamos lanzarlas con el método **executeOnExecutor().**

En la siguiente figura se muestra un ejemplo de utilización un Hilo secundario creado con la Clase AsyncTask.

MainActivity.java

```
public class MainActivity extends Activty
{
   protected void onCreate()
   {
        ThreadAsynctask hilo=new ThreadAsynctask();
        hilo.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR,params);
   }

   ------
   ------

  private class ThreadAsynctask extends AsyncTask
  {
     protected void onPreExecute()
     {
       .........
     }

    protected String doInBackground(String... params)
    {
       publishProgress("Hola");
       .........
       String msg = params[0]
       return msg;
    }

   protected void onPostExecute(String result)
   {
     Toast.MakeText(result);
   }

   protected void onProgressUpdate(String… Msg)
   {
     Toast.makeText(Msg);
   }
}
```

(1) (2) (3) (4)

Fig. 26 Ejemplo de un Hilo secundario con la clase AsyncTask

En el punto (1) se crea un objeto de una clase AsyncTask y se inicia la ejecución de su Hilo secundario con el comando **executeOnExecutor().** Luego de ejecutarlo, automáticamente se llama al método **onPreExecute**(), punto (2). Al finalizar dicha función, se lanza el Hilo secundario ejecutando para eso al método **doInBackground**() (3). Durante la ejecución de dicho método se

muestra por pantalla el mensaje “Hola”. Esto se realiza al utilizar la instrucción **publishProgress**, la cual invoca al **onProgressUpdate().** Por último, cuando finaliza doInBackground se invoca al método onProgressUpdate, el cual muestra por pantalla el dato retornado por doInBackground (4).

c. Service

Se puede crear un servicio heredando de dos clases:

- Service
- IntentService

Se analizará el primer caso: si se crea una servicio extendiendo de la clase **Service**, por defecto, cuando este se inicie desde una Activity se ejecutará en su mismo hilo. En otras palabras, se ejecutará en el Main Thread. Por consiguiente, se bloqueará la ejecución de la aplicación si el servicio realizara una operación que requiera mucho procesamiento. Para que esto no suceda dentro del servicio se debe crear un Thread secundario, que en segundo plano realice las operaciones que requieran mucho tiempo de procesamiento y cuando finalice su tarea le envié los resultados a la Activity que inicio el servicio. Existen distintas técnicas que permite que un Servicio pueda intercambiar datos con una Activity, tales como usar BraodcastReceiver, Handler0, objetos Messenger [15], entre otros. La que va a ser explicada en este documento es la de BraodcastReceiver. Para poder comprender su utilización consideremos el siguiente ejemplo: se crea un servicio, que es iniciado desde una Activity, el cual origina un Hilo secundario para que realice una operación y devuelva su resultado a la misma para que la muestre en pantalla. Para enviar el resultado el Hilo secundario utilizará un Broadcast. En la Fig. 27 se muestra la clase del servicio del ejemplo descripto.

```
public class ServicioMusica extends Service
{
    //Metodo que se llama cuando se crea el servicio con startservice
    public void onCreate()
    {
        //Se crea el Hilo secundario del Servicio
        ServiceThread threadService = new ServiceThread();
        threadService.start();
    }

    //Despues de onCreate automaticamente se ejecuta a OnstartCommand,
    public int onStartCommand(Intent intenc, int flags, int idArranque)
    {
        .....
    }

    //On destroy se invoca cuando se ejcuta stopService
    public void onDestroy()
    {
        ......
    }

    //clase del Hilo Secundario del Service
    public class ServiceThread extends Thread
    {
        public void run()
        {
            enviarBradcast();
        }
    }

    public void enviarBradcast()
    {
        Thread.sleep(10000);

        Intent i = new Intent("com.example.intentservice.intent.action.RESPUESTA_OPERACION");
        i.putExtra("mensaje", "Hola que tal");
        //Senvia el mensaje al "hola que tal" al Broadcast reciever de la activity principal
        sendBroadcast(i);
    }
}
```

