



## **UAMx: Android301x Jugando con Android - Aprende a programar tu primera App**

Documentación edición 10/2015

David Arroyo Guardeño  
Luis Lago Fernández  
Gonzalo Martínez Muñoz  
Estrella Pulido Cañabate  
Alejandro Sierra Urrecho

# Jugando con Android

Aprende a programar tu primera App

Semana 1. Introducción

## 1. El Entorno de Desarrollo de Android

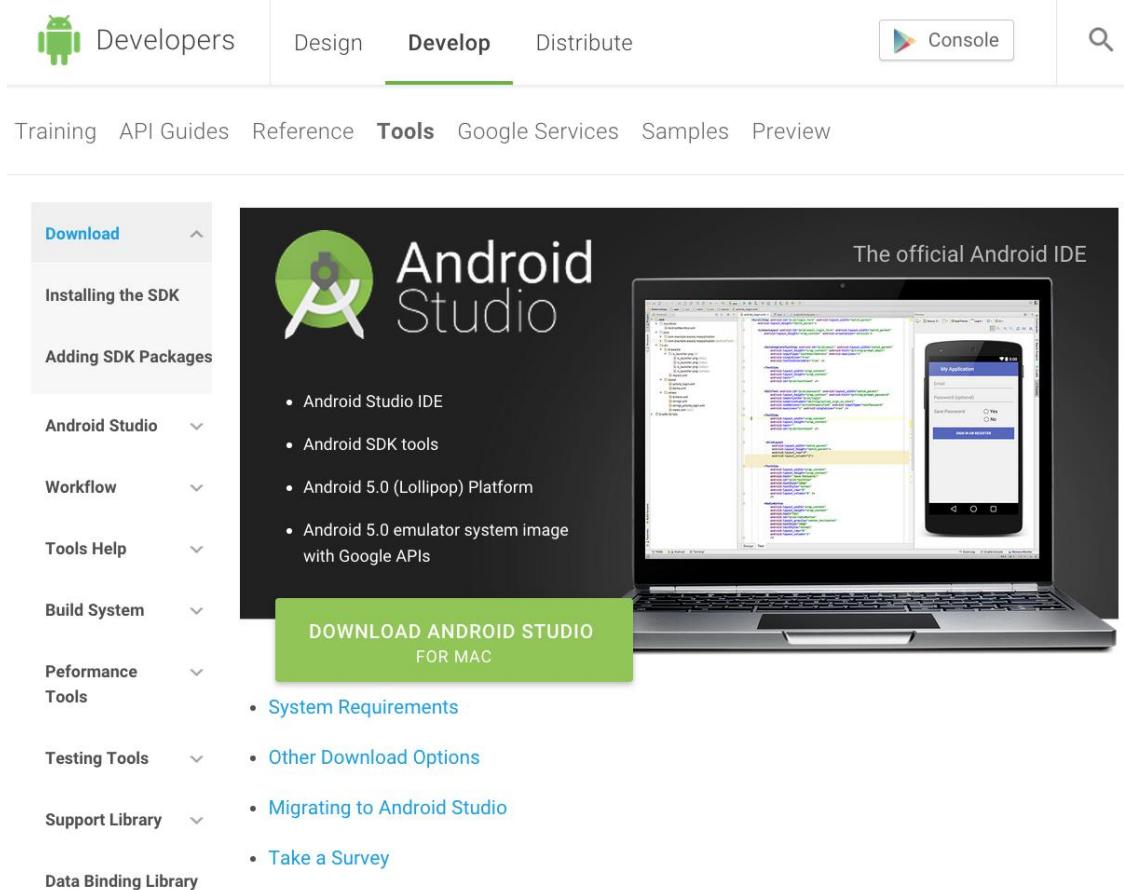
Android Studio incluye todo lo necesario para empezar a desarrollar apps para Android: un entorno de desarrollo y las herramientas del kit de desarrollo (Android SDK Tools).

Antes de descargarlo, asegúrate de tener instalado el kit de desarrollo de Java 6 (JDK 6) o superior. JDK 7 es necesario para desarrollar para Android 5.0. Para comprobar la versión del JDK que tienes instalado puedes ejecutar `javac -version`. Si no tienes el kit o éste es inferior a JDK 6, debes proceder a su instalación.

Puedes descargar Android Studio mediante el siguiente enlace:

<https://developer.android.com/sdk/index.html>

Esta es la página que se abre desde un iMac:



Al pulsar el botón verde de descarga accedemos a una página que muestra los términos y condiciones. Hemos de aceptar las cláusulas del contrato para poder continuar con la descarga:

# Download

Before installing Android Studio or the standalone SDK tools, you must agree to the following terms and conditions.

## Terms and Conditions

This is the Android Software Development Kit License Agreement

### 1. Introduction

1.1 The Android Software Development Kit (referred to in this License Agreement as the "SDK" and specifically including the Android system files, packaged APIs, and Google APIs add-ons) is licensed to you subject to the terms of this License Agreement. This License Agreement forms a legally binding contract between you and Google in relation to your use of the SDK.

1.2 "Android" means the Android software stack for devices, as made available under the Android Open Source Project, which is located at the following URL: <http://source.android.com/>, as updated from time to time.

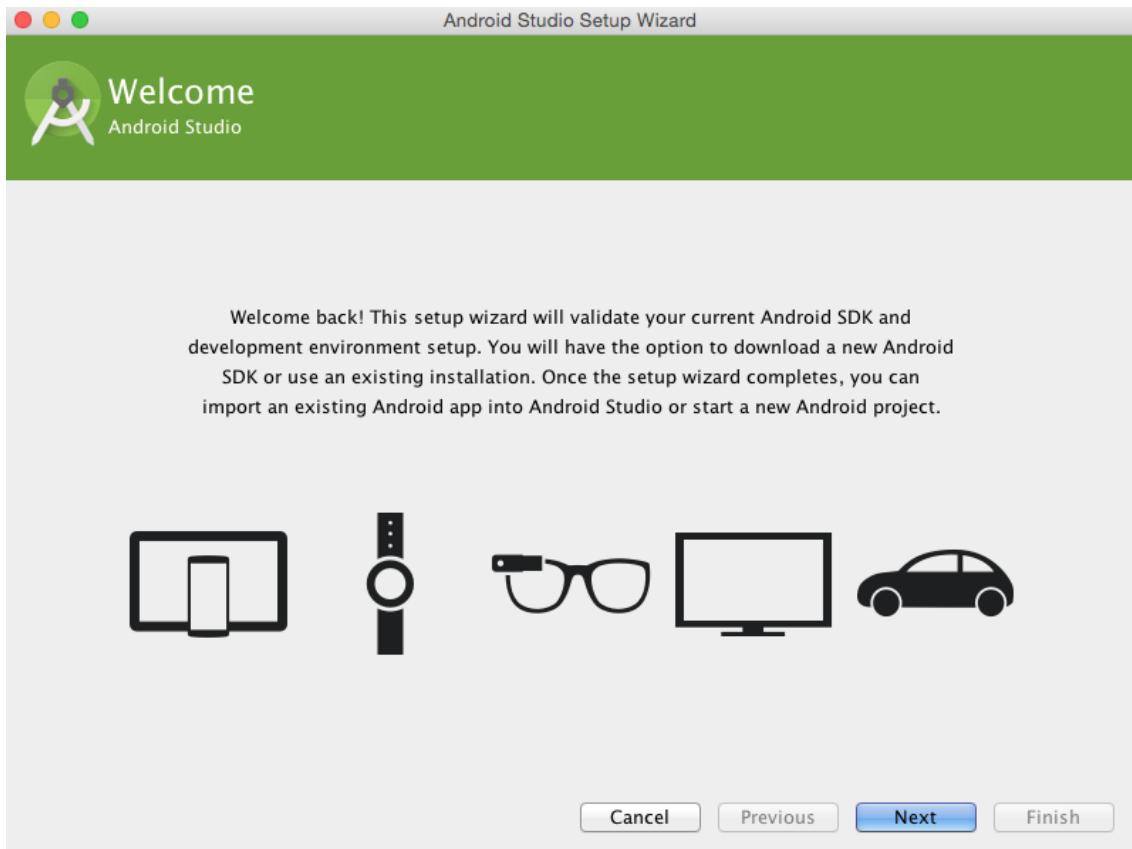
I have read and agree with the above terms and conditions

**DOWNLOAD ANDROID STUDIO FOR MAC**

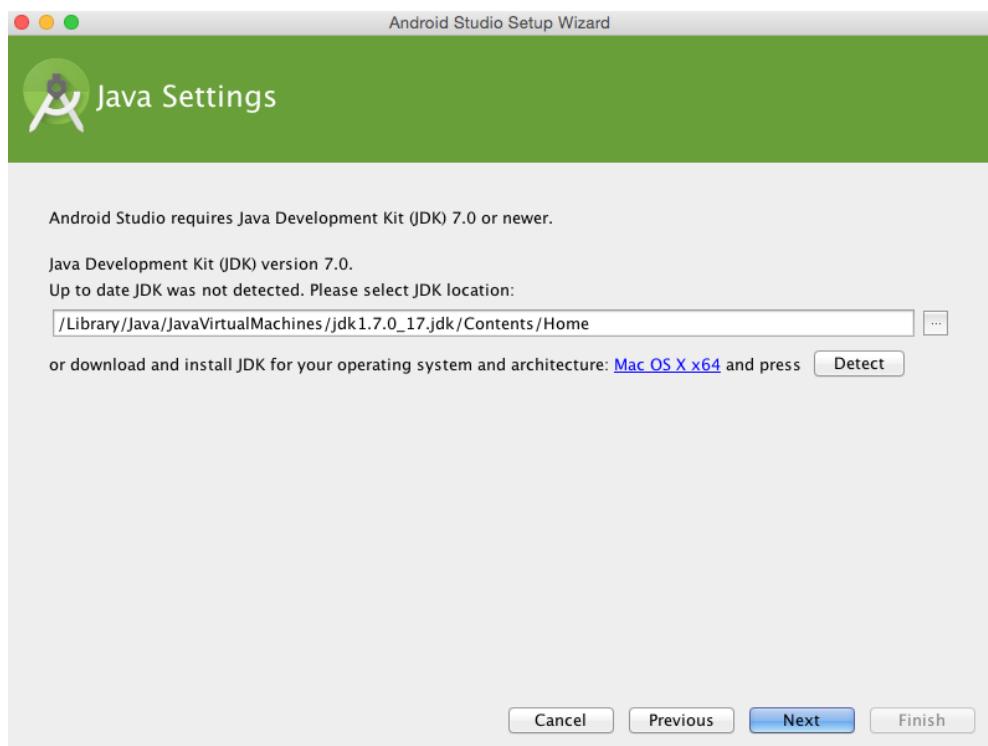
Ahora sí podemos descargar Android Studio que, en el caso de iOS, se descargará como un fichero .dmg. Ejecutamos el fichero .dmg y copiamos la app resultante en la carpeta Applications:



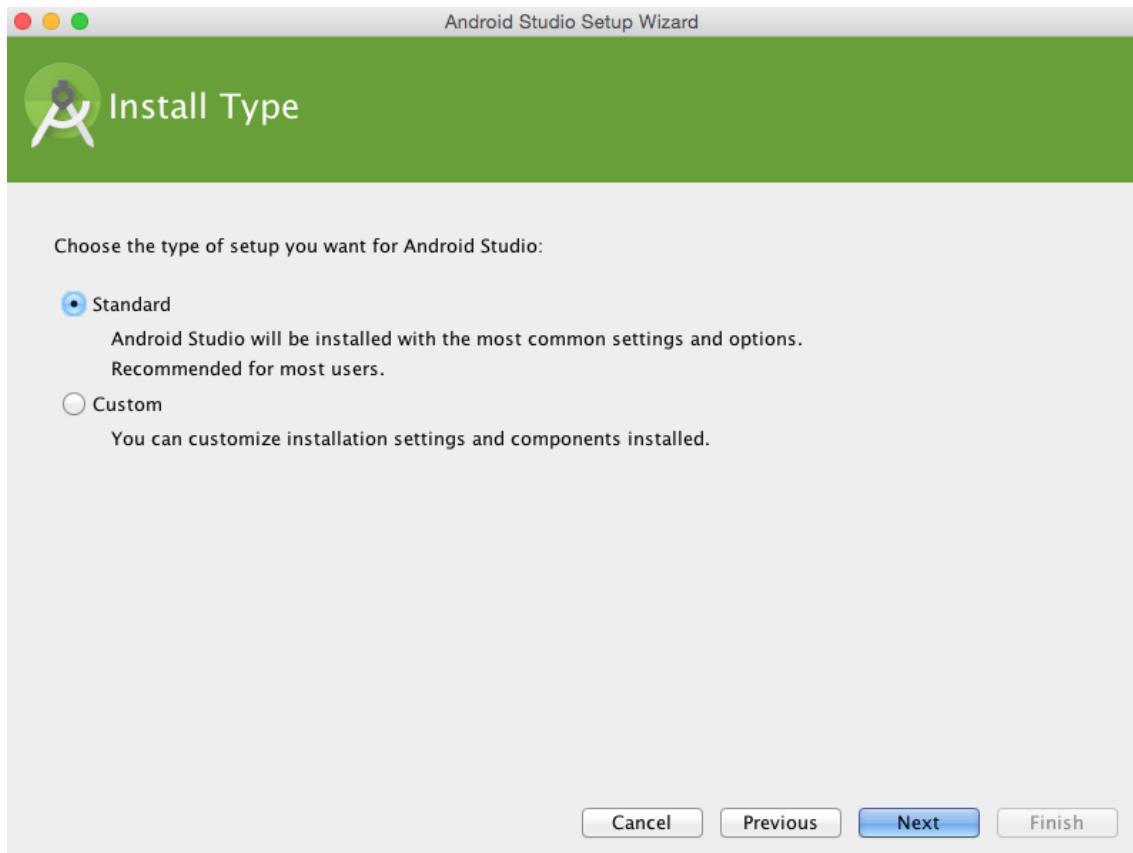
La primera vez que ejecutemos el programa deberemos responder a unas pocas preguntas:



