

Desarrollo de Aplicaciones



En Android

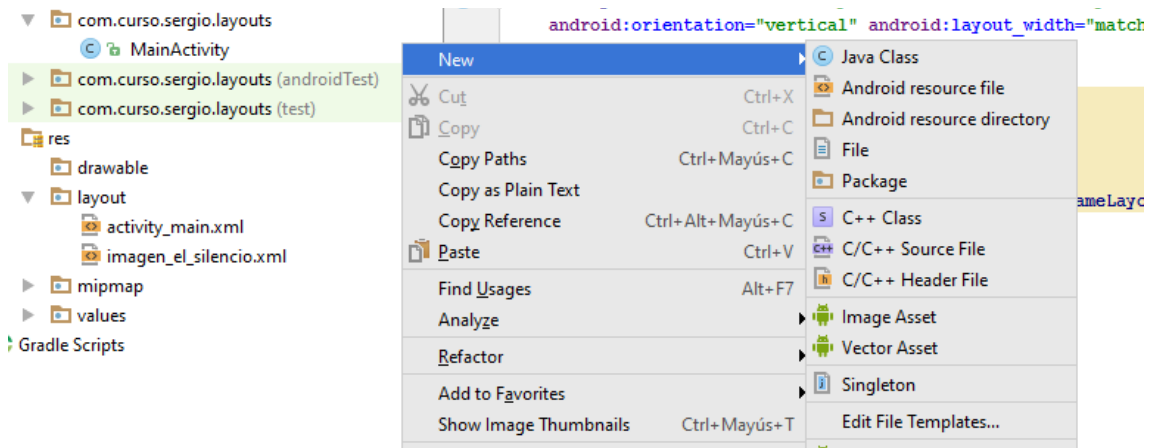
Contenido

Layouts	3
Parámetros comunes de Layouts y Views	4
FrameLayout	5
LinearLayout	5
RelativeLayout	6
Clases de interfaz de usuario (UI)	8
Button	8
ToggleButton, Switch y CheckBox	9
RatingBar	10
ViewGroups	11
Algunos ejemplos de ViewGroups	11
Mapas de Google	12
Multi-tarea. Hilos de ejecución	17
Threads	17
UI Thread	19
La classe AsyncTask	20
Broadcast Receivers	21
Registro estático	21
Registro dinámico	23
Sensores	24
Anexo: Java para Android	26

Layouts

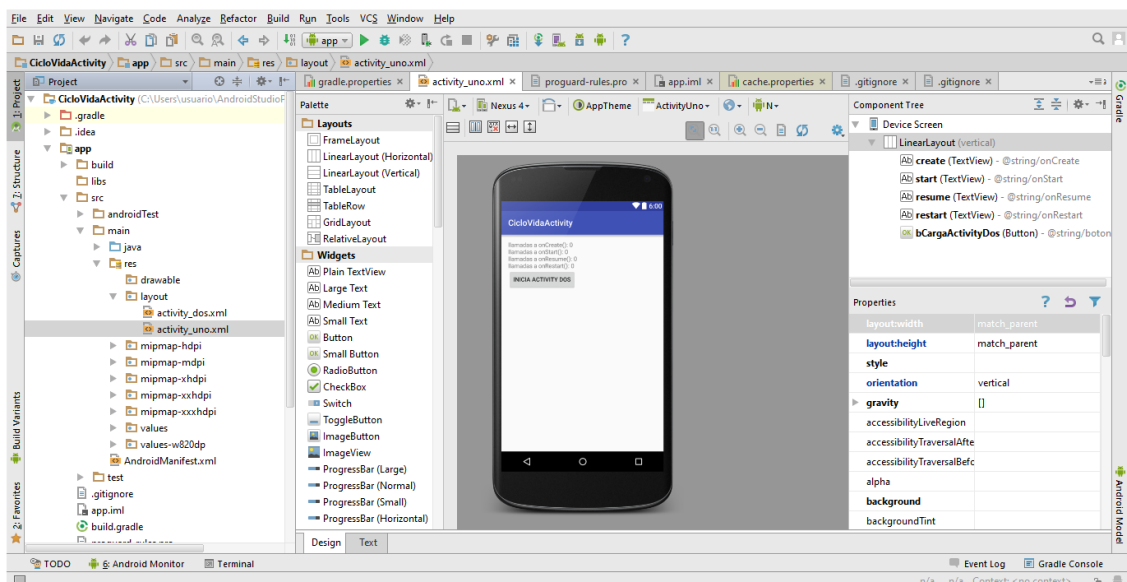
Los Layouts son un tipo específico de elementos visuales que presentamos en cada Activity como elemento distribuidor de todos los demás elementos visuales. Para crear los layouts con los que estructurar la Activity, pulsaremos el botón derecho dentro de la ventana Project.

Seleccionamos la opción **New > Android Resource File** y seleccionamos el tipo Layout dentro del diálogo abierto.



Así nos genera el nuevo Layout dentro de la carpeta adecuada **app\src\main\res\layout**.

Una vez creado lo podemos modificar accediendo al mismo desde la ruta anterior con la vista de diseño o con la vista XML.



A continuación veremos algunos parámetros de dichos layouts que nos permiten configurar su apariencia.

Parámetros comunes de Layouts y Views

Todos los elementos gráficos que pongamos de la interfaz de usuario (UI) tendrán dos parámetros obligatorios:

- `android:layout_height`
- `android:layout_width`

Estos dos parámetros nos definen, respectivamente, la altura y anchura de cada elemento.

Existen diversas posibilidades para estos parámetros. Pondremos siempre como ejemplo la etiqueta de `android:layout_width`, pero pueden aplicarse del mismo modo a `android:layout_height`

- **`android:id="@+id/nombre_elemento"`** – Definimos el nombre que asignamos al correspondiente elemento del Layout
- **`android:layout_height="match_parent"`** – El alto del elemento ocupa todo el alto del elemento padre que lo contiene
- **`android:layout_height="wrap_content"`** – El elemento tiene el alto justo para su contenido
- **`android:layout_height="200dp"`** – Definimos directamente la altura del elemento. En el ejemplo mostrado, la fijamos a 200 dp (density-independent pixel units). Los distintos tamaños implementables en Android son:

1. **dp** (Density-independent Pixels)
Reescala los elementos en concordancia con el tamaño de la pantalla en cuestión.
2. **sp** (Scale-independent Pixels)
Reescala en función del tamaño de pantalla y de las opciones de texto seleccionadas. Recomendado para texto.
3. **pt** (Points)
1/72 de una pulgada
4. **px** (Pixels)
Corresponde a píxeles reales de la pantalla. No recomendable debido al tamaño variable de pantallas.
5. **mm** (Millimeters)
Tamaño del elemento en milímetros.
6. **in** (Inches)
Tamaño del elemento en pulgadas. 1 pulgada = 2,54 centímetros.

Para obtener una referencia más completa consultad esta dirección:

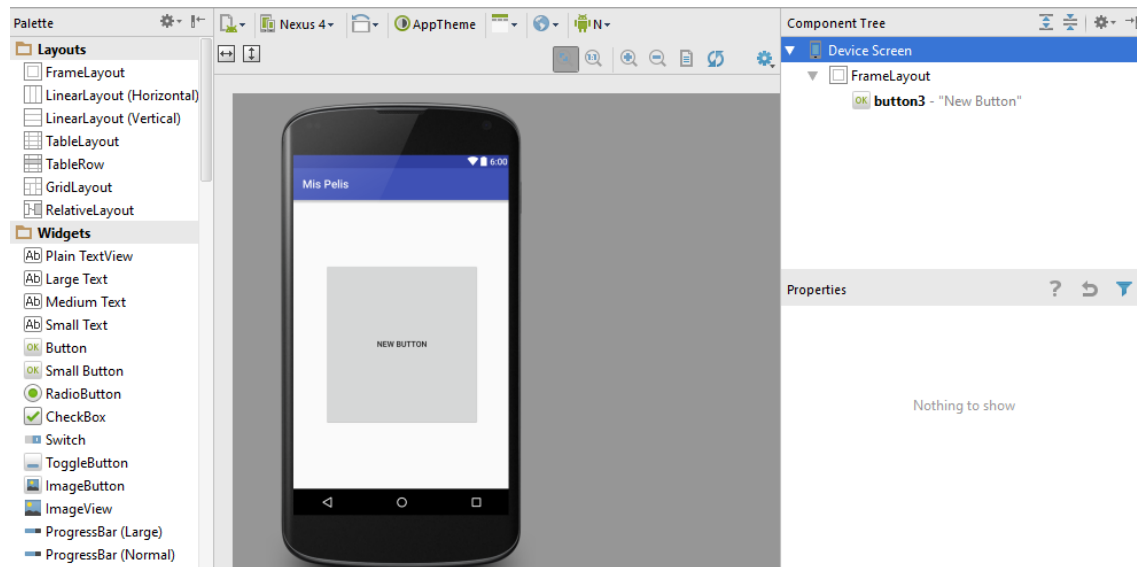
<https://developer.android.com/guide/topics/resources/more-resources.html#Dimension>

Vistos los elementos comunes de todo Layout vamos a ver algunos de los más sencillos que podemos utilizar.