(1) (2) (3)

Fig. 27 Clase del Service con el Hilo secundario

Cuando se inicia el servicio, el primer método que se ejecuta es onCreate()**(1).** Se muestra que dentro de esta función, se crea al Hilo secundario llamado ServiceThread. La clase perteneciente a este Thread se observa en el punto **(2).** Comenzando a ejecutarse el Hilo secundario se ocasiona una espera activa de 10 segundos, que no bloqueará a la aplicación, y posteriormente se envía un Broadcast a la Activity con el mensaje “Hola que tal”, realizado en el punto **(3).**

La clase perteneciente a la Activity Principal que inicia el servicio se muestra en la figura Fig. 28

```
public class MainActivity extends Activity {
    public IntentFilter filtro;
    private ReceptorOperacion receiver =new ReceptorOperacion();
    Intent intent;

    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //se inicia el servicio
        intent=new Intent(MainActivity.this, ServicioMusica.class);
        startService(intent);
        //se configura el Broadcast Receiver que comunicará el Servicio con la Acitivty
        configurarBroadcastReciever();
    }

    public void onDestroy() {
        super.onDestroy();
        stopService(intent);
    }

    //Metodo que crea y configurar un broadcast receiver que recibira
    //los mensaje que envia el servidor a la activity
    private void configurarBroadcastReciever()
    {
        //se asocia(registra) la  accion RESPUESTA_OPERACION, para que cuando
        //el Servicio ejecute un Bradcast con esa accion,automaticamente se invoque
        //al metodo OnRecive del la clase ReceptorOperacion
        filtro = new IntentFilter("com.example.intentservice.intent.action.RESPUESTA_OPERACION");
        filtro.addCategory(Intent.CATEGORY_DEFAULT);
        registerReceiver(receiver, filtro);
    }

    //Clase BroadcastReceiver que recibira el msj que envia el servicio a través del Braodcast
    public class ReceptorOperacion extends BroadcastReceiver
    {
        public void onReceive(Context context, Intent intent) {
            //Se obtiene el mesnaje que envio el servicio a traves de un untent
            String mensaje = intent.getStringExtra("mensaje");
            ......
        }
    }
}
```

(1) (2) (3)

Fig. 28 Clase de la Activity que inicia el Servicio

Se puede observar el momento en que se inicia el servicio dentro del método onCreate() de la Activity **(1)**. A su vez, en dicha función se ejecuta la configuración del BraodcastReceiver encargado de recibir los mensajes que le enviará el Service a la Activity. Esta configuración es llevada a cabo en el punto **(2),** en donde se define la clase RecepetorOperación. Dentro de esta clase, se recibirán los datos que enviará el Hilo secundario del Servicio cuando ejecute el Broadcast, además se almacenarán en una variable String con la finalidad de mostrarlos en pantalla y para su posterior procesamiento. Es importante mencionar, que se debe detener el servicio cuando se destruye la Activity **(3)**, sino seguirá ejecutandose en segundo plano por más que se haya cerrado la aplicación.

Si se creara el servicio utilizando la clase **IntentService**, en lugar de heredar de la clase Service, no sería necesario crear el Thread dentro del servicio. Esto se debe a que cuando se crea un objeto de una clase IntentService, automáticamente el S.O crea un Hilo secundario asociado a dicha clase. Con lo cual el servicio IntentService se ejecutará directamente en otro hilo diferente del Hilo Principal. No obstante para intercambiar datos entre el Hilo Secundario del Servicio IntentService y la Activity llamadora, se pueden utilizar las mismas técnicas que se mencionaron anteriormente. Utilizándose BraodcastReceiver, Handler, objetos Messenger [15], entre otros. Para conocer de qué forma será utilizado un BraodcastReceiver en un Servicio que herede de la clase IntentService, se recomienda leer el siguiente tutorial:

- http://www.sgoliver.net/blog/tareas-en-segundo-plano-en-android-ii-intentservice/

8. QUÉ ES UN WEB SERVICES Y CÓMO FUNCIONA CON EL PROTOCOLO REST

Es una tecnología utilizada para intercambiar recursos entre un cliente y un servidor. Los web services siguen un estándar de comunicación y consumir recursos expuestos por un servidor. Uno de los estándares de comunicación más conocidos es REST. REST es una arquitectura basada en el protocolo HTTP para comunicar distintas sistemas como cliente/servidor. Sus características principales son:

- Establece un formato de representación de transferencia para peticiones y respuestas entre un cliente y un servidor. Ejemplo: xml o json.
- Acceso a recursos específicos determinado a través de URIs, URLs o URNs. Ejemplo: "localhost: 8080/recurso/consumir".
- Métodos específicos para intercambio de datos. Ejemplo: GET, POST, PUT, DELETE.
- Comunicación sin estado. Esto se debe a que las ejecuciones son independientes porque cada petición a servidor contiene toda la información necesaria para su ejecución. Ejemplo: No requiere el uso de memoria cache para mantener estados previos.

Inicialmente cuando un Cliente quiere solicitar un determinado servicio a un servidor, le envía un mensaje de petición siguiendo un formato específico. Esta petición se la denomina **HTTP Request**. En dicho mensaje es obligatorio indicar cuál es la URI del servicio que le solicita al servidor. Opcionalmente se le pueden adicionar parámetros de ejecución al mensaje, en formato XML o JSON. Una vez que el servidor recibe el mensaje de petición, lo procesa, luego realiza la acción solicitada por el cliente y finalmente le envía un mensaje de respuesta denominado **HTTP Response**. En él, se le notifica al cliente el estado de la operación, indicándole si hubo un error durante su realización o si se pudo llevar a cabo correctamente. De esta forma este mensaje funciona como una confirmación de operación, ACK. Adicionalmente el servidor le puede enviar al cliente información adicional de la operación, dentro del mensaje del HTTP Response. Estos datos también pueden estar adicionados en formato XML o JSON. En la figura siguiente se puede observar dicho proceso de petición.