Primero el programa de ajuste nos pregunta sobre la localización del JDK:



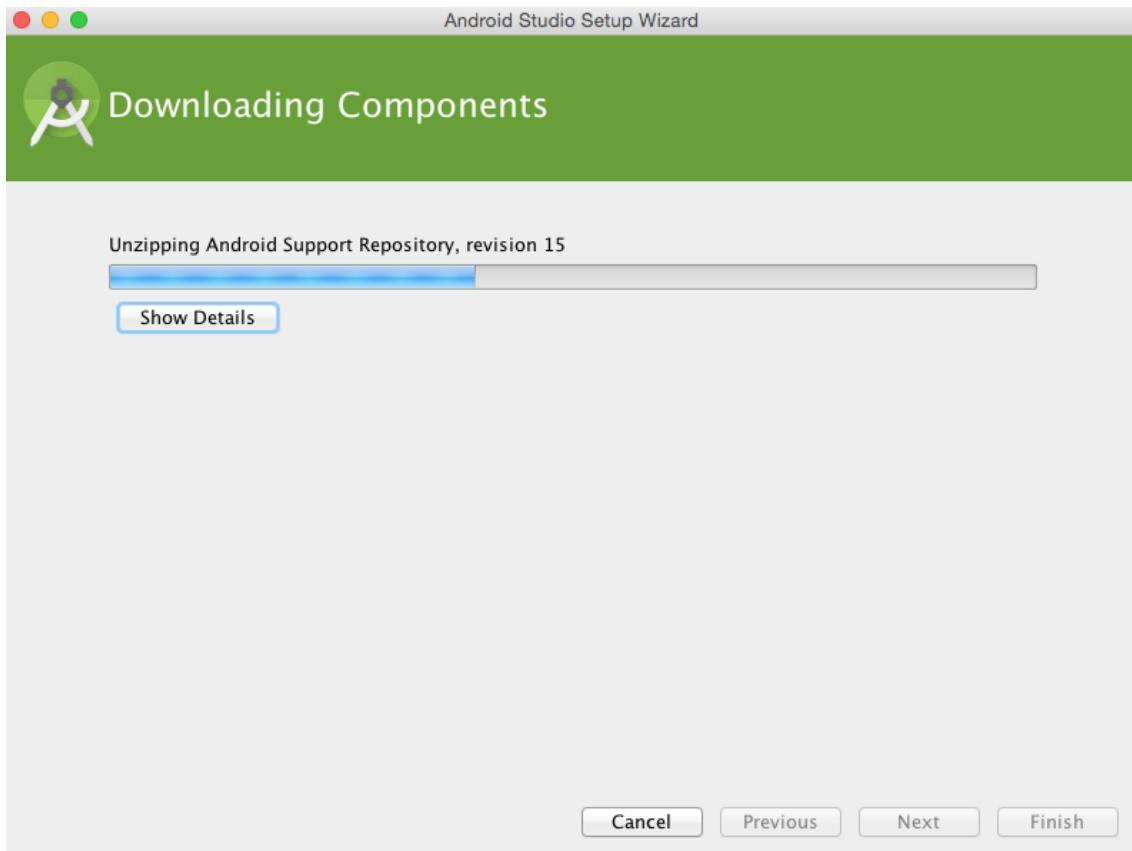
A continuación se nos pregunta sobre el tipo de instalación deseada, a lo que contestaremos Standard:



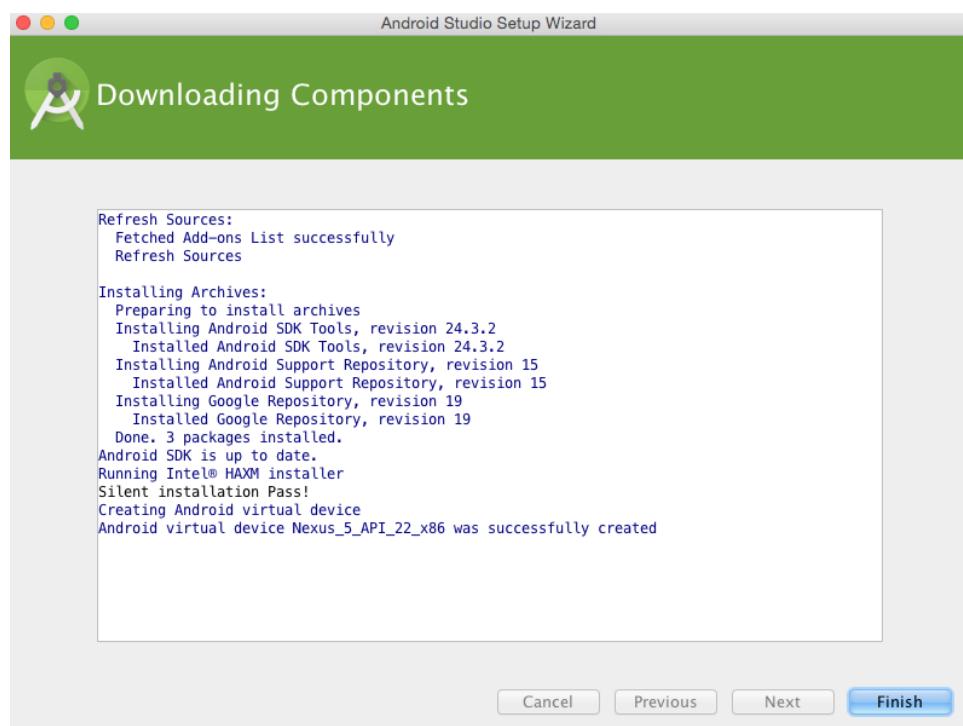
Finalmente, el setup actualizará el software necesario, lo que dependerá de cada instalación, para lo que tendremos que aceptar nuevas condiciones:



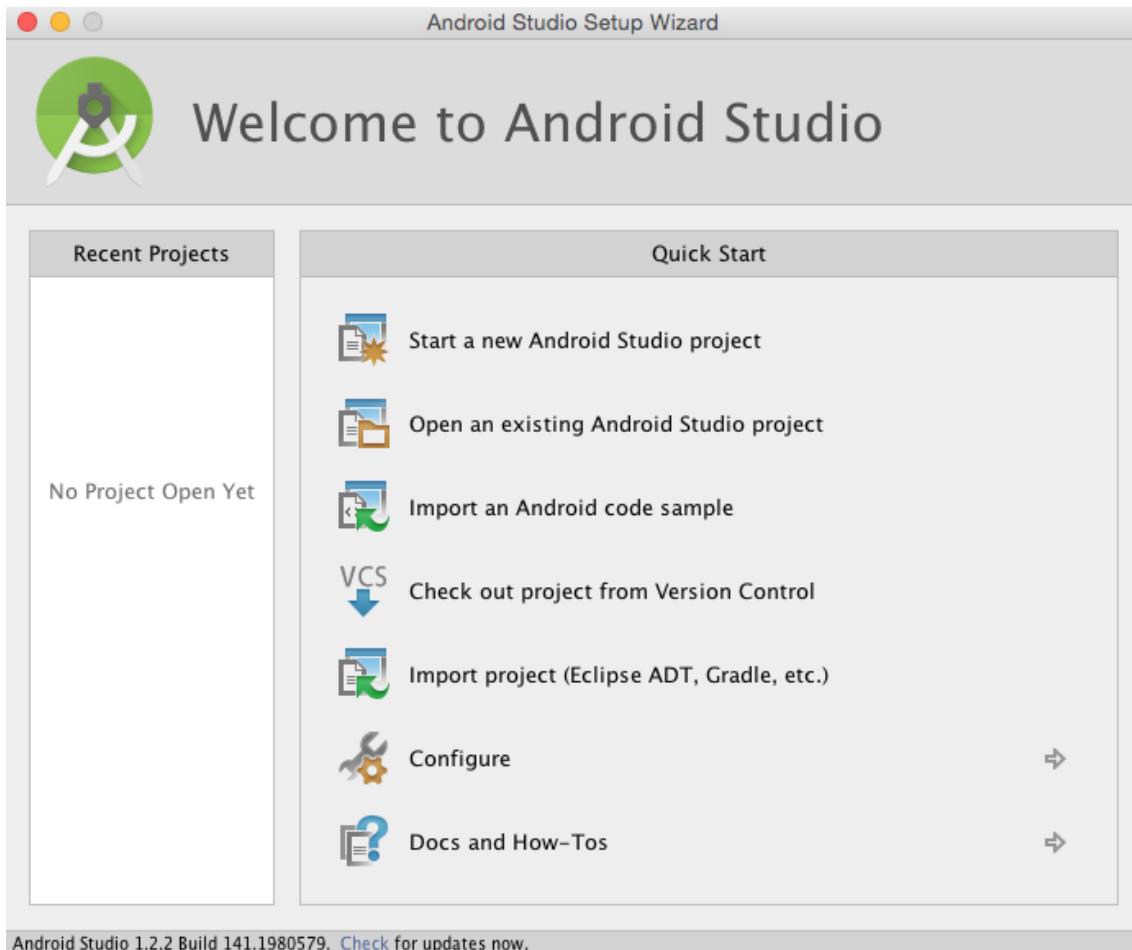
El proceso puede llevar más o menos tiempo dependiendo del software que ya haya instalado en la máquina:



Al final del proceso se mostrará un resumen de las tareas llevadas a cabo:



Lo que sigue es el menú del entorno de programación que se muestra al pulsar el botón Finish:

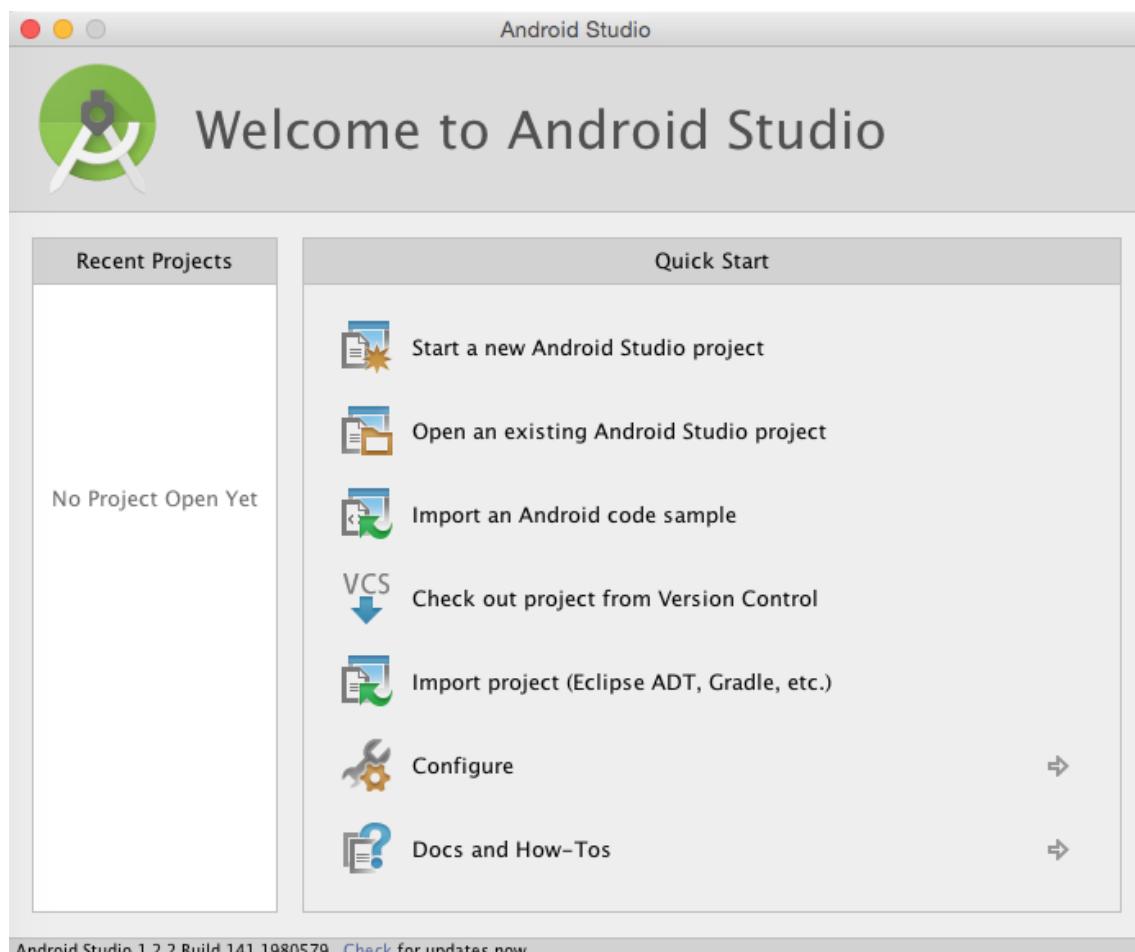


En la siguiente unidad veremos cómo crear una aplicación sencilla mediante un nuevo proyecto de Android Studio.

## 2. El gestor del SDK de Android

En esta unidad vas a aprender a manejar el gestor del kit de desarrollo de software de Android, una aplicación gráfica que te permitirá averiguar, por ejemplo, si tu versión del sistema operativo, así como de las herramientas de desarrollo, es la última disponible o, por el contrario, existen versiones más recientes. El SDK separa herramientas, plataformas y otros componentes en paquetes que se pueden descargar individualmente mediante el gestor.