Empezaremos por el **FrameLayout**, el **LinearLayout** y el **RelativeLayout**.

FrameLayout

Este tipo de Layout sirve para mostrar un único elemento en pantalla, sea un botón, una imagen, texto... También permite mostrar más de un elemento en pantalla superponiéndolos.



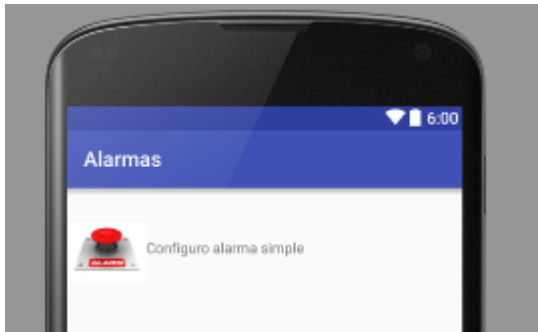
LinearLayout

En este layout colocamos los elementos que los componen de manera lineal y en el mismo orden en que se detallan en el xml. Podemos distribuirlos de manera horizontal o vertical con el parámetro

android:orientation="vertical"

o **android:orientation="horizontal"**

Puede ser útil para distribuir los componentes del layout en todo el ancho o el alto del mismo según las proporciones que le asignemos.



Por ejemplo, si quiero poner una imagen y un Textview distribuidos horizontalmente ocupando todo el ancho de pantalla y que la imagen ocupe una quinta parte de la misma y el texto 4/5 partes lo hago del siguiente modo, tal como vemos en la imagen, lo puede hacer así:

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

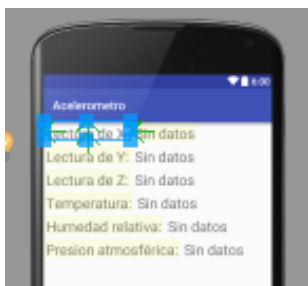
    <ImageView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:id="@+id/imageView"
        android:src="@drawable/alarma"
        android:layout_weight="1"/>

    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Configuro alarma simple"
        android:id="@+id/textView"
        android:layout_weight="4"
        android:layout_marginTop="50dp" />
</LinearLayout>
```

He definido el `android:layout_width="0dp"` de ambos elementos para que me los distribuya según los pesos que le voy a asignar con la propiedad `android:layout_weight="1"`. Al asignarle peso 1 al ImageView y peso 4 al TextView, el primero ocupa 1/5 del ancho y segundo 4/5.

RelativeLayout

Muy útil para distribuir los elementos en posiciones relativos a otros del mismo Layout. Añadido el elemento o elementos de referencia, podemos ir colocando el resto indicando que irán debajo del primero, a su derecha, a su izquierda...



En la pantalla mostrada podemos diversos TextViews de una aplicación que muestra lecturas de sensores del sistema. A la izquierda se indica con un TextView lo que estamos midiendo y a la derecha de cada uno tenemos otros TextView que pone "Sin datos" hasta que se realice la lectura. Con el uso del Relative Layout podemos alinear todos los TextViews adecuadamente.

Copiamos un extracto de la capa de texto de la aplicación mencionada:

```
<TextView
    android:id="@+id/valor_presion"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/valor_humedad"
    android:padding="5dp"
    android:text="Presion atmosférica:"
    android:textSize="24sp" />

<TextView
    android:id="@+id/valor_real_x"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_toRightOf="@+id/valor_x"
    android:text="Sin datos"
    android:padding="5dp"
    android:textSize="24sp" />
```

Como ejemplo hemos cogido un TextView que colocará a la izquierda del todo y debajo de otro. Definimos estas propiedades con las siguientes líneas

```
android:layout_alignParentLeft="true"
android:layout_below="@+id/valor_humedad"
```

En el segundo elemento hacemos algo parecido, pero en este caso estamos colocándolo arriba del todo y a la derecha de otro elemento dado

```
android:layout_alignParentTop="true"
android:layout_toRightOf="@+id/valor_x"
```

Clases de interfaz de usuario (UI)

Las clases de la interfaz de usuario son los elementos básicos que nos permiten la construcción de cualquier elemento visible en pantalla. Mediante dichos elementos el usuario y la aplicación intercambian información. Destacamos los siguientes tipos de clases:

- **View y View Events.** Los Views son los bloques básicos con los que añadir los componentes de la interfaz de usuario. Ocupan un espacio rectangular en pantalla y se encargan de dibujarse ellos mismos. Los View Events son los eventos producidos en tiempo de ejecución a los que los Views deben responder.
- **View Groups.** Son estructuras no visibles de tipo View capaces de contener, agrupar y organizar un conjunto de Views. Entre los View Groups más destacados hallamos los Layouts.
- **Menús y ActionBar.** Permiten acceso rápido y destacado a las acciones más frecuentes o destacadas para el usuario.
- **Dialogs.** Views emergentes que se muestran superpuestos a la aplicación en ejecución y que por lo general nos dan información de la aplicación o de su estado de ejecución o permiten recibirla del usuario.

Iniciaremos una revisión de los parámetros comunes entre los distintos tipos de Views.

Para utilizar cualquier View lo primero que debemos hacer es crear una referencia al objeto en cuestión del siguiente modo:

```
TipoView nombre_variable = (TipoView) findViewById(R.id.nombre_elemento_en_layout);
```

Después ya podremos utilizar los métodos específicos de cada View (Listeners, Adapters, etc) para trabajar con ellos.

Repasamos a continuación una colección de los Views más simples a utilizar, empezando por el Button.

Button

Corresponde al clásico botón a pulsar para realizar una acción concreta. Creamos la referencia al mismo con la siguiente sentencia:

```
Button nombre_boton = (Button) findViewById(R.id.nombre_boton_en_layout);
```

Usaremos el método `setOnClickListener` y el método `onClick` para determinar la acción a realizar al pulsarlo.


```

nombre_boton.setOnClickListener(new View.OnClickListener() {
    // el método onClick nos dice la acción a realizar cuando hay un clic

    @Override
    public void onClick(View v) {

        // Acción realizada cuando se genera el evento de pulsar el botón

    }
});

```

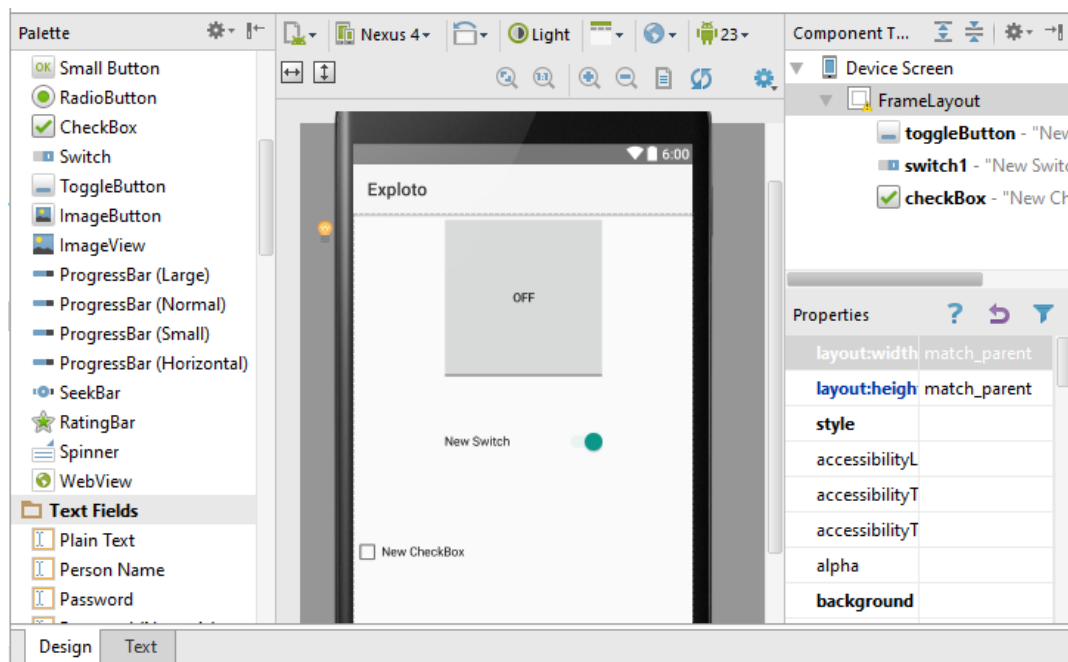
Si en lugar de poner texto en el botón queremos poner una imagen, debemos usar el tipo ImageButton del mismo modo, pero definiendo la propiedad

android:src="@drawable/nombre_imagen"



ToggleButton, Switch y CheckBox

Corresponden a botones con dos estados posibles (On/Off, por ejemplo).



Creamos la referencia a ellos del siguiente modo:

```

ToggleButton nombre_boton = (ToggleButton) findViewById(R.id.nombre_boton_en_layout);
CheckBox nombre_checkbox = (CheckBox) findViewById(R.id.nombre_checkbox_en_layout);
Switch nombre_switch = (Switch) findViewById(R.id.nombre_switch_en_layout);

```

Podemos definir el estado inicial del checkbox y el switch con el parámetro checked:

android:checked="false"

El `ToggleButton` tiene dos textos por defecto a mostrar, el `textoOn` y el `textoOff`.