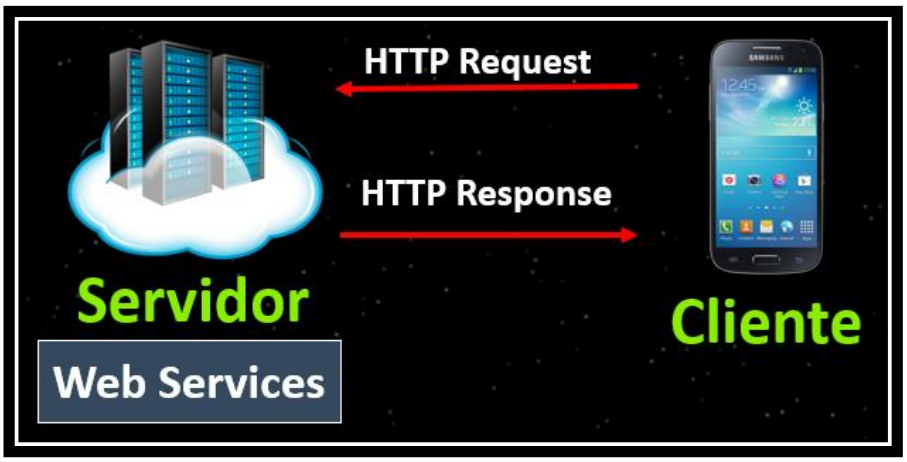


Fig. 29 Proceso de petición y respuesta a un servicio de un web service

9. Explicar métodos de comunicación con servidores.

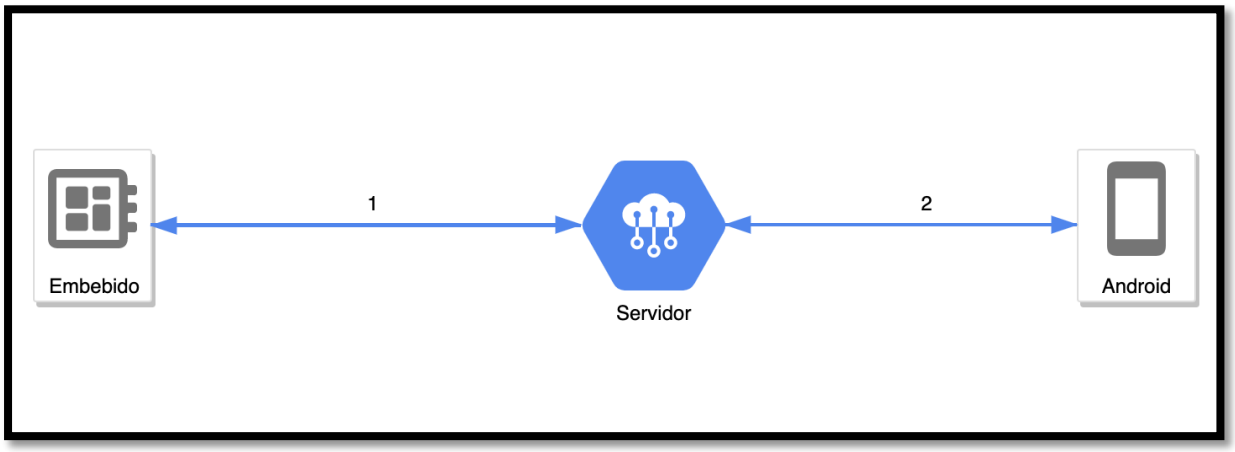


Fig. 30 Comunicación entre Cliente y Servidor

Los métodos de comunicación que se pueden realizar con un servidor pueden ser **sincrónicas** y **asincrónicas**.

- En el método de comunicación **sincrónica** se realiza una petición al servidor y este detiene su ejecución hasta que retorne una respuesta o falle por excepción. En Android, este tipo de método puede generar que la aplicación no responda debido a que detiene la ejecución del Hilo principal de Android, generándose así una espera bloqueante, inhabilitando la interfaz graficas de las Activities. Por este motivo, se tiene que realizar la petición en paralelo en segundo plano, y luego volver al Hilo principal cuando se reciba la respuesta del servidor. En consecuencia, para realizar la consulta al servidor en Background, se pueden usar las técnicas que fueron explicadas en el apartado sobre **Ejecución en Background en Android**. Estas son: Thread, AsyncTask o Service (IntentService). En la figura Fig. 31 se muestra la forma de utilización de los mecanismos de comunicación sincrónica con el Servidor. En el grafico se puede observar que en un Hilo Secundario se realizan las tareas de envíos de mensajes HTTP Request y de recepción de los HTTP Response. Mientras que por otro lado, el Hilo Principal se encarga de realizar otro tipo de tareas, como actualizar las interfaces gráficas.