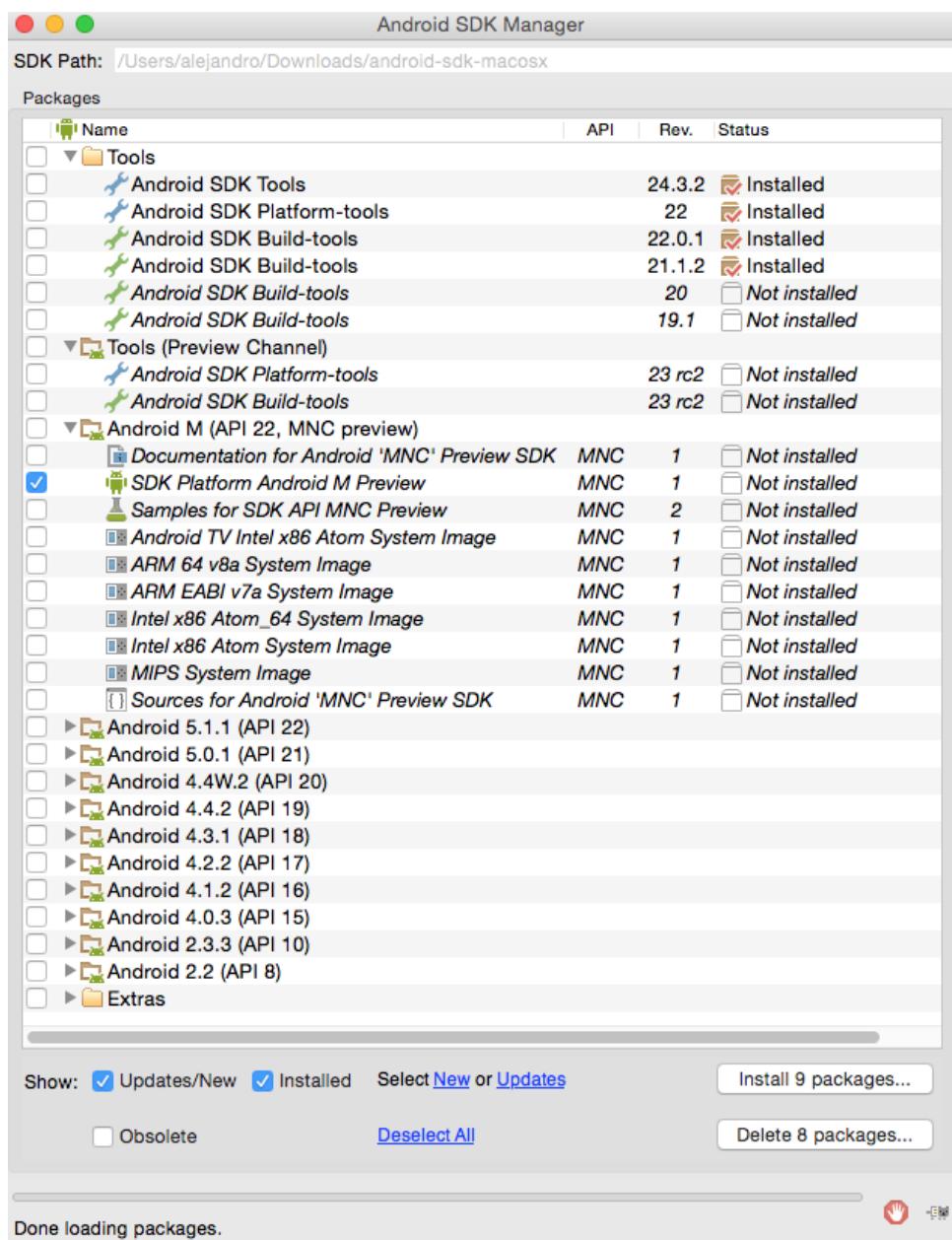
Después de arrancar Android Studio, pasados unos segundos, verás el menú principal del entorno de desarrollo:



Para acceder al gestor del SDK debes pulsar en **Configure**:



La primera de las opciones del menú de configuración es la del gestor. Pulsa en `SDK Manager` para acceder al gestor. La ventana que aparece muestra las herramientas del SDK (*Tools*, *Platform-tools* y *Build-tools*) así como toda la colección de APIs ordenados de más a menos reciente. La carpeta *Extras* contiene, entre otras cosas, la biblioteca de apoyo (*Android Support Library*):

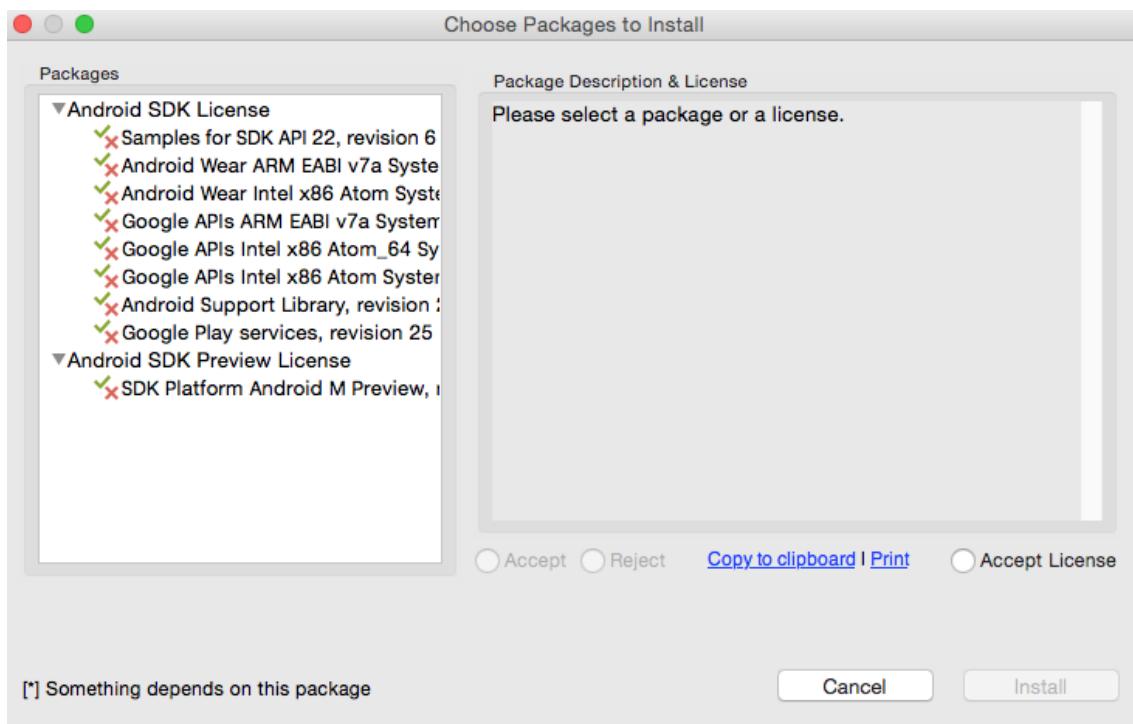


Cada versión del sistema operativo tiene su propio nivel API. Por ejemplo, Android 5.1.1 tiene un nivel API igual a 22. Para cada nivel existen dos plataformas:

- SDK Platform.
- Google APIs.

La plataforma Google API contiene APIs adicionales de Google como, por ejemplo, la Biblioteca de Mapas de Google necesaria si tu app utiliza Google Maps.

Al pulsar el botón `Install 9 packages...` se muestra el siguiente diálogo:

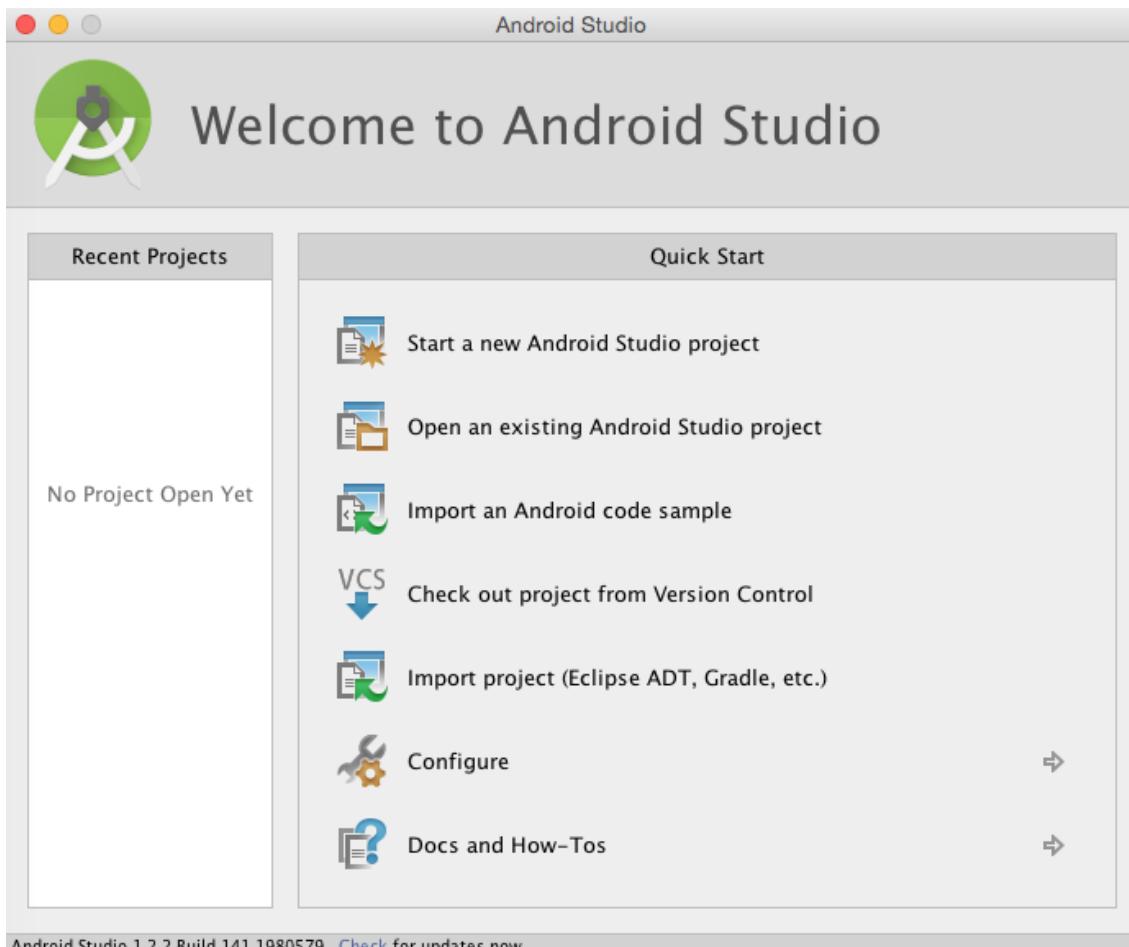


Hemos de marcar los paquetes deseados en la ventana de la izquierda y marcar la casilla `Accept License` antes de pulsar `Install`. El proceso de instalación puede llevar un cierto tiempo dependiendo de los paquetes que se instalen.

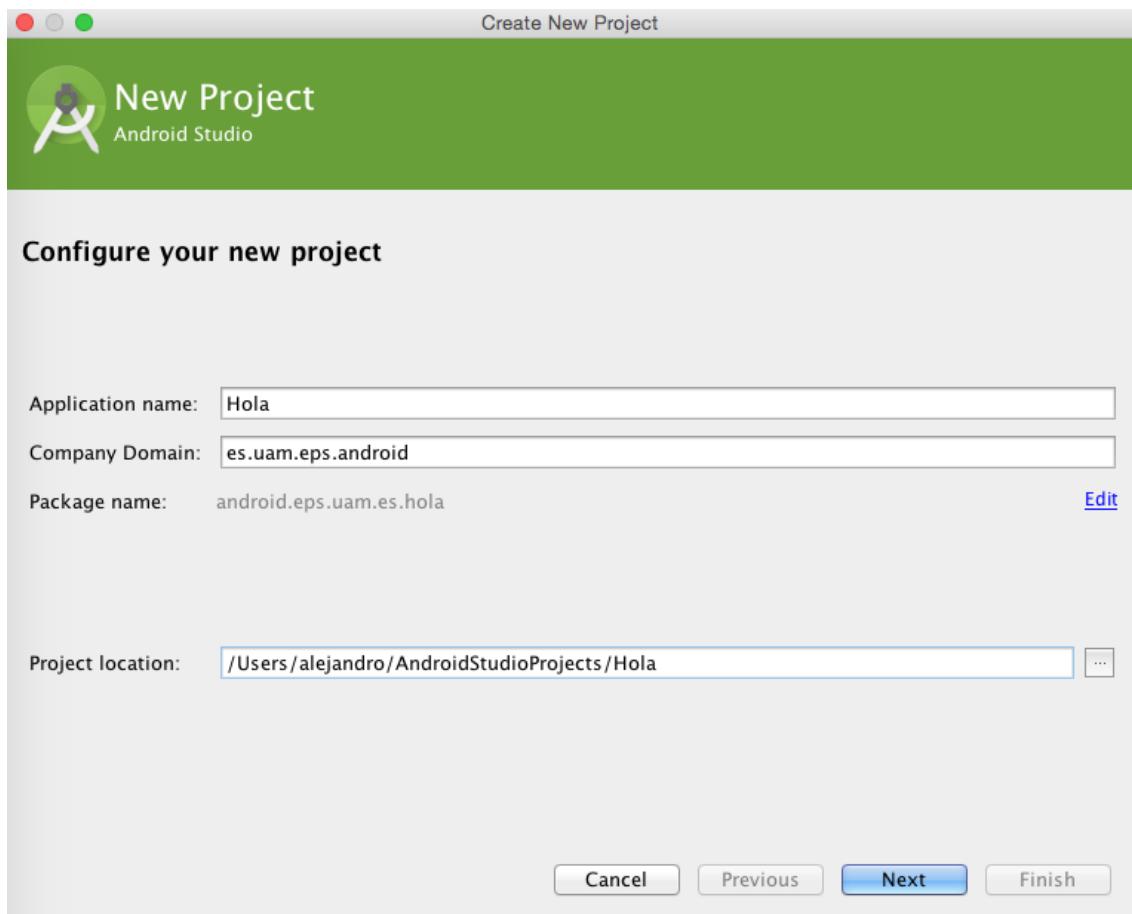
En la siguiente unidad aprenderemos a simular un dispositivo dentro del entorno de desarrollo.