```
android:textOn="textoOn"
```

```
android:textOff="textoOff"
```

Usaremos también los métodos `setOnClickListener` y `onClick` para efectuar las acciones deseadas, pero en este caso controlando si el estado está verificado o no:

```
nombre_boton.setOnClickListener(new View.OnClickListener() {  
    // el método onClick nos dice la acción a realizar cuando hay un clic  
  
    @Override  
    public void onClick(View v) {  
  
        // Acción realizada cuando se genera el evento de pulsar el botón  
        if(nombre_boton.isChecked()){  
  
            //Acción a realizar cuando el estado es verificado  
  
        }else{  
  
            //Acción a realizar cuando el estado es NO verificado  
  
        }  
    }  
});
```

RatingBar

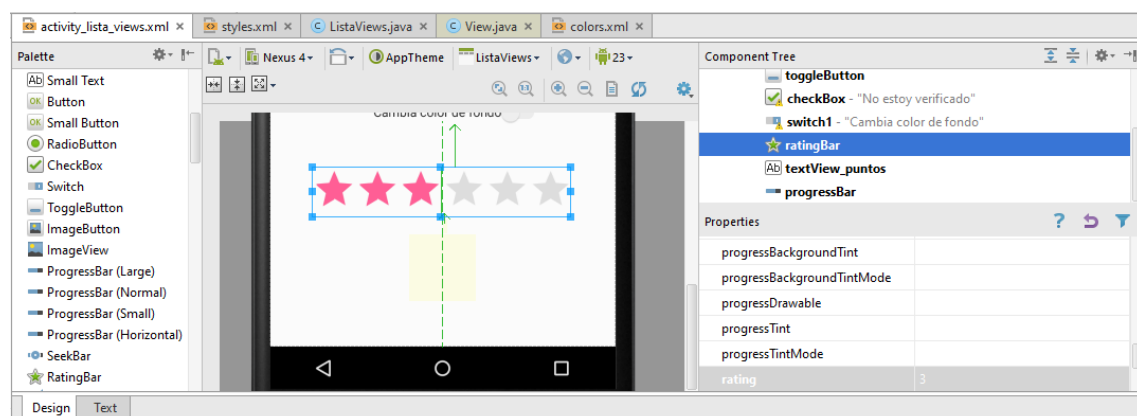
Corresponde a una barra para puntuar con estrellas. Para crear la referencia hacemos:

```
RatingBar nombre_rating_bar = (RatingBar) findViewById(R.id.nombre_bar_en_layout);
```

Sus parámetros específicos indican el número de estrellas para calificar y el paso mínimo de evaluación.

```
android:numStars="5"
```

```
android:stepSize="0.5"
```



El Listener de la RatingBar es un poco más complejo, controlando ahora cambios de estado de la votación.

```
puntuacion.setOnRatingBarChangeListener(new
RatingBar.OnRatingBarChangeListener() {