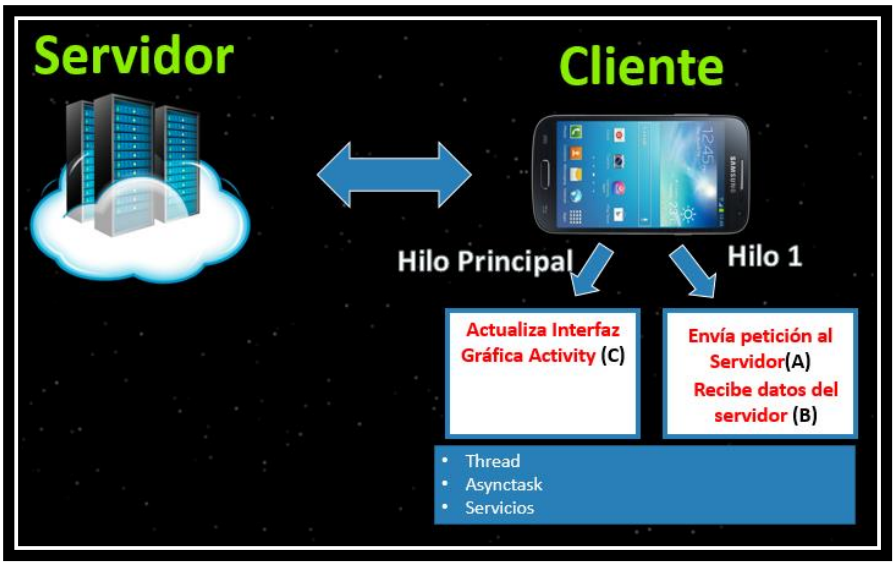


Fig. 31 Método de comunicación Sincrónica con el Servidor

En el Framework de Android existen varias bibliotecas que le permiten a una aplicación Android intercambiar mensajes REST con un servidor de forma sincrónica. Un ejemplo de este tipo de bibliotecas, es **HttpURLConnection.** En la figura siguiente se muestra brevemente un resumen sobre los pasos que deben realizarse para poder utilizarla dentro de un proyecto Android.

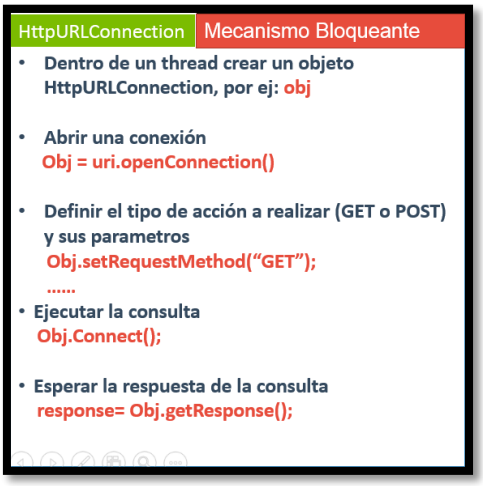


Fig. 32 Resumen sobre la utilización de la Biblioteca HttpURLConnection

- En el método de comunicación **asincrónico** se realiza la petición al servidor también a través del envío de un mensaje HTTP Request, pero este no detiene el flujo principal de ejecución. Por lo tanto el Main Thread puede seguir ejecutando instrucciones del programa hasta que el servidor retorne su respuesta o genere una excepción. En este tipo de comunicación la recepción de los mensajes de respuesta HTTP Response que envía el servidor al cliente, son recibidos en forma asincrónica a través de métodos Handlers, llamados **Callback**. Estos son invocados automáticamente por el S.O, cuando el sistema recibe un mensaje HTTP Response como respuesta a un determinado HTTP Request. En la siguiente figura se muestra la forma de utilización de los mecanismos de comunicación asincrónica con un servidor.