### 3. La primera aplicación

Para crear nuestra primera aplicación tenemos que arrancar el entorno de programación para acceder al menú principal:



A continuación pulsaremos el botón Start a new Android Sutdio project, el primer ítem de la lista. Esta es la ventana que el entorno muestra a continuación:



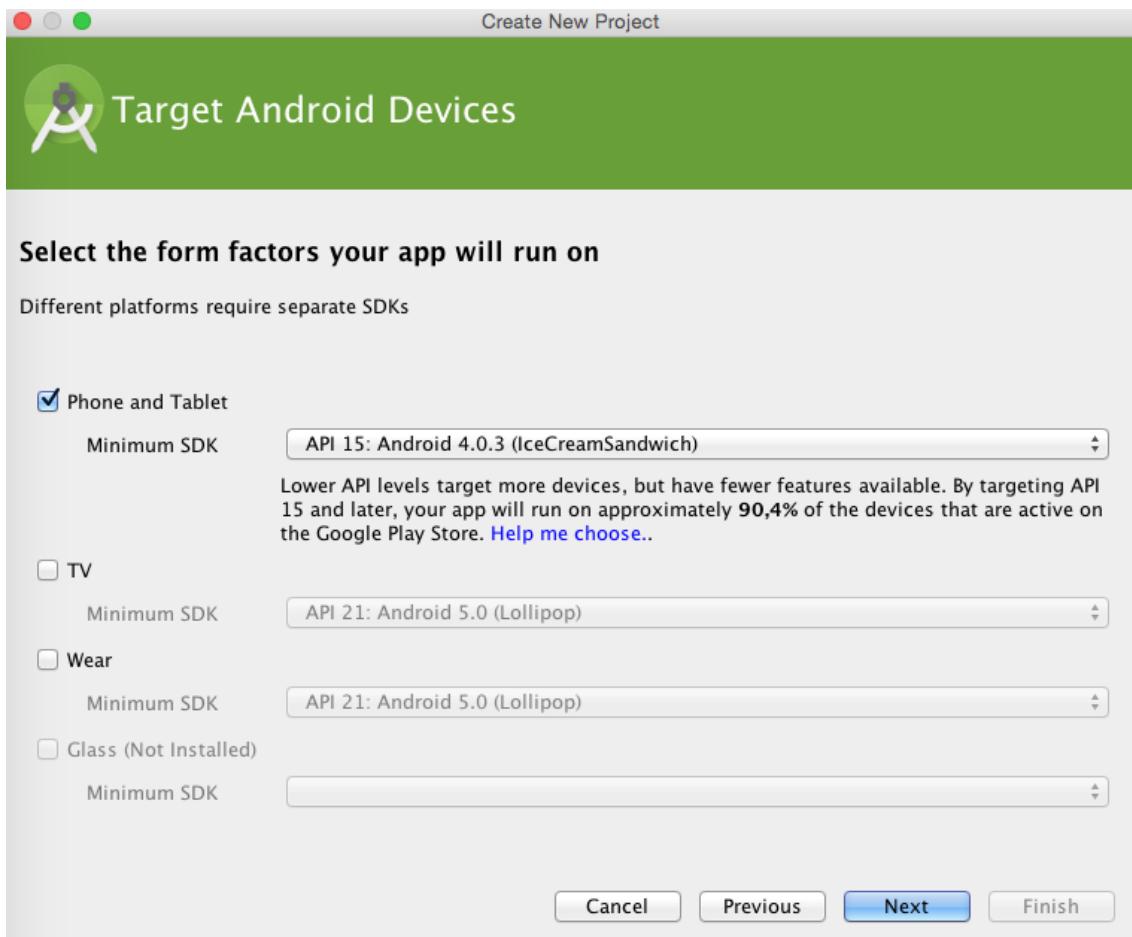
Esta ventana te permite elegir un nombre para tu aplicación (`Application name`), así como un dominio (`Company Domain`) y una carpeta para alojar el proyecto (`Project location`). Como puedes comprobar enseguida, Android Studio crea una carpeta de nombre igual al elegido para la aplicación dentro de la carpeta `AndroidStudioProjects`.

Elige el nombre del dominio (`Company Domain`) siguiendo el estilo de Java conocido como convenio de dominio inverso:

`es.uam.eps.android`

En primer lugar, el nombre de dominio garantiza que los nombres de los paquetes sean únicos evitando colisiones de nombres entre organizaciones. Además, este convenio garantiza que los paquetes dentro de una misma organización quedan bien estructurados. Hay que tener en cuenta que cada '.' del nombre del paquete se traducirá en un directorio. De este modo, los directorios se organizarán de más general ('`es`' en nuestro caso) a más particular ('`android`').

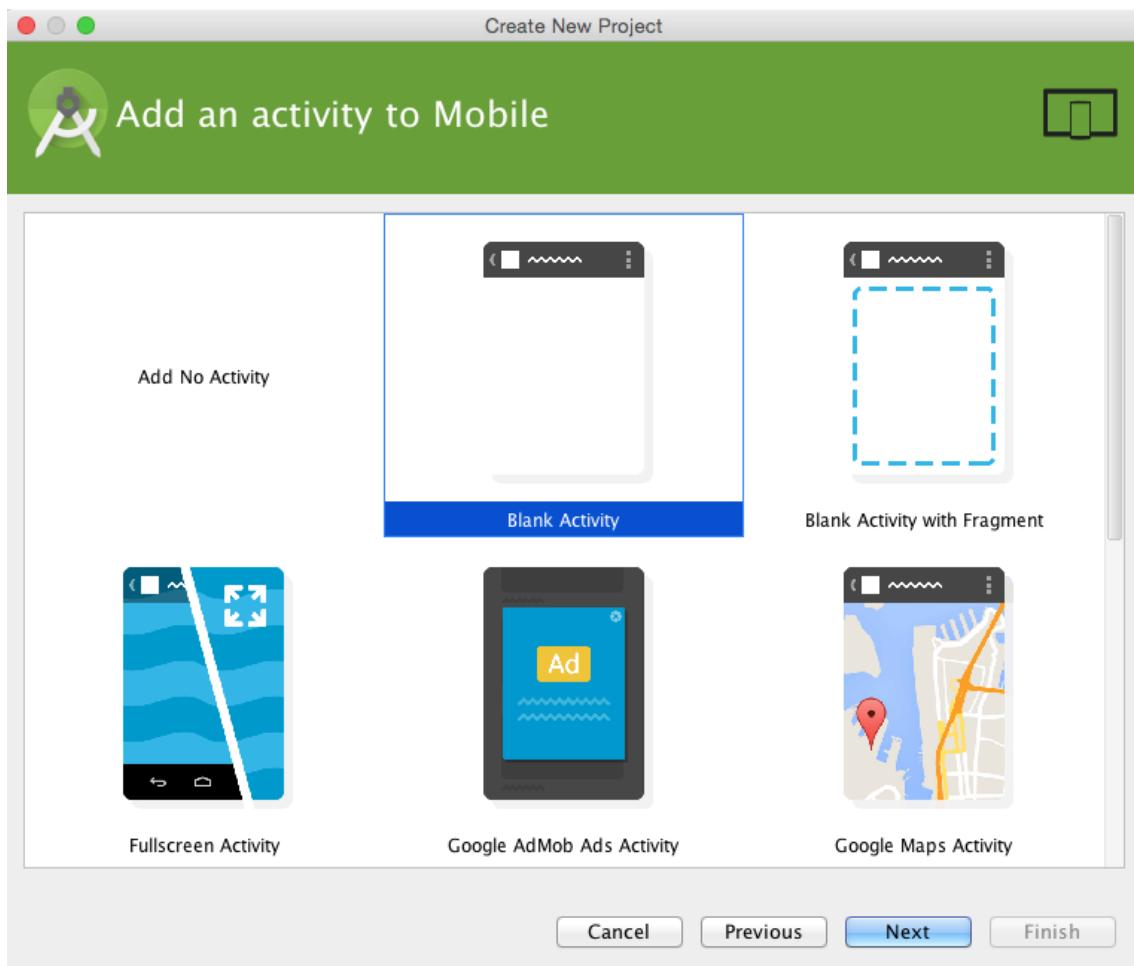
Al pulsar `Next` tendremos ocasión de seleccionar la plataforma para la que desarrollamos nuestra aplicación (Phone and Tablet, TV, Wear, ...). Diferentes plataformas exigen SDKs independientes:



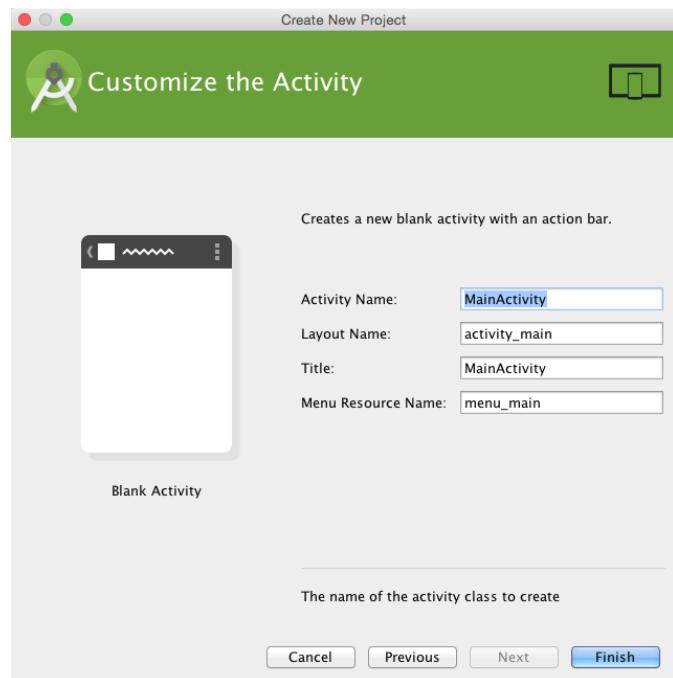
Elige un valor para `Minimum SDK`. En nuestro caso hemos elegido API 15. Esto quiere decir que esta aplicación no podrá ser instalada en dispositivos con versiones del API igual a 14 o inferiores. Cuanto más baja sea esta versión, más dispositivos podrán instalar la aplicación. Sin embargo, hay que tener en cuenta que si escogemos una versión demasiado baja no podremos utilizar muchos elementos de Android, los introducidos por las versiones posteriores.

Como puedes observar en la imagen de arriba, en el momento de escribir este documento, el API elegido permite instalar la aplicación en más del 90% de los dispositivos activos en Google Play Store.

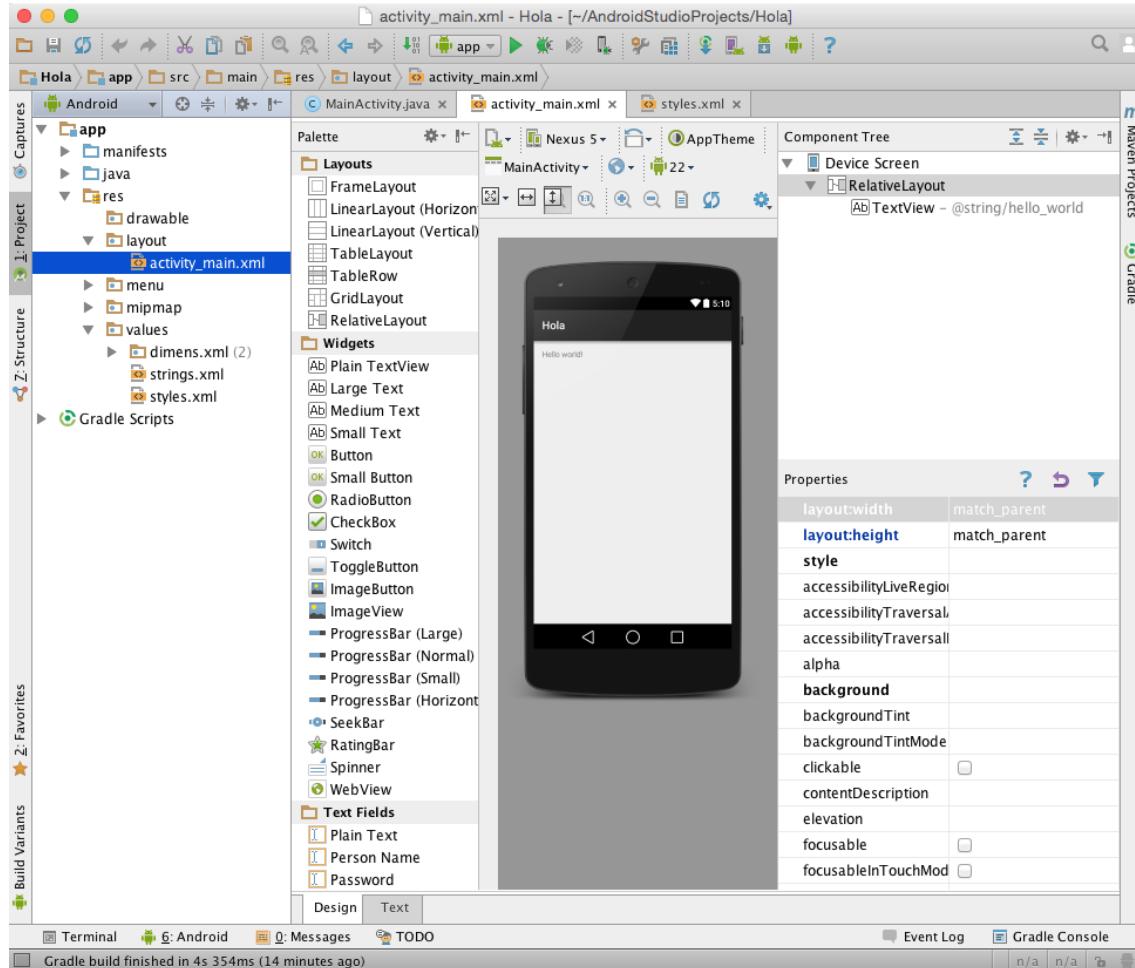
Tras pulsar `Next` podremos elegir el tipo de actividad o pantalla que queremos para nuestra aplicación. Hemos seleccionado `Blank Activity`:



Pulsa **Next** para poder elegir el nombre de la actividad (`MainActivity`) así como el nombre del fichero de diseño, título y fichero de diseño del menú de opciones:

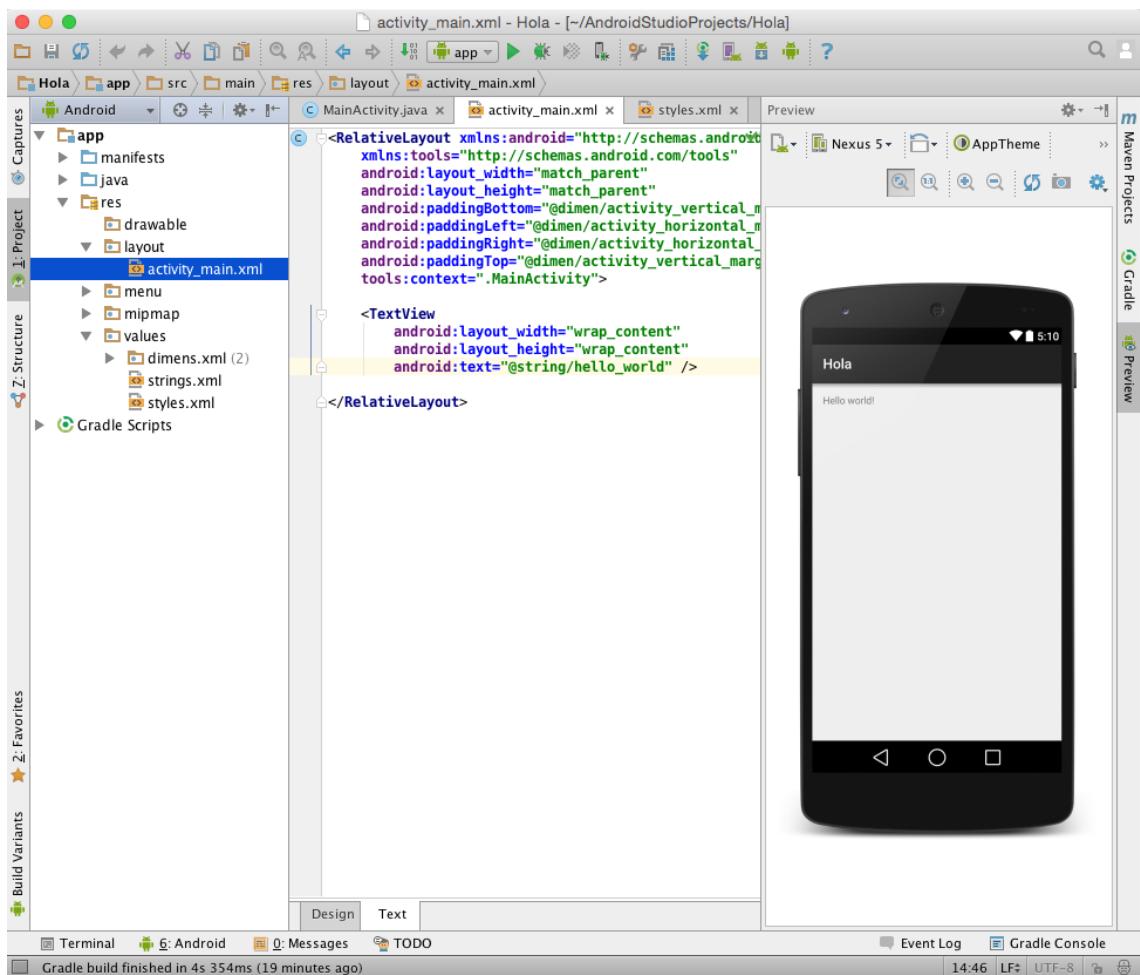


Tras pulsar `Finish` y después de unos segundos de construcción, veremos un diálogo con recetas (tips) cubriendo en parte la ventana del entorno de programación. De momento cerraremos este diálogo para apreciar mejor la estructura de la ventana que es la siguiente:



Como puedes observar esta ventana tiene muchas componentes, que estudiaremos más adelante con detalle. De momento, fíjate en la zona central donde puedes apreciar el aspecto de la aplicación que acabas de crear, una pantalla que muestra el mensaje `Hello world!`.

Android Studio facilita mucho el desarrollo de la interfaz de usuario pues nos muestra su aspecto dinámicamente, a la vez que lo desarrollamos. Para ello tienes que seleccionar el fichero `activity_main.xml` en la ventana de la izquierda y la pestaña `Text` en la parte inferior del entorno. Verás una simulación del dispositivo elegido, un Nexus 5 en este caso. Puedes elegir otros dispositivos, así como orientaciones y estilos con los iconos que se muestran encima de la simulación:



Además de esta ayuda gráfica, también podemos crear dispositivos virtuales específicos con los que ejecutar la aplicación sin la necesidad de disponer de un dispositivo físico. Esto es lo que veremos en la siguiente unidad.

## 4. Dispositivos virtuales

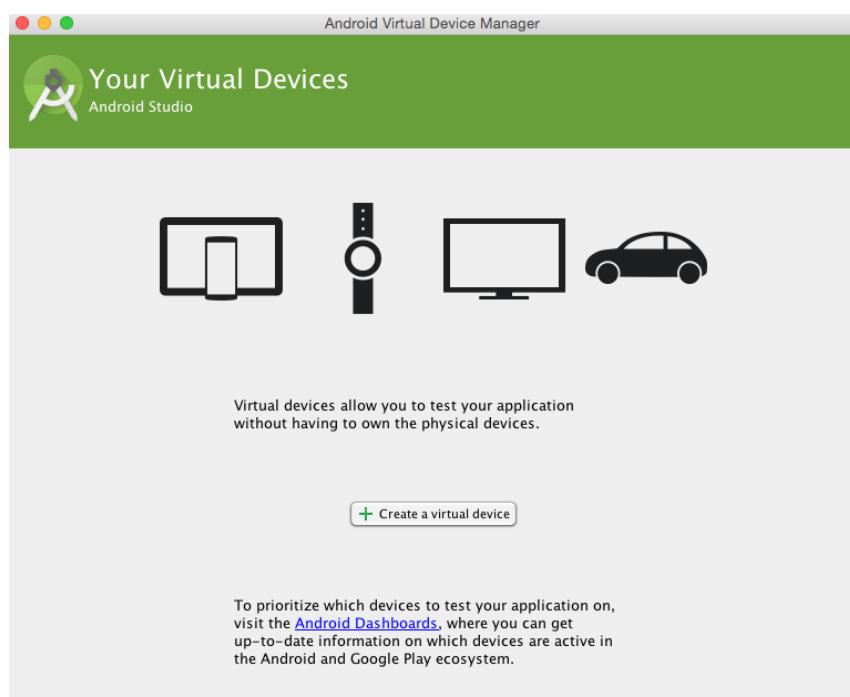
Un Dispositivo Virtual Android (AVD en inglés) es un emulador de un dispositivo real, que te permite ejecutar una aplicación sin necesidad de contar con un teléfono físico. Puedes crear tantos AVDs como quieras con configuraciones diferentes, es decir, distintas versiones del sistema operativo, tamaños de pantalla, teclado, memoria interna, ...

Para poner a prueba tu app deberías crear al menos un dispositivo virtual por cada nivel API igual o superior a la versión mínima elegida al crear el proyecto. De esta forma podrás comprobar la compatibilidad hacia delante de tu app, es decir, comprobar que los usuarios que se la instalen y reciban actualizaciones del sistema, seguirán disfrutando de un funcionamiento adecuado.

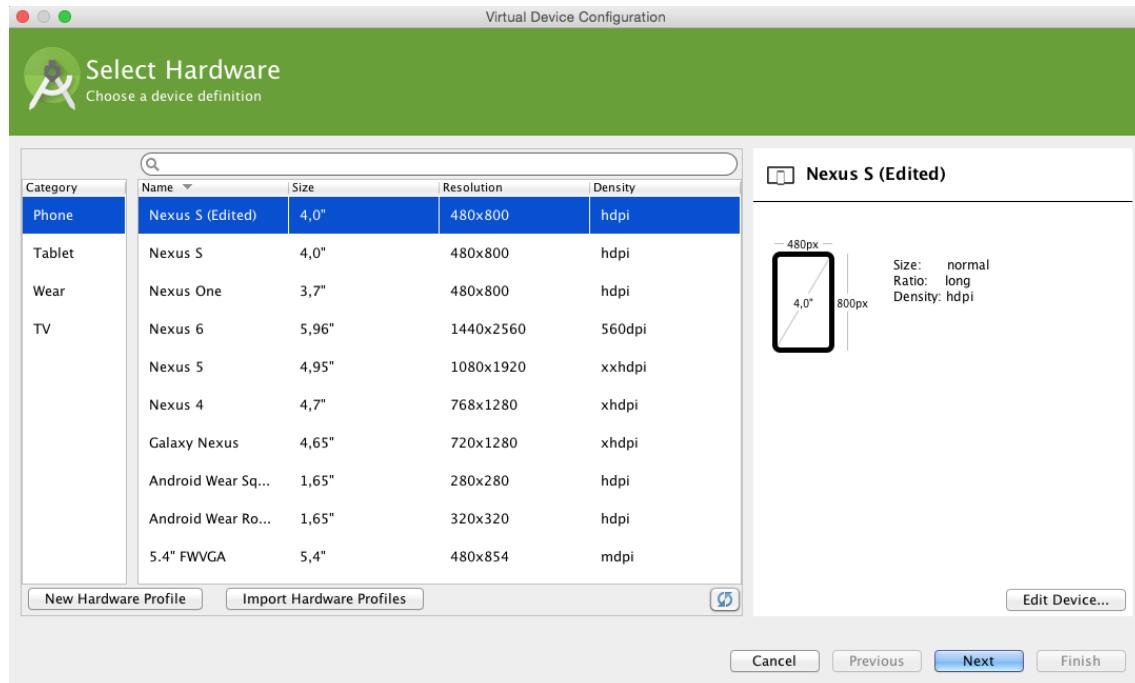
Para acceder al gestor de dispositivos virtuales, vamos a abrir primero Android Studio, para luego seleccionar el proyecto `Hola` que hemos desarrollado en la unidad anterior. A continuación ejecutaremos el gestor de dispositivos virtuales pulsando su ícono, que encontrarás en la parte derecha de la barra de herramientas del entorno:



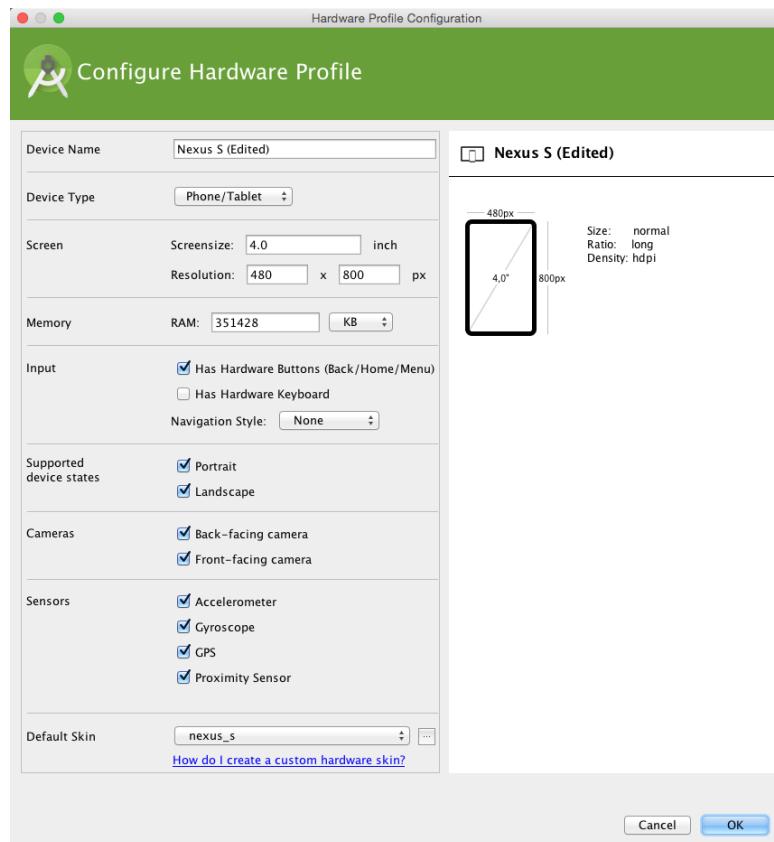
Este es el aspecto del gestor de dispositivos virtuales:



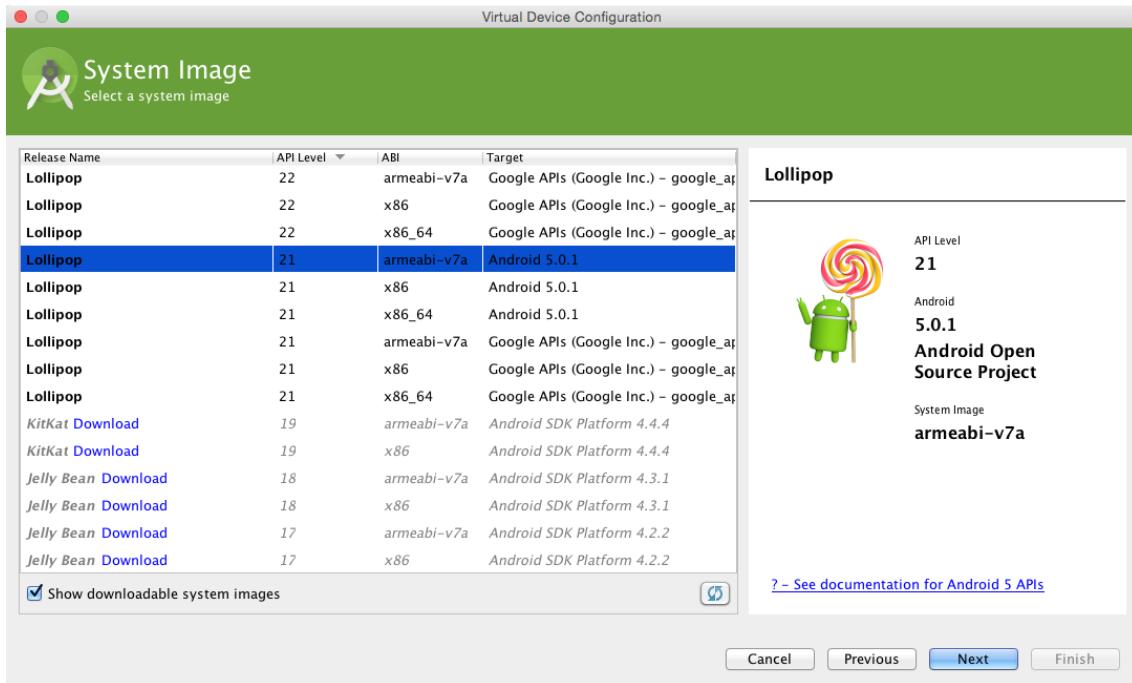
Para crear un nuevo AVD debes pulsar el botón `Create a virtual device`. Lo que se abre es la ventana de configuración del dispositivo:



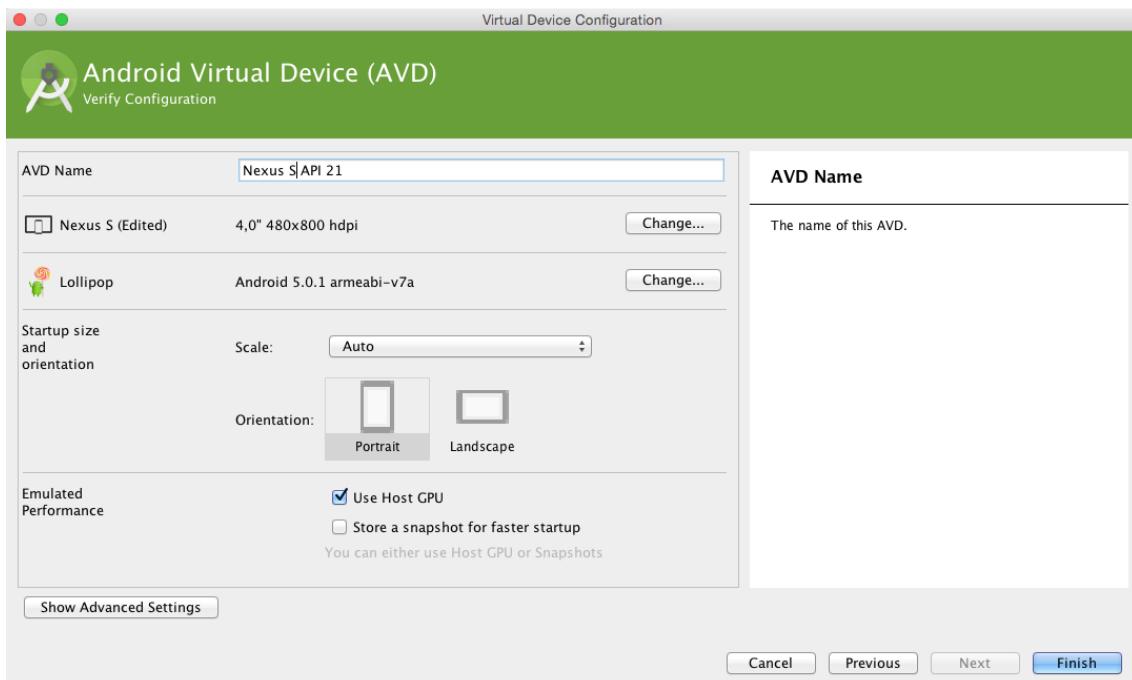
Hemos elegido un dispositivo Nexus S. Podemos modificar la configuración de hardware como la cámara o los sensores pulsando el botón `Edit Device...`:



Pulsamos **Cancel** para volver a la ventana de selección de dispositivo y el botón **Next**, con lo que se muestra el selector de versiones del sistema:



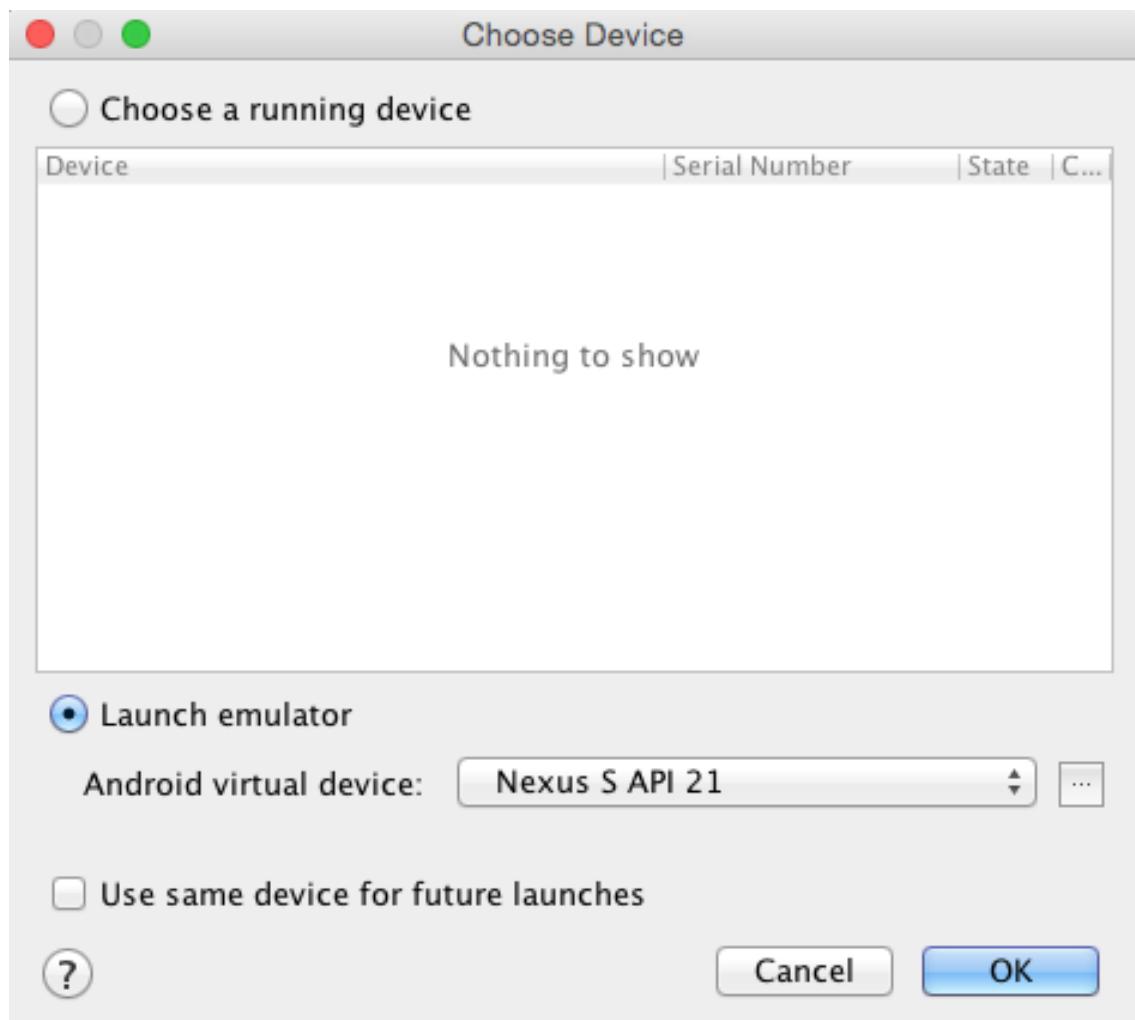
Hemos seleccionado el API 21 y pulsado **Next**, con lo que solo nos falta elegir un nombre para el dispositivo y verificar la configuración:



Tras pulsar el botón `Finish` y esperar unos minutos para la creación del dispositivo, ya estamos preparados para ejecutar la app. Para ello has de pulsar el icono de ejecución en la zona central de la barra de herramientas del entorno:



Al pulsarlo aparecerá un diálogo para seleccionar el dispositivo en el que se desea ejecutar la app:



En este momento no tenemos ningún dispositivo ejecutándose, por lo que el sistema nos invita a que arranquemos nuestro nuevo dispositivo virtual, de nombre `Nexus S API 21`. Pulsamos el botón `OK` y, tras un periodo de unos minutos, podremos ver nuestra app simulada en el dispositivo virtual.

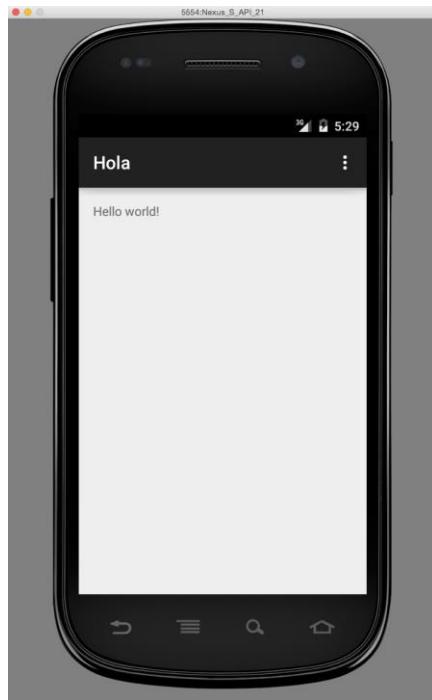
Antes de nada, has de desbloquear el emulador, arrastrando el candado hacia arriba con el puntero del ratón, como si se tratara de un dispositivo físico:



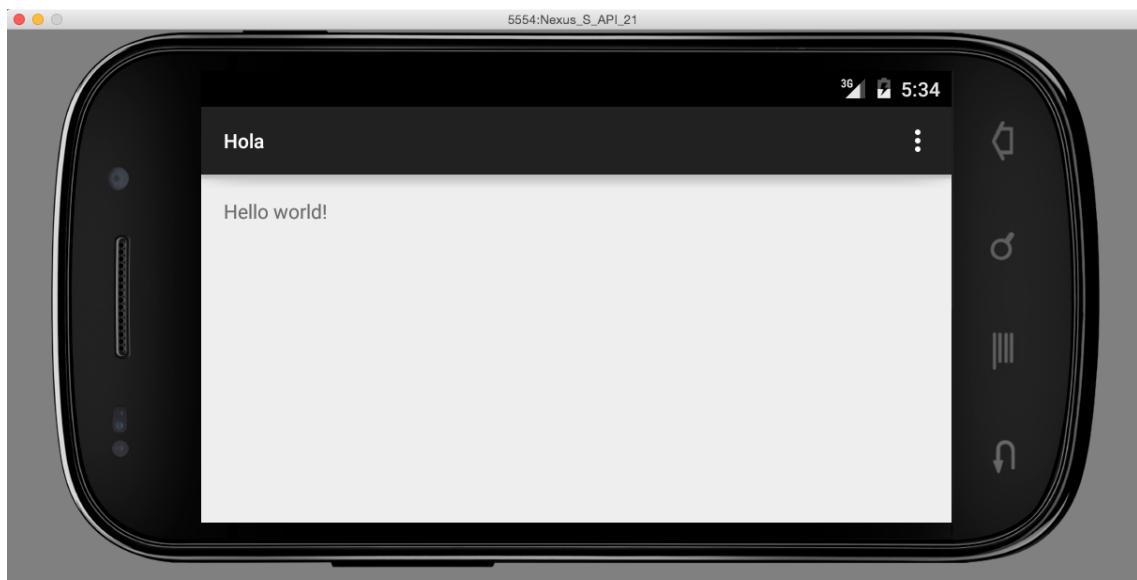
El resultado es el siguiente:



Es probable que la app no haya arrancado debido al tiempo transcurrido para el arranque del dispositivo. Vuelve a pulsar el botón de ejecución de la app para poderla ver simulada en el dispositivo virtual:



Para ejecutar la aplicación en modo apaisado debes pulsar la siguiente combinación de teclas: Linux (ctrl + F11), Mac (ctrl + cmd + fn + F12) y Windows (left-control + F11):

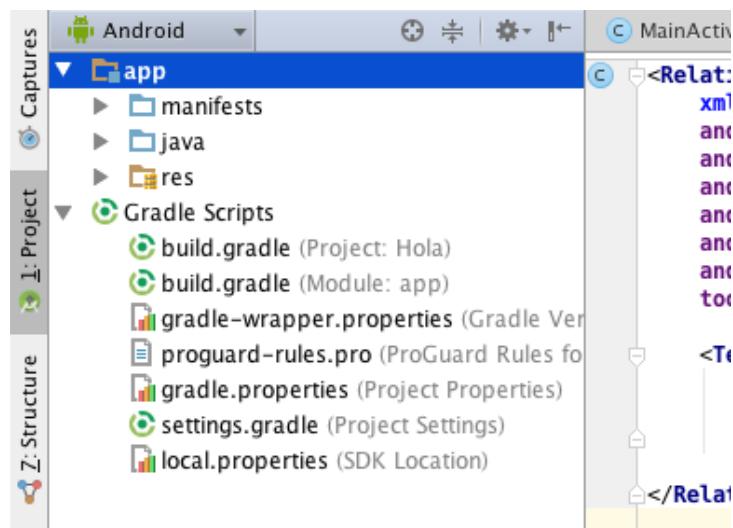


En la siguiente unidad estudiaremos la estructura de un proyecto de Android Studio.