    // Llamado cuando el usuario cambiar la puntuación
    @Override
    public void onRatingChanged(RatingBar ratingBar, float rating,
boolean fromUser) {
        puntos.setText("La puntuación otorgada es:" + rating);
    }
});
```

ViewGroups

Los ViewGroups son Views un poco más sofisticados, que permiten agrupar diversos elementos o que tienen funcionalidades más avanzadas.

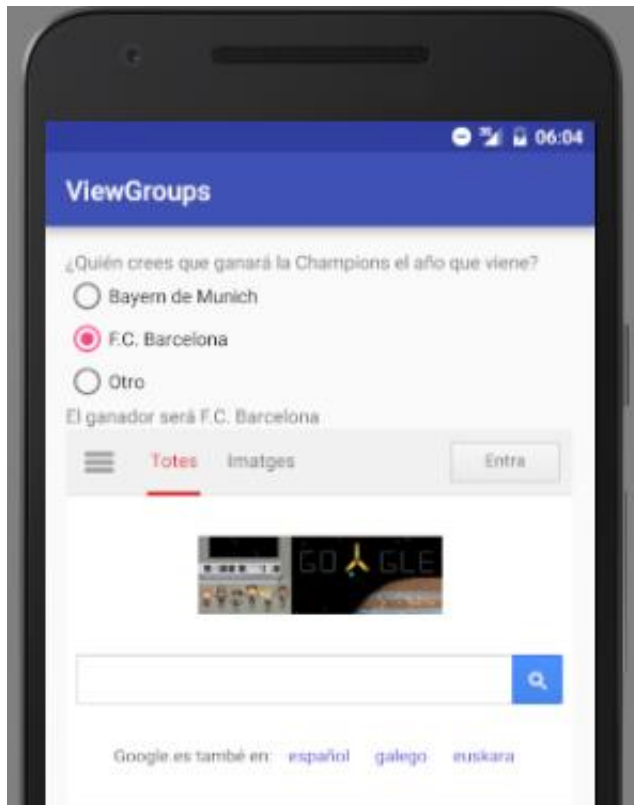
Algunos ejemplos de ViewGroups

WebView es un ViewGroup que nos permite poner un navegador dentro de la pantalla de nuestra aplicación, por ejemplo.

Los **RadioGroups** son elementos que agrupan un conjunto de RadioButtons para escoger entre determinadas opciones. En la siguiente aplicación, **ViewGroups**, que podemos encontrar en

<https://github.com/sergiofbertolin/CursoAndroid>

Tenemos implementado y explicado el uso de dichos elementos.



Tal y como nos muestra la pantalla, aquí podemos seleccionar una única respuesta a una pregunta determinada con el uso de un RadioGroup.

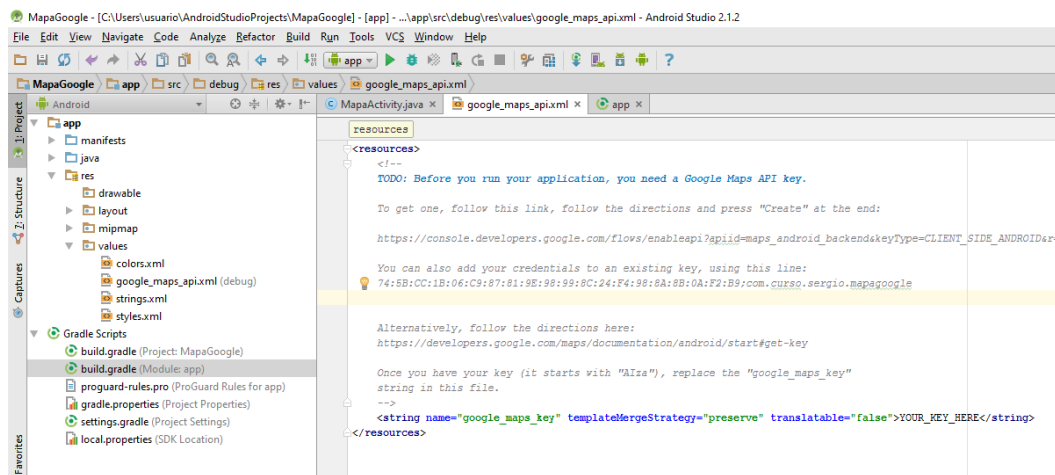
Debajo del anterior tenemos un WebView que nos permite la navegación dentro de la misma interfaz de la aplicación, sin necesidad de dejar de ver el RadioGroup anterior.

Otros ViewGroups usuales son los **ListViews** o **Spinners**, que nos permiten implementar listas desplegables o añadir imágenes a cada elemento de la lista. Para ver una referencia completa de ellos, acceder a los proyectos de Github ListView y Spinner, respectivamente.

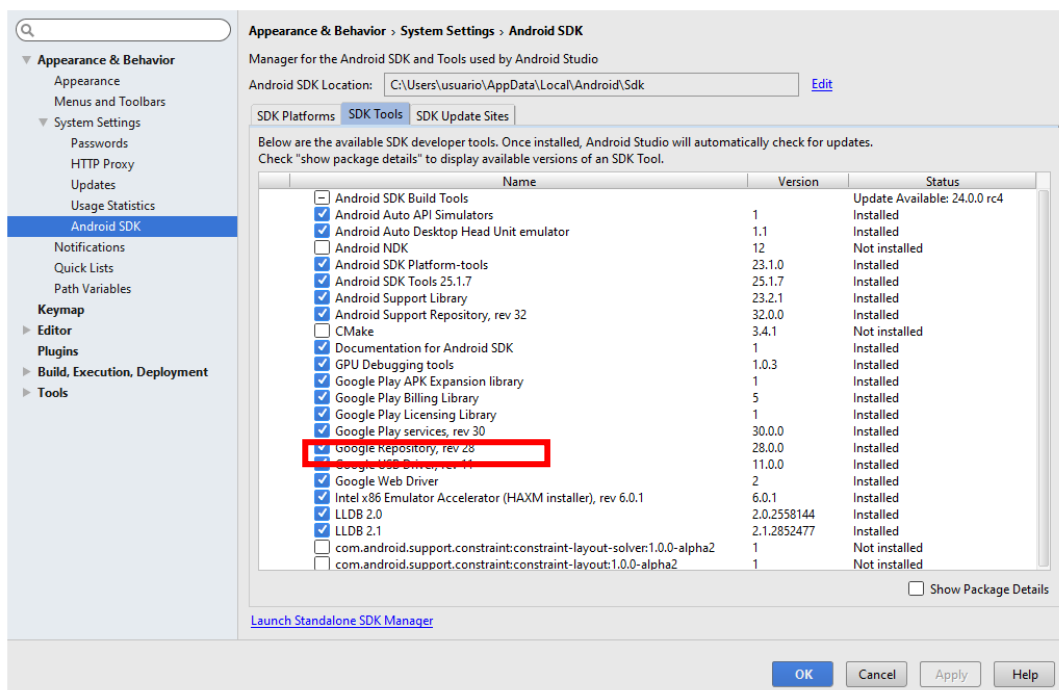
Mapas de Google

Android facilita mucho el uso e implementación de Mapas de Google dentro de sus aplicaciones, pero es necesario registrarnos como desarrolladores en la API de Mapas de Google para Android. Detallamos el procedimiento a seguir.

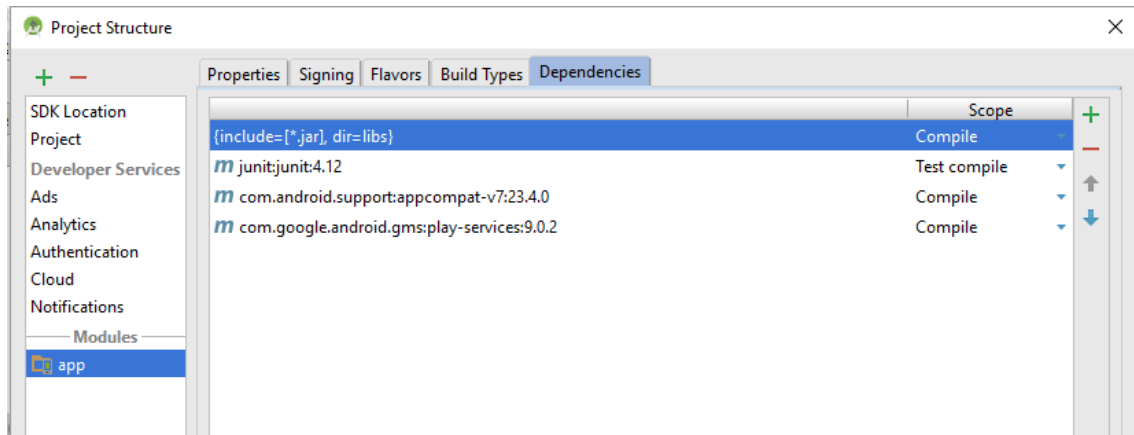
1. Creo un nuevo proyecto o Activity del tipo **GoogleMapsActivity**. Android Studio me genera automáticamente una Activity y un fichero llamado **google_maps_api.xml** que nos ayudará más adelante a configurar nuestro uso de Google Maps.



2. Asegúrate en **Tools>Android>SDK Manager** de tener instalados los Google Play Services entre las SDK Tools



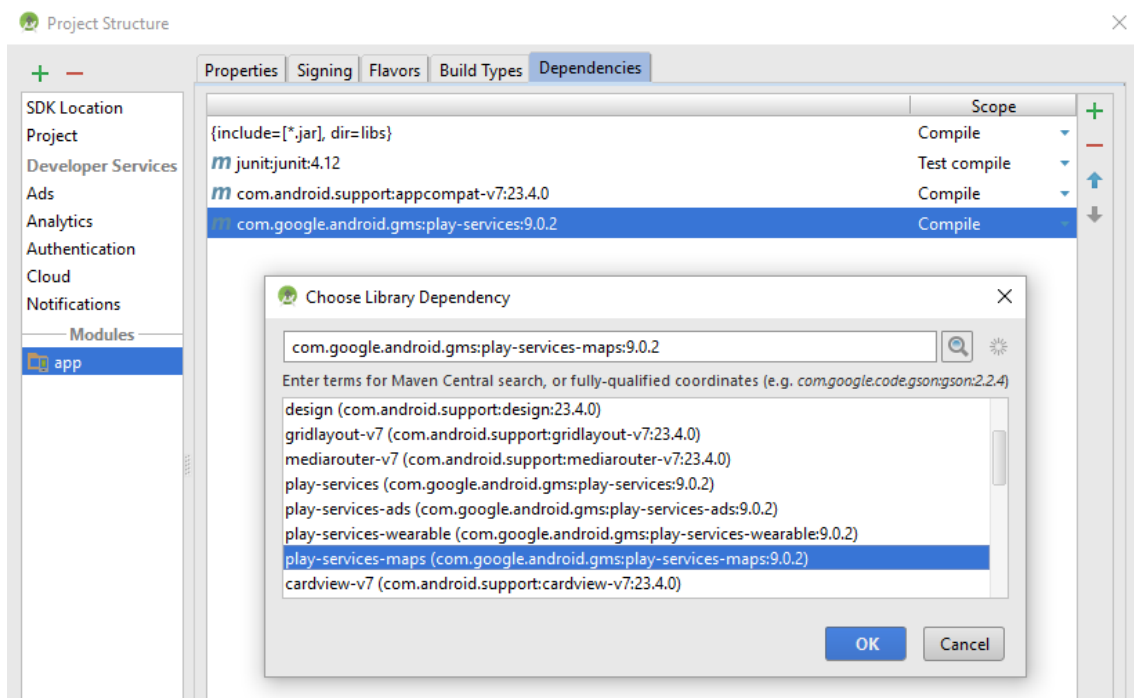
3. Añade una dependencia a build.gradle. Para ello, entra en **File>Project Structure** y selecciona la pestaña **Dependencies**