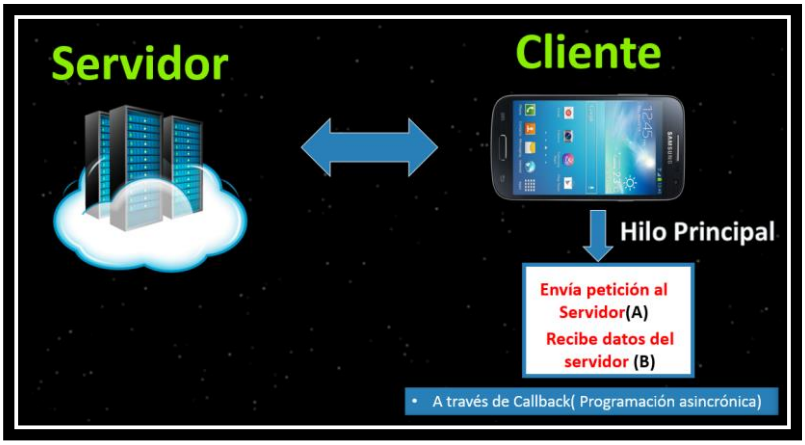


Fig. 33Método de comunicación Asincrónica con el Servidor

Como se puede observar, se debe tener cuidado cuando se utiliza este tipo de comunicación. Esto se debe a que el envío y recepción de mensajes se realizan en el mismo hilo donde se ejecuten dichos métodos. Por ende si se ejecutan en el hilo Principal, también lo hará el método Callback que capturará los mensajes HTTP Response. En consecuencia si dentro de dicho método se realiza una operación bloqueante, se detendrá la ejecución del Hilo principal, inhabilitando así la pantalla.

Por ese motivo no se aconseja realizar operaciones bloqueantes en los métodos Callback que capturan los HTTP Response.

En el Framework de Android existen varias bibliotecas que permiten a una aplicación Android intercambiar mensajes REST con un servidor de forma asincrónica. Un ejemplo de este tipo es Retrofit**.** En la siguiente figura se muestra brevemente un resúmen sobre los pasos que deben realizarse para poder utilizarla dentro de un proyecto Android.

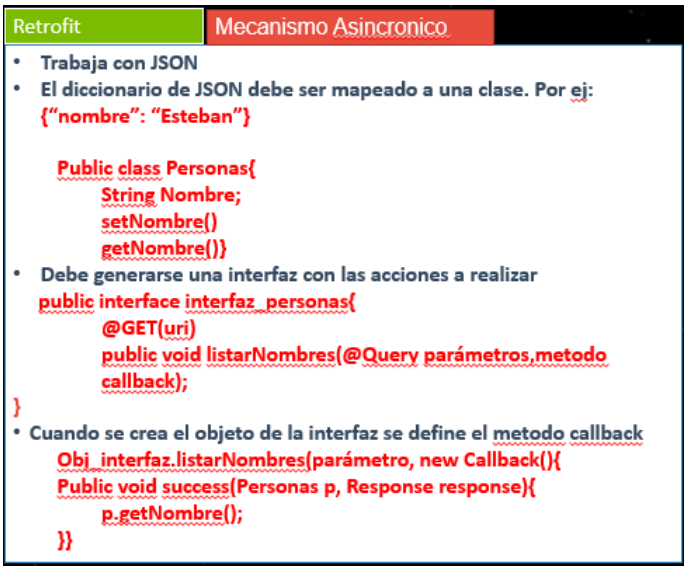


Fig. 34 Resumen sobre la utilización de la Biblioteca Retrofit

10. FIREBASE

Es una plataforma de servicios que pueden ser implementadas en aplicaciones tanto web como Mobile. Para este apartado, vamos a explicar qué son las notificaciones push, cómo funcionan y cómo configurarlo.