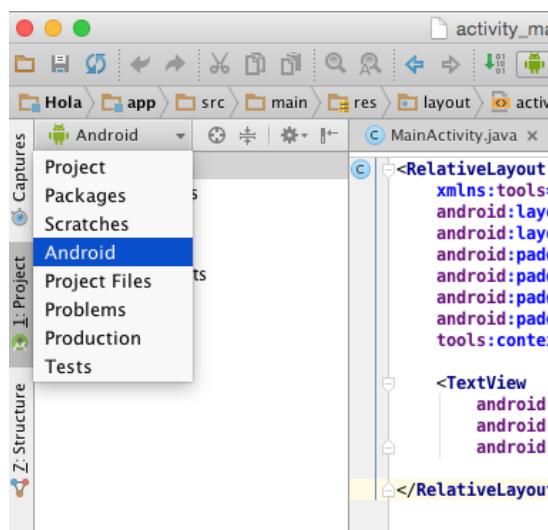
## 5. Estructura de un proyecto

Android Studio permite elegir entre distintas formas de presentación de los ficheros de tu proyecto. Por omisión, lo hace en la vista de proyecto denominada **Android**. Éstas son las carpetas en las que esta vista organiza los ficheros:

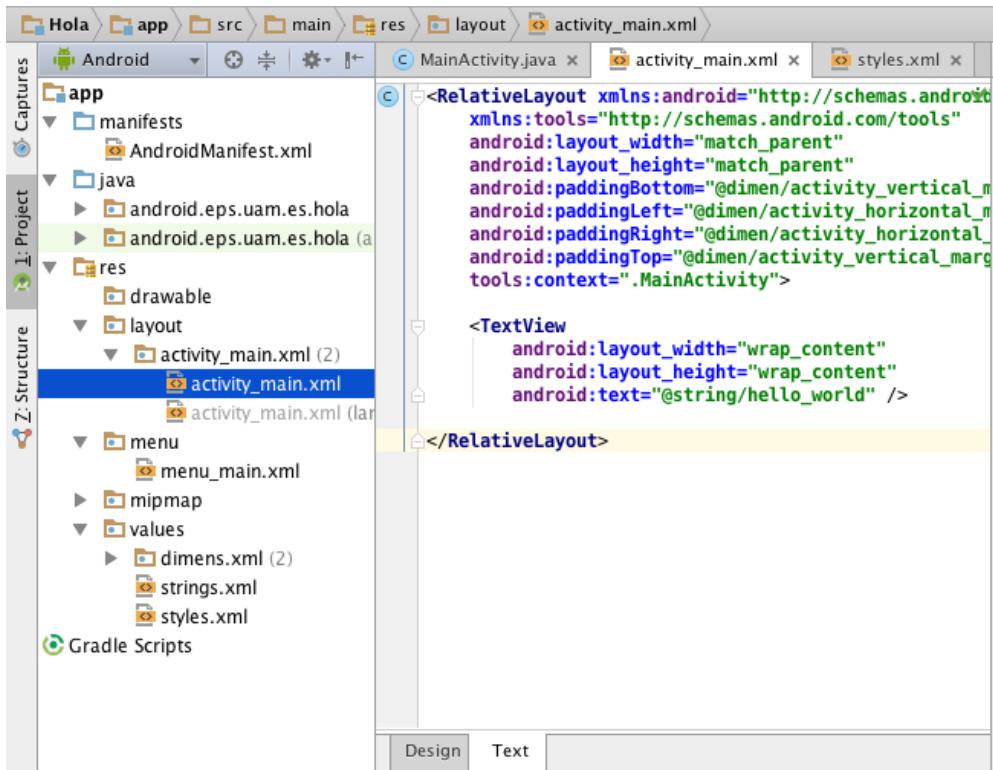
- **manifests**: ficheros de manifiesto donde se especifica, por ejemplo, la versión mínima del sistema operativo para poder instalar la aplicación.
- **java**: ficheros fuente escritos en Java.
- **res**: ficheros de recursos como, por ejemplo, cadenas de caracteres, imágenes, colores, etc.
- **Gradle Scripts**: Gradle es una herramienta que automatiza la construcción de tu proyecto mediante ficheros `build.gradle` escritos en Groovy gracias a los elementos proporcionados por el plugin de Android para Gradle.



La vista de proyecto se puede modificar mediante el siguiente selector desplegable:



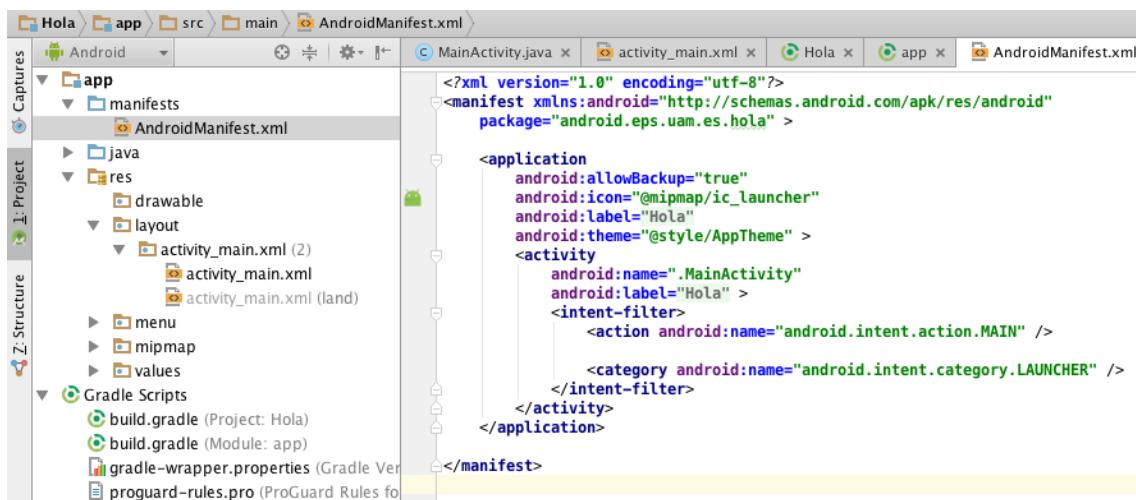
En la siguiente captura puedes estudiar con más detalle la estructura de las tres primeras carpetas de la vista Android del proyecto Hola:



## 5.1 La carpeta manifests

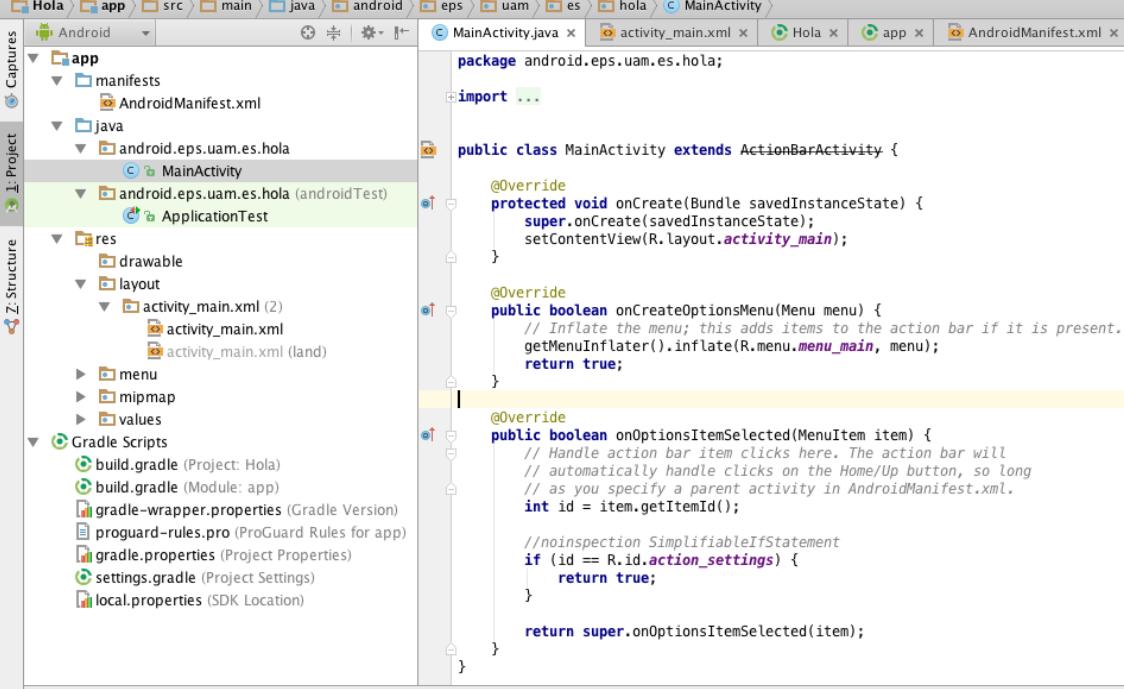
Esta carpeta contiene el fichero llamado `AndroidManifest.xml` en la raíz del proyecto. Este fichero contiene información como por ejemplo:

- El nombre del paquete Java especificado en el atributo `package`.
- Las actividades que integran la aplicación: `MainActivity` en nuestro caso.
- La etiqueta de cada actividad (`android:label`) que se mostrará en la barra de acción del dispositivo.



## 5.2 La carpeta java

Esta carpeta contiene el código Java de nuestro proyecto, organizado por paquetes:



The screenshot shows the Android Studio Project Structure view. The left sidebar shows the project tree with the following structure:

- Hola
- app
- src
- main
- java
- android
- eps
- uam
- es
- hola
- MainActivity
- activity\_main.xml
- AndroidManifest.xml

The right pane displays the code for `MainActivity.java`:

```
package android.eps.uam.es.hola;
import ...;

public class MainActivity extends ActionBarActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

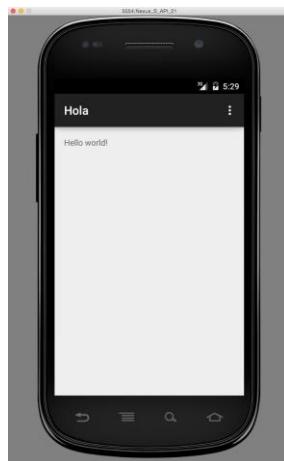
        return super.onOptionsItemSelected(item);
    }
}
```

## 5.3 La carpeta res

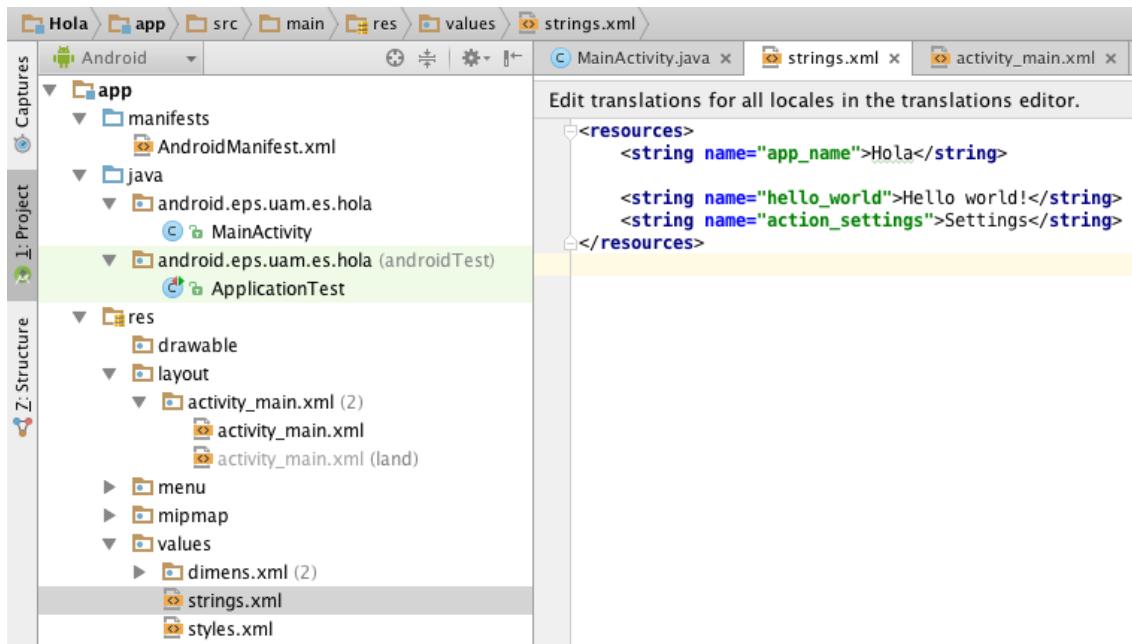
La carpeta res contiene recursos (*resources* en inglés) utilizados por tu aplicación.