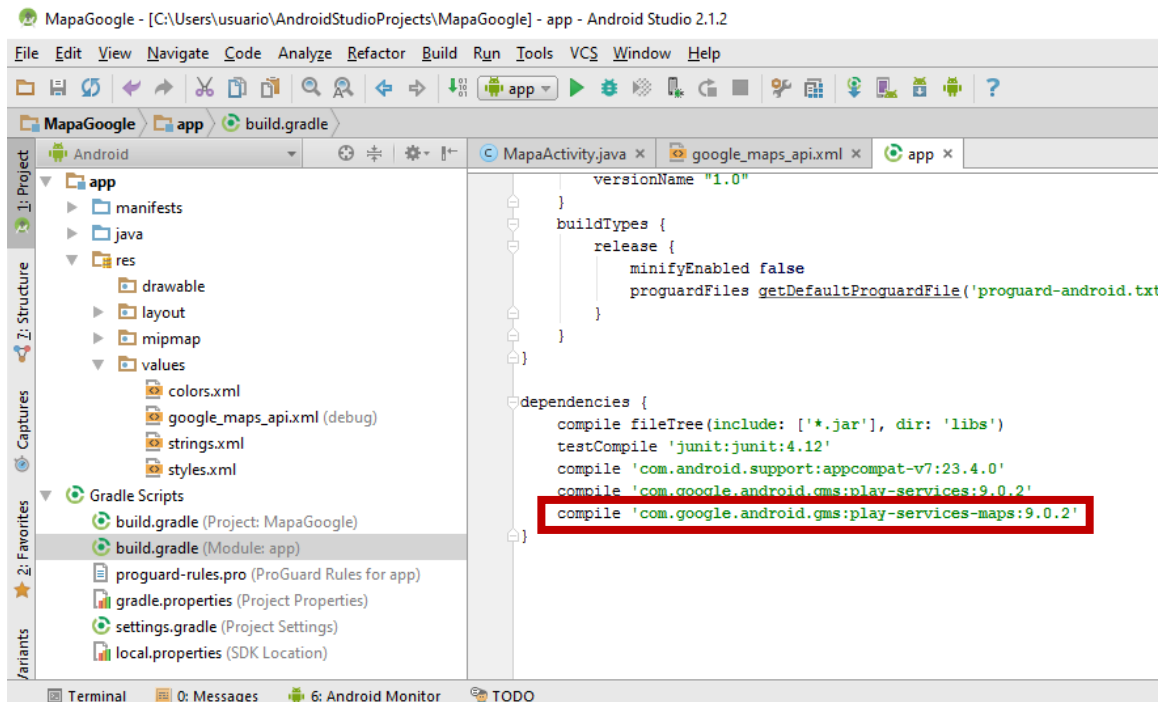
Deberás tener una dependency que ponga:

com-google.android.gms:play-services:(versión)

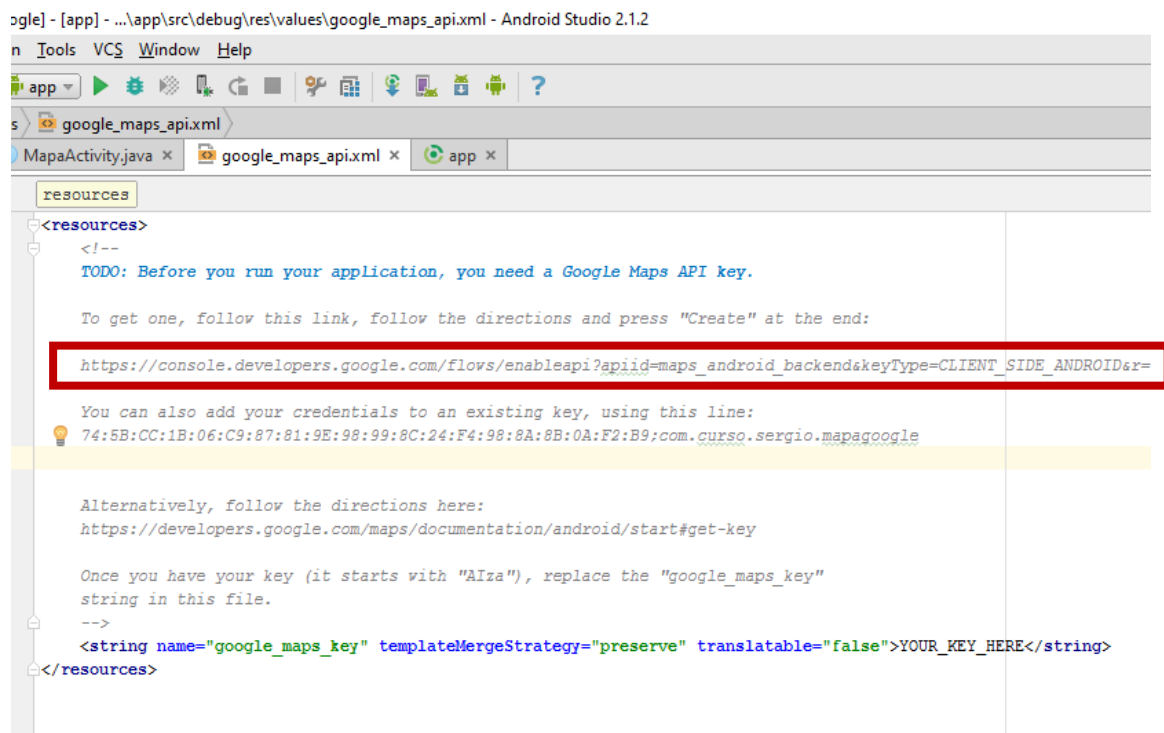
En mi caso no la tengo, por lo que le doy al botón **+** verde que hay a la derecha de la pantalla y añado una **Library Dependency**



Un modo alternativo de hacerlo es editar manualmente el archivo **build.gradle**, agregando la siguiente línea marcada en rojo:



Ahora creamos nuestro proyecto en la API de GoogleMaps usando el link que nos proporciona el propio AndroidStudio



Acto seguido, tras habernos identificado como usuarios de Google, nos aparecerá una pantalla como esta:

 ¿Te gustan nuestras API? Descubre nuestra infraestructura. Regístrate y consigue un crédito. [Más información](#)

 **Google APIs** 

Registrar la aplicación para Google Maps Android API en Google Developers Console

Google Developers Console te permite gestionar tu aplicación y supervisar el uso de la API.

No tienes ningún proyecto. Se creará un proyecto nuevo llamado "My Project".

Quiero recibir por correo electrónico información sobre las funciones del producto, sugerencias de rendimiento, encuestas para aportar mis observaciones y ofertas especiales.



☐ Sí ☐ No

Acepto que el uso que haga de cualquier [servicio y API relacionadas](#) queda sujeto a mi cumplimiento de las [Condiciones de Servicio](#) correspondientes.

☐ Sí ☒ No

[Aceptar y continuar](#)

Después de pulsar el botón continuar tendremos la siguiente pantalla:

 **Google APIs** 

La API se ha habilitado

El proyecto se ha creado y Google Maps Android API se ha habilitado.

A continuación, para usar la API necesitas las credenciales correctas.

[Ir a las credenciales](#)

Accedemos a las credenciales, donde ya tendremos el fingerprint SHA1 y el nombre del package.

API Administrador de API: Credenciales

Visión general
Credenciales

←

Crear clave de API de Android

Nombre
Clave de Android 1

Restringir el uso a tus aplicaciones Android (Opcional)
Añade el nombre del paquete y la huella digital del certificado de firma SHA-1 para restringir el uso de tus aplicaciones de Android. [Más información](#)
Puedes encontrar el nombre del paquete en el archivo AndroidManifest.xml. A continuación, usa el comando siguiente para obtener la huella digital:

```
$ keytool -list -v -keystore mystore.keystore
```

Nombre de paquete	Huella digital de certificado SHA-1
com.curso.sergio.mapagoogole	74:5B:CC:1B:06:C9:87:81:9E:98:99:8C:24:F4:98:8A:8B:0A:F2:B9

+ Añadir nombre de paquete y huella digital

Nota: Pueden pasar hasta 5 minutos antes de que se aplique la configuración

Crear Cancelar

Presionamos crear y ya tenemos la Key necesaria para nuestro proyecto, que empieza por AIZA. La copiamos en el documento xml, donde poner YOUR_KEY_HERE.

```
ogle] - [app] - ...app\src\debug\res\values\google_maps_api.xml - Android Studio 2.1.2
n Tools VCS Window Help
s google_maps_api.xml
MapaActivity.java x google_maps_api.xml x app x
resources
<resources>
  <!--
  TODO: Before you run your application, you need a Google Maps API key.

  To get one, follow this link, follow the directions and press "Create" at the end:

  https://console.developers.google.com/flows/enableapi?apiid=maps_android_backend&keyType=CLIENT_SIDE_ANDROID&r=

  You can also add your credentials to an existing key, using this line:
  74:5B:CC:1B:06:C9:87:81:9E:98:99:8C:24:F4:98:8A:8B:0A:F2:B9;com.curso.sergio.mapagoogole

  Alternatively, follow the directions here:
  https://developers.google.com/maps/documentation/android/start#get-key

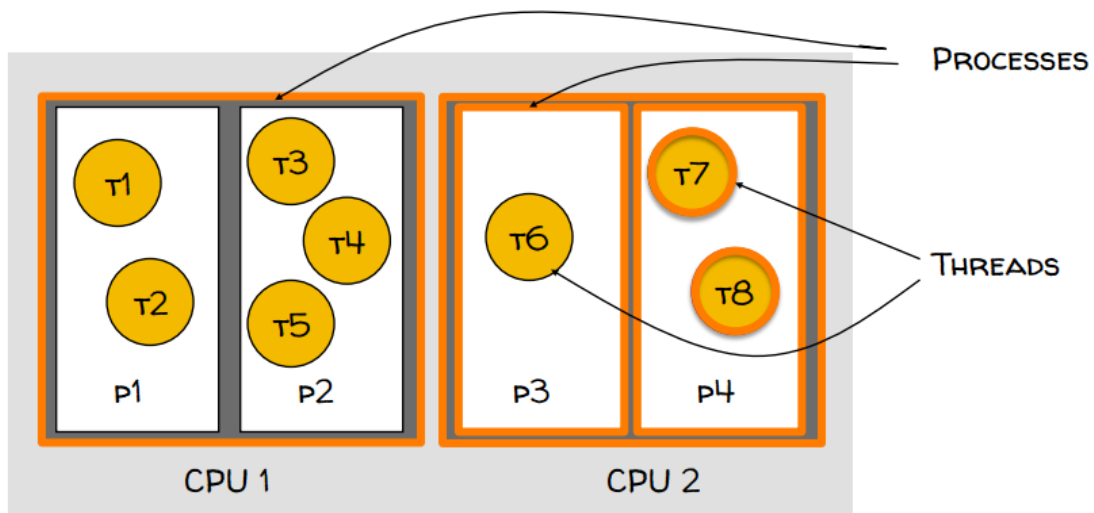
  Once you have your key (it starts with "AIza"), replace the "google_maps_key"
  string in this file.
  -->
  <string name="google_maps_key" templateMergeStrategy="preserve" translatable="false">YOUR_KEY_HERE</string>
</resources>
```

¡Ya podemos usar GoogleMaps en nuestro proyecto! Para acceder a una variedad completa de configuraciones de mapas de Google, consultar el proyecto MapaGoogle en github:

<https://github.com/sergiofbertolin/CursoAndroid/tree/master/MapaGoogle>

Multi-tarea. Hilos de ejecución

En el panorama actual, donde los dispositivos cuentan con varios procesadores, parece imposible no habilitar el uso de aplicaciones que soporten tareas ejecutadas de forma concurrente o en paralelo. De hecho, debemos poder distinguir entre ejecuciones en diversos procesadores, en distintos procesos o en diversos hilos de ejecución o Threads.



Cada proceso corresponde a un entorno de ejecución auto-contenido. Esto quiere decir que dentro del mismo proceso podemos ejecutar diversos Threads o hilos de ejecución que compartirán recursos, memoria y abrirán y accederán al mismo archivo. Este uso compartido de recursos no se produce de manera natural entre distintos procesos.

Threads

Dentro de cada proceso podemos alojar más de un hilo de ejecución o Thread. Estos son un conjunto estructurado de instrucciones ejecutadas de manera secuencial por su proceso y que cuentan con una pila propia de recursos, aun compartiendo ciertos recursos con todo el proceso que los contiene.

Para acceder a un sencillo tutorial sobre Threads en Java podemos consultar:

<http://docs.oracle.com/javase/tutorial/essential/concurrency/threads.html>

Para nuestro uso más básico debemos conocer dos formas de implementar threads en Java:

- Proveer un objeto de tipo Runnable. La interfaz Runnable define el método run, que debe contener el código ejecutado en el hilo. El objeto Runnable se pasa al Threadconstructor, como en el siguiente ejemplo HelloWorldRunnable:

```

public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}

```

- Subclase Thread. La propia clase Thread implementa un Runnable, aunque su método run no hace nada por sí solo. Una aplicación puede crear una subclase de tipo Thread y usar su propia implementación del método run, como en el ejemplo HelloThread:

```

public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a thread!");
    }

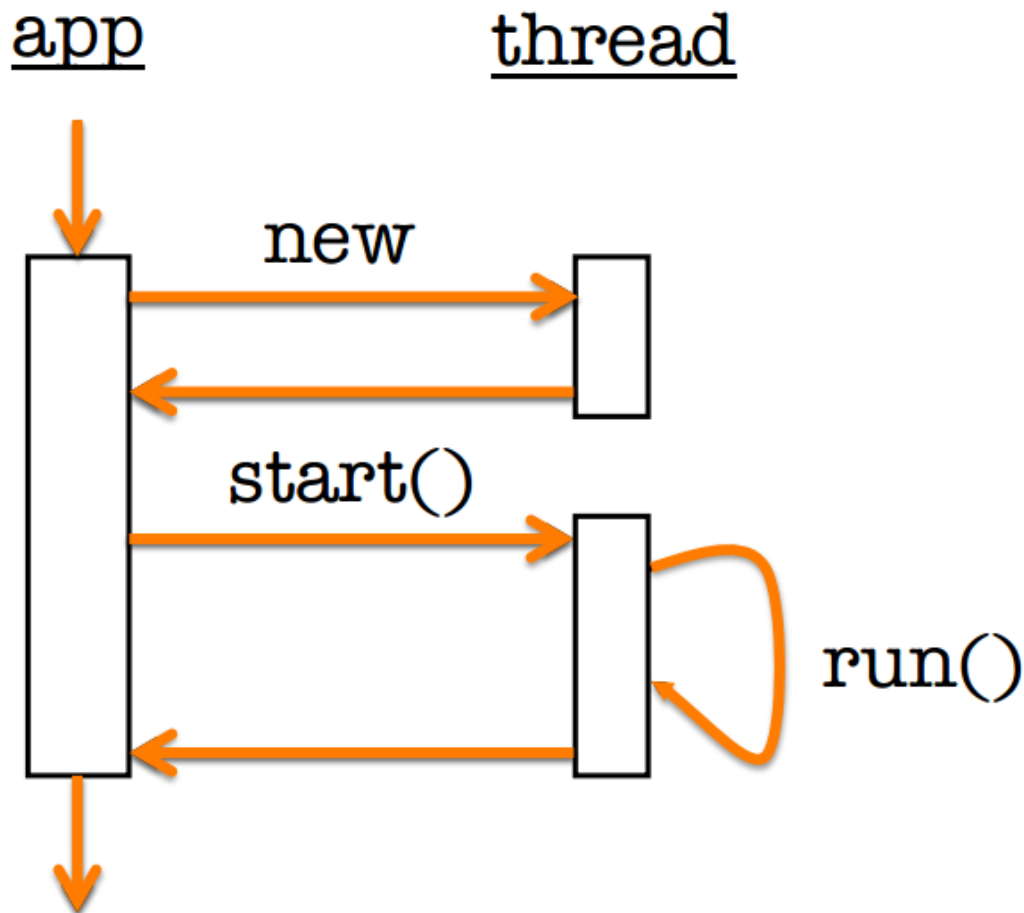
    public static void main(String args[]) {
        (new HelloThread()).start();
    }

}

```

En ambos casos deberemos usar un Thread.start() para iniciar el hilo. La clase Runnable puede ser creada desde cualquier clase, mientras que la Thread sólo puede ser usada desde objetos de tipo Thread. Algunos métodos útiles de la clase Thread son:

- **void start()** : Método llamado para iniciar el método run del Thread o del Runnable
- **void sleep(long time)** : Método llamado para permanecer parado por un periodo específico de tiempo
- **void wait()** y **void notify()** : Con este método paramos la ejecución del Thread hasta que otro Thread llame al notify del Objeto a despertar
- **void start()** y **void run()** : Llamamos al start para iniciar un Thread. Tras esta llamada se ejecuta el método run, aunque no inmediatamente después, quedando los dos hilos de ejecución en paralelo hasta que acabe el método run.



Ahora sería un buen momento de probar la aplicación *ConcurrenciaSimple*.
¿Por qué falla?

UI Thread

Android no permite que modifiquemos Views que pertenecen a un thread distinto al nuestro. En el UI Thread ejecutamos los Views de todos los componentes de la aplicación. Así evitamos que la aplicación se caiga por modificar Views desde Thread distintos. El problema que surge es que el uso de las herramientas del UI Thread no son seguras del todo y no son accesibles desde fuera del UI Thread.

- Debemos realizar las operaciones más laboriosas fuera del UI Thread, en un hilo en background (segundo plano).
- Del mismo modo todas las operaciones que trabajen con las herramientas del UI deben realizarse en el UI Thread, procurando restringir dichas operaciones a las mínimas posibles.
- Todas las interacciones con el usuario, los callbacks del sistema y los métodos de ciclo de vida se ejecutan en el UI Thread

Disponemos de diversos métodos en Android que nos garantizan la ejecución en el UI Thread.

- boolean View.post (Runnable action)
- void Activity.runOnUiThread (Runnable action)

Como mejora de la aplicación *ConcurrenciaSimple* implementamos la aplicación *ConcurrenciaViewPost*, que soluciona el problema anterior.

La clase AsyncTask

Android provee una clase, AsyncTask, específicamente diseñada para solucionar estas ejecuciones en hilos en background o en el UI Thread. Deberemos asignar los procesos que se ejecuten a largo plazo e indicaciones de progreso en segundo plano. Las ejecuciones en el UI Thread las reservamos para configurar las operaciones costosas, indicar progreso a medio plazo y completar las operaciones en la UI cuando han finalizado las operaciones en background más costosas en tiempo. La estructura de dicha clase es:

```
AsyncTask<Params,Progress,Result>{
...
}
```

Donde Params es el tipo de variable o variables necesarias para los procesos en background, Progress para los procesos responsables de indicar progreso y Result para mostrar los resultados de las ejecuciones en segundo plano.

El flujo de trabajo del AsyncTask es el siguiente:

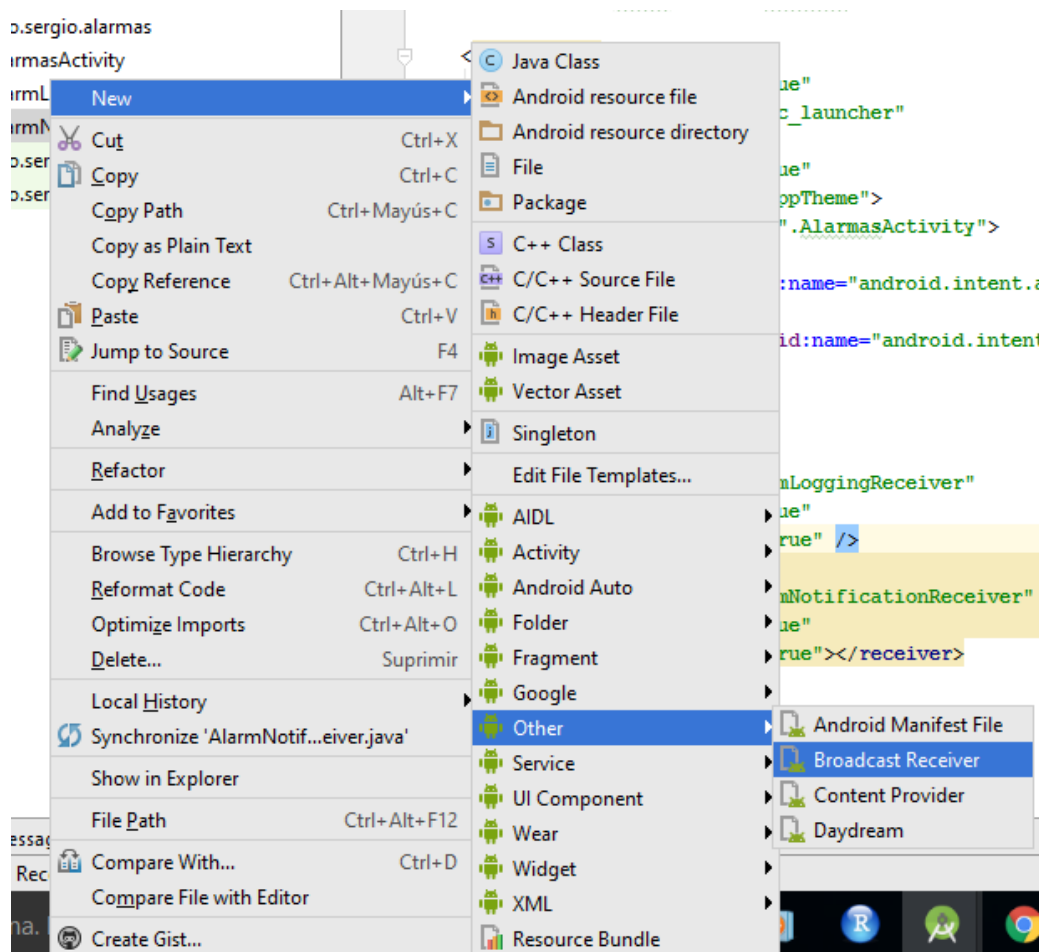
1. **onPreExecute()**. Método ejecutado en el UI Thread. Lo usamos para realizar las configuraciones previas de las tareas a ejecutar en background.
2. **doInBackground(Params... params)**. Se ejecuta en un Thread en background. Retorna siempre un resultado del tipo Result especificado en la definición del AsyncTask. Puede llamar al método **publishProgress(Progress... values)** para actualizar los contadores de progreso.
3. **onProgressUpdate(Progress... values)**. Ejecutado si publishProgress es llamado para actualizar el contador de progreso en el UI Thread.
4. **onPostExecute(Result result)**. Ejecutamos en el UI Thread el tratamiento adecuado tras la ejecución del **doInBackground**.

Broadcast Receivers

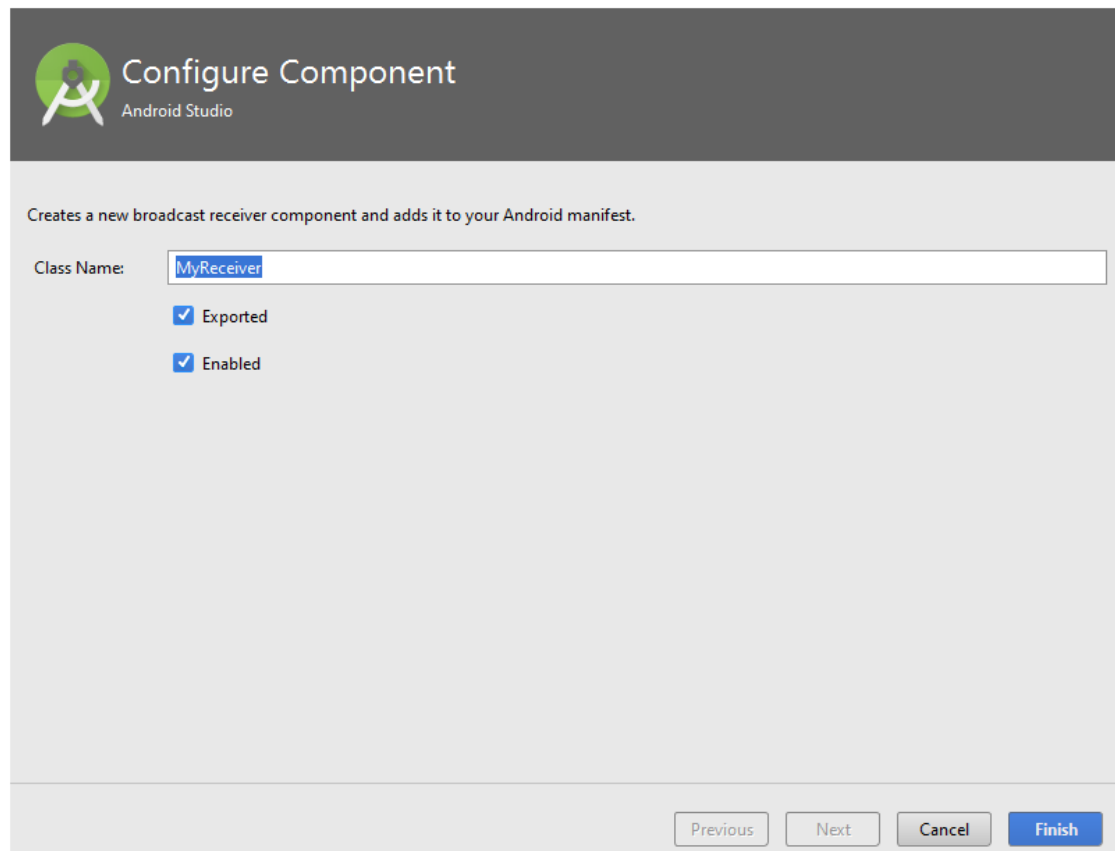
Es una clase que se encarga de esperar a que unos determinados eventos sucedan en el sistema. Entonces ejecutan la respuesta implementada. La primera acción a realizar, pues, es registrar en el sistema qué eventos espera cada BroadcastReceiver (Por ejemplo, esperar recepción de SMS o de llamadas). Los eventos son representados como Intents, que serán enviados al sistema. Podemos registrar los Receivers de dos maneras, estáticamente en el AndroidManifest o dinámicamente en tiempo de ejecución.

Registro estático

En este caso se registran los BroadcastReceivers durante el arranque del dispositivo o durante la instalación del package. Para crearlos estáticamente desde Android Studio los añadimos con la opción **New>Other>BroadcastReceiver**.



Inmediatamente se nos pedirá el nombre del BroadcastReceiver y si queremos activas las opciones **enabled** y **exported**. Con la opción enabled el Receiver queda habilitado desde el principio y con la opción exported el Receiver podrá ser ejecutado por Intents llamados desde otra aplicación.



Una vez creado podemos ver que, aparte de tener la clase creada, el Manifest ya incorpora nuestro Receiver:

```
<receiver
    android:name=".MiReceiver"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action
            android:name="com.curso.sergio.BroadcastEstatico.show_toast" >
        </action>
    </intent-filter>
</receiver>
```

Dentro del Receiver definimos los **intent filter** que especifican los intents que "dispararán" la ejecución del método **onReceive()** de nuestro BroadcastReceiver.

En el caso anterior nuestro onReceive() se ejecuta cuando recibimos intents con action de tipo `"com.curso.sergio.BroadcastEstatico.show_toast"`

Para cargar Intents de este tipo sólo nos faltará definir las permissions necesarias para que se ejecuten. En nuestro ejemplo activamos un aviso por vibración, por lo que deberemos añadir el permiso requerido para hacer vibrar el dispositivo:

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

Tenemos ya el registro estático finalizado. Ahora debemos enviar el Intent al sistema, hacerle un "Broadcast", del siguiente modo:

```
sendBroadcast(new Intent(CUSTOM_INTENT),  
    android.Manifest.permission.VIBRATE);
```

El primer parámetro es el Intent enviado (para nosotros el CUSTOM_INTENT) y el segundo corresponde a los permisos necesarios para que se ejecute.

Únicamente queda implementar lo que hará el BroadcastReceiver al recibir el Intent, escribiendo el código correspondiente en el método **onReceive()** del BroadcastReceiver

```
@Override  
public void onReceive(Context context, Intent intent) {  
    // TODO: This method is called when the BroadcastReceiver is  
    // receiving  
    // an Intent broadcast.  
  
    Código de la acción a realizar al recibir el Intent  
}
```

Para ver una lista de los Receivers estáticos ejecutados podemos usar este comando en la consola

```
adb shell dumpsys package
```

Registro dinámico

Para registrar el Receiver en tiempo de ejecución, no usaremos el AndroidManifest. Para empezar crearemos el IntentFilter en tiempo de ejecución.

```
//Creando el Intent Filter con la action correspondiente  
private final IntentFilter mi_intentFilter = new  
IntentFilter(CUSTOM_INTENT);
```

Ahora haremos lo propio con el Receiver y con una variable de tipo LocalBroadcastManager.

```
//Creando un objeto de tipo MiReceiver()  
private final MiReceiver mi_receiver = new MiReceiver();  
  
//Variable con un LocalBroadcastManager  
private LocalBroadcastManager mBroadcastMgr;
```

En el método **onCreate()** deberemos obtener el LocalBroadcastManager que corresponde al contexto de la aplicación y registrar el Intent objetivo en nuestro receiver.

```
//Obtenemos un objeto de tipo BroadcastManager con el contexto  
de mi aplicación  
mBroadcastMgr = LocalBroadcastManager  
    .getInstance(getApplicationContext());  
//Registro el Receiver dinámicamente con el Receiver e Intent  
Filter adecuados  
mBroadcastMgr.registerReceiver(mi_receiver, mi_intentFilter);
```

Con el Intent registrado hacemos un broadcast del Intent del mismo modo que antes, usando el método `sendBroadcast()`

```
mBroadcastMgr.sendBroadcast(new Intent(CUSTOM_INTENT));
```

La implementación del Receiver es exactamente igual al caso anterior. Tan sólo queda eliminar el registro del Intent al finalizar la aplicación, en el método `onDestroy()` o en el método `onPause()`.

```
@Override
protected void onDestroy() {
    //Borro el registro del Receiver
    mBroadcastMgr.unregisterReceiver(mi_receiver);
    super.onDestroy();
}
```

Para ver una lista de los Receivers registrados dinámicamente podemos usar este comando en la consola

```
adb shell dumpsys activity b
```

Sensores

No debemos olvidar que muchos de nuestros dispositivos actuales cuentan con sensores de diversos tipos que nos permiten implementar funciones avanzadas. Los sensores son componentes físicos que toman medidas del entorno físico. Tenemos tres tipos básicos de sensores según qué estamos midiendo:

- Movimiento: Acelerómetro de 3 ejes, giróscopo
- Posición: Medidor de campo magnético de 3 ejes, GPS
- Entorno: Sensor de presión, de temperatura

Para usar cualquiera de estos sensores usamos el `SensorManager`.

Debemos primero obtenerlo del sistema:

```
// Obtener referencia al SensorManager
mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
```

Después usamos el `SensorManager` para obtener la referencia al sensor concreto a utilizar.

```
// Obteniendo referencia al Acelerómetro
mAcelerometro =
mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

Nos queda registrar el Listener del sensor para permanecer a la espera de recepción de datos por parte del mismo.

```
mSensorManager.
registerListener(this, mAcelerometro, SensorManager.SENSOR_DELAY_UI);
```


Cuando cerramos la Activity que usa el sensor no debemos olvidarnos de deshacer el registro del Listener

```
// Unregister listener
@Override
protected void onPause() {
    mSensorManager.unregisterListener(this);
    super.onPause();
}
```

Los eventos recibidos por el sensor tienen las siguientes propiedades:

- **Unidades.** Cada sensor recibe los datos en unas unidades específicas. Por ejemplo, el acelerómetro marca las unidades en m/s², el sensor de humedad da un porcentaje...
- **Registro de la hora de la medida**
- **Precisión.** Cada medida se toma con una precisión máxima determinada.

Los Listeners de los sensores tienen los siguientes callbacks, asociados a cambios de precisión en la medida y a nuevas medidas recibidas, respectivamente:

```
@Override
public final void onAccuracyChanged(Sensor sensor, int accuracy) {
    //Implementa una acción con un cambio de precisión del sensor
}

@Override
public final void onSensorChanged(SensorEvent event) {
    // Implementa una acción para una lectura nueva del sensor
}
```

Para una referencia más completa y detallada al uso de sensores visita la siguiente página

https://developer.android.com/guide/topics/sensors/sensors_overview.html

Anexo: Java para Android

En Android crearemos o modificaremos el código principalmente de clases de Java. Cada clase corresponderá normalmente a uno de los cuatro componentes básicos de Android:

- Activity
- Service
- BroadcastReceiver
- ContentProvider

Cuando hacemos un **New** en **Android Studio** observamos que nos da plantillas para muchas de estas clases. Al crear una Activity, por ejemplo, nos genera un código que empieza del siguiente modo:

```
//package es el nombre del conjunto de clases que formarán mi App
package com.curso.sergio.ciclovidaactivity;

//A continuación importamos el resto de clases o packages necesarios
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

//Aquí empieza la clase realmente
public class MiClase extends Activity {

    Aquí dentro iría el código de la clase
}
```

Atributos

Las clases tienen atributos o variables. Una variable puede ser un único valor relacionado con un concepto o atributo de la clase. Las variables pueden tener un valor que no cambie nunca o modificarse durante la ejecución del programa. Por ejemplo, si creo un programa que cuente el número de letras que he escrito en un texto, puedo usar la siguiente variable:

```
int num_letras = 0;
```

Donde int es el tipo de variable y num_letras su nombre. Inicialmente le doy valor 0 a la variable porque todavía no he contado letras. Si en vez del número de letras necesito una variable para definir el número máximo de letras a contar, definiré una constante. Esto se hace de la siguiente manera:

```
static final int num_max_letras = 0;
```

Métodos

En cada clase nos interesa definir **métodos**, que son las partes del código que desarrollan funciones concretas para obtener un resultado o realizar un procedimiento.

```
void metodoCuentaLetras() {  
  
    //Código que contará las letras  
}
```

Constructores

Si lo que quiero es crear un objeto de una clase determinada lo que debo hacer es instanciarlo usando uno de sus constructores.

```
Clase1 objeto = new(Clase1(argumentos));
```

Donde Clase1 es el tipo de Clase a crear y objeto su nombre. Argumentos depende del constructor que utilicemos. Un **constructor** es un **método** especial de cada clase que nos permite inicializar un objeto de esa clase.

Visibilidad de clases, atributos y métodos

Tanto variables como clases y métodos pueden declararse con protección de tipo public, private o protected. La visibilidad de dichos tipos puede verse en la siguiente tabla:

MODIFICADOR	CLASE	PACKAGE	SUBCLASE	TODOS
public	Sí	Sí	Sí	Sí
protected	Sí	Sí	Sí	No
No especificado	Sí	Sí	No	No
private	Sí	No	No	No

Referencias

Breve introducción a Java

<http://www3.uji.es/~belfern/pdidoc/IX26/Documentos/introJava.pdf>

Tutorial de Java

<http://www.ibm.com/developerworks/ssa/java/tutorials/j-introtojava1/>