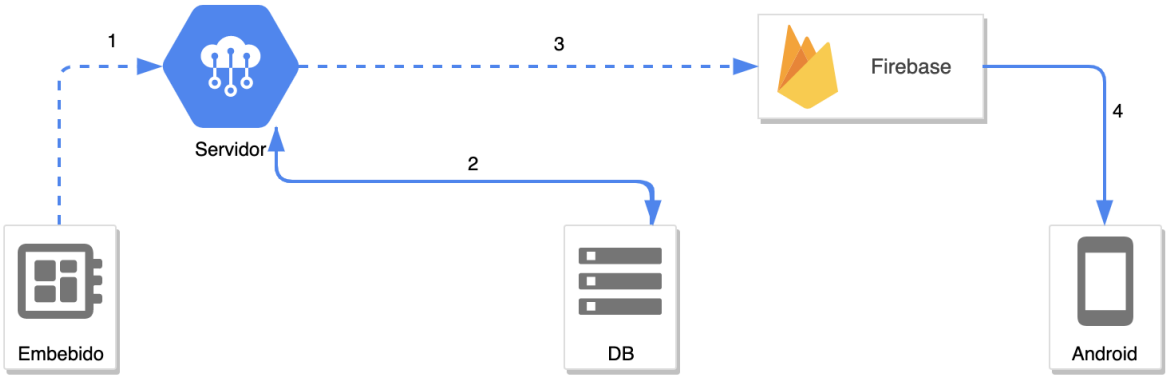


Fig. 35 Comunicación entre un Cliente-Firebase y un servidor externo

Firebase Push Notification es una tecnología que mantiene una conexión entre una aplicación y un servidor con el propósito de enviar un aviso a una aplicación o grupo de aplicaciones suscriptas de acuerdo a un evento definido. Para realizar esto, tiene que estar extendida de un servicio llamado `FirebaseMessagingService` y los mensajes serán recibidos por la función llamada `onMessageReceived`. A su vez, la conexión al servidor de notificaciones puede realizarse de dos

formas. La primera, es a través de un Registro individual al servidor de notificaciones, el cual les generara un token como identificador único que el servidor utilizará para determinar a quién enviar el mensaje. El código que se debe utilizar en el proyecto Android para obtener poder obtener el Token de Firebase se muestra en la siguiente imagen.

```
// Get token
// [START retrieve_current_token]
FirebaseInstanceId.getInstance().getInstanceId()
        .addOnCompleteListener(new OnCompleteListener<InstanceIdResult>() {
            @Override
            public void onComplete(@NonNull Task<InstanceIdResult> task) {
                if (!task.isSuccessful()) {
                    Log.w(TAG, "getInstanceId failed", task.getException());
                    return;
                }

                // Get new Instance ID token
                String token = task.getResult().getToken();

                // Log and toast
                String msg = getString(R.string.msg_token_fmt, token);
                Log.d(TAG, msg);
                Toast.makeText(MainActivity.this, msg, Toast.LENGTH_SHORT).show();
            }
        });
// [END retrieve_current_token]
```

Fig. 36 Código para obtener el Token de Firebase

La segunda forma consiste en una suscripción a un topic en particular. De manera tal que se siga el modelo de comunicación Publicador-Suscriptor. En la siguiente imagen se muestra el ejemplo de código de Android para realizar la suscripción a un topic de clima.

```
// [START subscribe_topics]
FirebaseMessaging.getInstance().subscribeToTopic("weather")
        .addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                String msg = getString(R.string.msg_subscribed);
                if (!task.isSuccessful()) {
                    msg = getString(R.string.msg_subscribe_failed);
                }
                Log.d(TAG, msg);
                Toast.makeText(MainActivity.this, msg, Toast.LENGTH_SHORT).show();
            }
        });
// [END subscribe_topics]
```

Fig. 37 Código de suscripción a un topic de Firebase

En consecuencia cuando Firebase nos envié una notificación al topic clima, al que se encuentra suscripto nuestro programa, será capturado por la función que se muestra en la Fig. 37. En dicho

método se podrán recibir los parámetros que envíe Firebase a nuestra Aplicación, por medio de la notificación al topic de Clima.

a. Pasos para configurar las notificaciones de Firebase

Para configurar las notificaciones, se tienen que seguir los siguientes pasos:

- Crear un proyecto desde la consola de firebase.
- Descargar el archivo de configuración llamado `google-services.json` y agregarlo en la raiz del proyecto.
- Editar el archivo build.gradle que se encuentra a nivel de proyecto, y agregar entre sus dependencias `classpath com.google.gms:google-services:4.2.0`. La versión del service puede cambiar entre versiones de firebase.
- Editar el archivo build.gradle que se encuentra a nivel de /app y agregar `apply plugin: 'com.google.gms.google-services'` al final del archivo.
- En el mismo archivo build.gradle, agregar las dependencias de firebase tanto `com.google.firebase:firebase-iid:17.0.0` como `com.google.firebase:firebase-messaging:17.0.0`
- Agregar en Manifest los metadatos de configuración para las notificaciones entrantes, como se muestra en la siguiente figura.

```
<!-- [START fcm_default_icon] -->
<!-- Set custom default icon. This is used when no icon is set for incoming notification messages.
     See README(https://goo.gl/l4GJaQ) for more. -->
<meta-data
    android:name="com.google.firebase.messaging.default_notification_icon"
    android:resource="@drawable/ic_stat_ic_notification" />
<!-- Set color used with incoming notification messages. This is used when no color is set for the incoming
     notification message. See README(https://goo.gl/6BKBk7) for more. -->
<meta-data
    android:name="com.google.firebase.messaging.default_notification_color"
    android:resource="@color/colorAccent" />
<!-- [END fcm_default_icon] -->
<!-- [START fcm_default_channel] -->
<meta-data
    android:name="com.google.firebase.messaging.default_notification_channel_id"
    android:value="@string/default_notification_channel_id" />
<!-- [END fcm_default_channel] -->
```

Fig. 38 Configuración del archivo Android Manifest para poder usar Firebase

- Además se debe configurar el servicio que va a realizar la escucha. Este services está dentro del tag **<application>**:

```
<!-- [START firebase_service] -->
<service
    android:name=".java.MyFirebaseMessagingService"
    android:exported="false">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>
</service>
<!-- [END firebase_service] -->
```

Fig. 39 Configuración del Services de Firebase

11. BIBLIOGRAFÍA

[1] desarrollador-android.com, «https://desarrollador-android.com/desarrollo/formacion/empezar-formacion/gestionar-el-ciclo-de-vida-de-una-actividad/iniciar-una-actividad/,» [En línea].

[2] G. Developers, «https://developer.android.com/guide/components/intents-filters?hl=es,» [En línea].

[3] I. K. MSCS, «http://www.virtualians.net/group/cs619/forum/topics/mobile-app-to-calculate-steps-using-pedometer-or-accelerometer-se,» 2017. [En línea].

[4] J. Diaz Moreno, «https://www.youtube.com/watch?v=y3CXdQBXkMc,» 2016. [En línea].

[5] A. Cernat, «https://www.alexcernat.com/call-for-submissions-for-a-special-issue-on-using-mobile-apps-and-sensors-in-surveys-in-social-science-computer-review/,» 2019. [En línea].

[6] prometec, «https://www.prometec.net/usando-el-mpu6050/,» [En línea].

[7] L. llamas, «https://www.luisllamas.es/medir-la-inclinacion-imu-arduino-filtro-complementario/,» [En línea].

[8] L. Morales, «https://www.researchgate.net/publication/326423974_Medicion_de_Angulos_de_Inclinacion_por_Medio_de_Fusion_Sensorial_Aplicando_Filtro_de_Kalman,» [En línea].

[9] developer.android.com, «https://developer.android.com/guide/topics/sensors/sensors_overview?hl=es-419#java,» [En línea].

[10] M. Montero, Desarrollo de aplicaciones para Android, Grupo Editorial RA-MA, 2012.

[11] steakrecords, «https://steakrecords.com/es/48747-android-in-what-user-case-will-onaccuracychanged-trigger-android-accelerometer.html,» [En línea].

[12] S. Gomez, «https://www.sgoliver.net/blog/tareas-en-segundo-plano-en-android-i-thread-y-asynctask/,» 2012. [En línea].

[13] academiaandroid, «http://academiaandroid.com/hilo-principal-jerarquia-procesos-android/,» [En línea].

[14] androidcurso.com, «http://www.androidcurso.com/index.php/tutoriales-android/37-unidad-6-multimedia-y-ciclo-de-vida/158-ciclo-de-vida-de-una-actividad,» [En línea].

[15] developer.android.com, «https://developer.android.com/guide/components/bound-services?hl=es-419#java,» [En línea].