Por ejemplo, las constantes de tipo cadena de caracteres se suelen situar en ficheros de recursos XML. El código Java puede acceder a estas cadenas a través de su identificador. Android se encarga de generar estos identificadores automáticamente y los guarda en la clase `R`. Veamos un ejemplo:

El texto mostrado por la aplicación `Hola` utiliza uno de estos recursos:



Se trata de una cadena de nombre `hello_world` y valor `Hello World!` declarada en el fichero `strings.xml` dentro de la carpeta `res/values`:



El código Java se refiere a esta cadena por medio del siguiente identificador generado automáticamente: `R.string.hello_world`.

Si mantienes los recursos separados del código en ficheros de recursos te resultará más fácil adaptar tu aplicación a las distintas configuraciones del dispositivo, tal y como idiomas diferentes o distintos tamaños de pantalla. Android utilizará automáticamente los recursos apropiados comparando la configuración actual del dispositivo con los nombres de las carpetas de recursos.

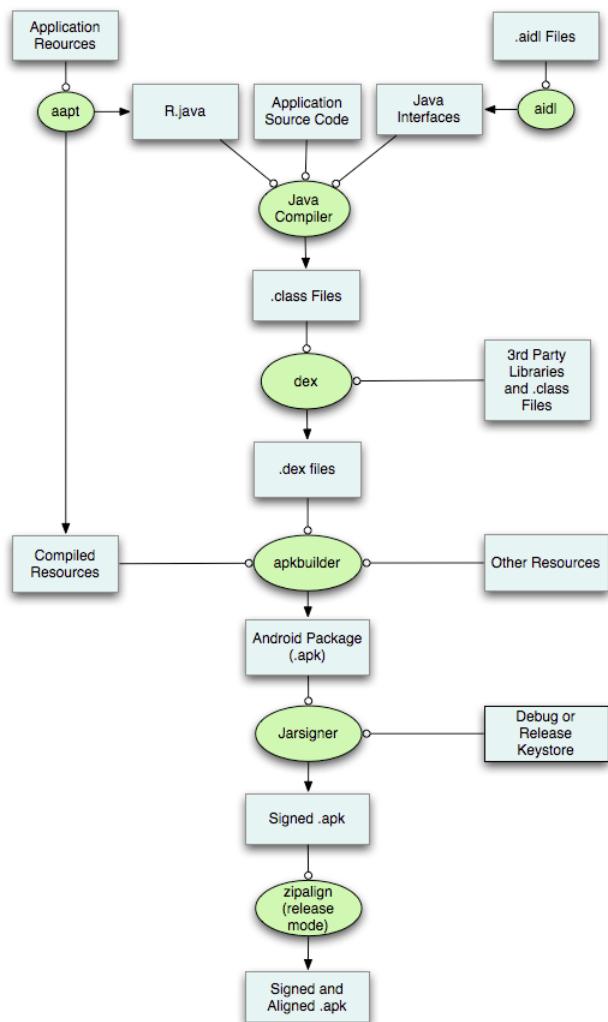
Deberías situar cada recurso en la carpeta adecuada como se explica a continuación:

- `anim/` para ficheros XML que especifican animaciones de interpolación.
- `color/` para especificaciones de color.
- `drawable/` para ficheros XML o bitmap.
- `layout/` para ficheros XML que especifican interfaces de usuario.
- `menu/` para ficheros XML que especifican la interfaz de los menús.
- `raw/` para cualquier fichero guardado en su formato original. Nosotros colocaremos aquí un archivo mp4 que servirá de fondo musical a nuestro juego.
- `values/` para ficheros que contienen cadenas, enteros, arrays, entre otros, con los siguientes convenios:
  - `arrays.xml` para recursos de tipo array
  - `colors.xml` para recursos que especifican colores
  - `dimens.xml` para recursos que especifican dimensiones
  - `strings.xml` para cadenas de caracteres
  - `styles.xml` para recursos que definen estilos
- `xml/` para ficheros XML en general que se pueden leer durante la ejecución de la aplicación.

## 5.4 La carpeta Gradle.scripts

Gradle es la herramienta que utiliza Android Studio para automatizar el proceso de construcción de las aplicaciones. El objetivo de este proceso es construir un fichero empaquetado (.apk) a partir de los recursos, fuentes java y bibliotecas. Este proceso está compuesto por varias etapas que se describen a continuación y en la figura inferior:

- La herramienta `aapt` compila los ficheros de recursos, incluido el de manifiesto. También genera el fichero `R.java` que como sabes contiene identificadores de recursos.
- Todo el código Java, incluido `R.java`, se compila para generar los ficheros `.class`.
- La herramienta `dex` convierte los ficheros `.class` y también las bibliotecas externas, en código byte Dalvik (ficheros `.dex`).
- La herramienta `apkbuilder` empaqueta en un fichero `.apk` los ficheros `.dex` junto con los recursos, tanto los compilados como los no compilados.
- El fichero `.apk` se ha de firmar en modo `debug` o `release`. En este segundo caso, a continuación se ha de alinear con la herramienta `zipalign`.



El entorno permite generar distintas variantes de la aplicación (`build variants`), cada una con su propio APK, dentro de un mismo proyecto. Cada variante es una combinación de un sabor (`product flavor`) y un tipo (`build type`). Dos sabores típicos podrían ser la versión demo y la versión de pago, cada uno con sus propios directorios `src` que pueden diferenciarse en algunos ficheros fuente. Además, Android dispone de los tipos `debug` y `release` por defecto. Esto da lugar a cuatro variantes cada una con su propio APK en la subcarpeta `app/build/outputs/apk/`.

La gestión de las variantes se automatiza con `Gradle` mediante los ficheros `build.gradle`. Existe un fichero `build.gradle` de alto nivel para el proyecto y otro por cada módulo. Este es el aspecto del fichero de construcción para el módulo `app` de nuestro proyecto `Hola`:

```

apply plugin: 'com.android.application'

android {
    compileSdkVersion 22
    buildToolsVersion "21.1.2"

    defaultConfig {
        applicationId "android.eps.uam.es.hola"
        minSdkVersion 15
        targetSdkVersion 22
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt')
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:22.2.0'
}

```

El significado de algunos de los principales elementos es el siguiente:

- `apply plugin` especifica que se debe utilizar el plugin de Android para Gradle.
- `android{}` especifica opciones de construcción específicas de Android como, por ejemplo, la propiedad `compileSdkVersion`. En su interior, el elemento `buildTypes` controla la construcción y empaquetado de la aplicación. Por omisión se generan dos tipos de construcción (`build types`): `debug` y `release`.
- `dependences{}` especifica las dependencias del módulo.
- `productFlavors{}` sirve para definir distintos sabores.

Con esto concluye nuestra rápida descripción de la estructura del primer proyecto en Android Studio. Tendrás ocasión de familiarizarte con ella de forma paulatina a medida que vayas trabajando.

En la siguiente unidad introduciremos el lenguaje XML, que va a ser fundamental para escribir nuestros recursos.

## 6. Una introducción a XML

XML utiliza los mismos bloques de construcción que HTML: elementos, atributos y valores:

Un **elemento** tiene una etiqueta de apertura consistente en un nombre entre los signos < y >, y termina con una etiqueta de cierre compuesta por el mismo nombre precedido por una barra inclinada entre los signos < y >:

```
<LinearLayout>
</LinearLayout>
```

Dentro de la etiqueta de apertura pueden aparecer **atributos**, a los que se dan valores delimitados entre dobles comillas. El siguiente elemento tiene 3 atributos:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_weight="1">
</LinearLayout>
```

Por ejemplo, `android:layout_width="match_parent"` especifica que el atributo `android:layout_width` (la anchura del elemento) vale `match_parent`, es decir, el elemento ocupa todo el ancho del contenedor dentro del cual se encuentra.

Un elemento puede contener otros elementos en su interior. Por ejemplo, el siguiente elemento `LinearLayout` contiene dos elementos `Button`:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_weight="1" >
    <Button
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:text="1"/>
    <Button
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:text="2"/>
</LinearLayout>
```

Todo documento XML tiene un elemento **raíz** que contiene a todos los demás. Fuera solo se pueden colocar comentarios e instrucciones de proceso. Estas últimas comienzan por <? y terminan con ?>. La siguiente instrucción de proceso declara la versión de xml y la codificación:

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

```

Aunque, como hemos dicho antes, cada elemento debe tener su etiqueta de cierre, aquellos elementos que no contengan otros en su interior, se pueden cerrar con />:

```
<Button  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:layout_weight="1" />
```

Quizá te preguntes cuál es el papel que van a jugar este tipo de ficheros en Android. Pues bien, básicamente, los ficheros XML se van a utilizar en Android para especificar interfaces de usuario y para almacenar recursos.

Las actividades de Android, que de momento puedes imaginar como pantallas en un dispositivo móvil, van a definirse mediante un fichero XML de diseño, donde se especifica la interfaz gráfica, y un fichero Java, donde se encuentra el código que, entre otras cosas, va a inflar las especificaciones del fichero de diseño XML. Veremos todo esto con detalle en la unidad 8.

En la siguiente unidad utilizaremos ficheros XML para almacenar recursos como, por ejemplo, cadenas de caracteres, colores o dimensiones. Al almacenar estos valores independientemente del código Java no solo garantizamos su unicidad sino que también permitimos su adaptación a cambios del dispositivo como, por ejemplo, el idioma o el tamaño de la pantalla.

## 7. Recursos

Como hemos visto brevemente en la unidad 4, los recursos son datos (cadenas, imágenes, ...) que se almacenan fuera del código de nuestra aplicación para mejorar su organización, garantizar su unicidad y permitir su adaptación a ciertas características del dispositivo como, por ejemplo, el idioma o el tamaño de la pantalla.

Los recursos se dividen en recursos del sistema y recursos de la aplicación. Los de la aplicación se crean y almacenan dentro de la carpeta `res`. Los recursos de la aplicación no pueden compartirse con otras aplicaciones.

A los **recursos de la aplicación** se puede acceder directamente desde otro fichero de recursos o desde una actividad utilizando código Java. Veamos la diferencia entre el acceso a un recurso desde un fichero de recursos o mediante código Java:

- Para acceder a un recurso desde otro fichero de recursos utilizaremos la notación:

```
@[tipo del recurso]/[nombre del recurso]
```

Por ejemplo, para acceder al recurso de nombre `hello` desde un fichero de diseño (`layout`), utilizaremos la siguiente expresión:

```
@string/hello
```

- Para acceder programáticamente al recurso `hello` desde una actividad hemos de utilizar la siguiente instrucción:

```
String string = getResources.getString(R.string.hello);
```

Los **recursos del sistema** se almacenan en el paquete `android.R`. En esta dirección <http://goo.gl/T1SuP> encontrarás una lista de todos los recursos del sistema. Por ejemplo, la clase `android.R.string` contiene los recursos de cadenas del sistema.

Para acceder a un recurso del sistema desde otro recurso hay que utilizar la notación:

```
@android:[tipo del recurso]/[nombre del recurso]
```

Por ejemplo, para acceder al recurso del sistema correspondiente a la cadena `ok`, utilizaremos la siguiente expresión:

```
@android:string/ok
```

Como ves, el acceso es muy parecido al descrito arriba para los recursos de la aplicación, solo debes añadir `android:` entre el símbolo `@` y el tipo de recurso.

## 7.1 Especificación de recursos

Vamos a crear dos pequeños proyectos que ilustran la definición y utilización de recursos de tipo cadena, dimensión y color. El primero determina el aspecto del texto de dos botones fijando sus propiedades en el fichero de diseño, mientras que el segundo proyecto lo hace programáticamente con código Java. Ambos utilizan recursos especificados en tres ficheros de recursos que se estudian a continuación.

Los ficheros de recursos contienen un elemento raíz de tipo `resources`. En el interior de este elemento se pueden añadir recursos mediante etiquetas adecuadas. Las etiquetas de las cadenas, colores y dimensiones son, respectivamente, `string`, `color` y `dimen`. Echemos un vistazo a los ficheros de recursos de nuestro proyecto (`strings.xml`, `colors.xml` y `dimens.xml`):

```
/res/values/strings.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Recursos</string>
    <string name="mensaje1">Grande en verde</string>
    <string name="mensaje2">Reducido en rojo</string>
</resources>
```

```
/res/values/colors.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="verde">#00FF00</color>
    <color name="rojo">#F00</color>
</resources>
```

```
/res/values/dimens.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="tipo_grande">20dp</dimen>
    <dimen name="tipo_reducido">10dp</dimen>
</resources>
```

Los colores en Android vienen representados como enteros en forma de 4 bytes: alfa, rojo, verde y azul (ARGB). Cada componente está comprendida entre 0 y 255. La primera mide la transparencia: 0 es totalmente transparente y 255 totalmente opaco. Para las demás, 0 significa que la componente no contribuye al color y 255 que contribuye al 100%. Por ejemplo, el azul opaco al 100% es `0xFF0000FF`, y el verde `0xFF00FF00`.

Veamos dos proyectos sencillos que utilizan estos recursos.

## Primer proyecto

Los recursos declarados en los ficheros XML de arriba se utilizan en el fichero de diseño `activity_main.xml` para especificar el tamaño y color del texto de los dos botones de la interfaz. Por ejemplo, el tamaño del texto del botón superior de la interfaz utiliza el recurso `@dimen/tipo_grande`, cuyo valor se asigna al atributo `textSize`:

```
android:textSize="@dimen/tipo_grande"
```

El fichero de diseño completo se muestra a continuación:

```
/res/layout/activity_main.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/mensaje1"
        android:textSize="@dimen/tipo_grande"
        android:textColor="@color/verde"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/mensaje2"
        android:textSize="@dimen/tipo_reducido"
        android:textColor="@color/rojo"/>
</LinearLayout>
```



El código de la actividad es muy sencillo y consiste en una llamada al método `setContentView()` dentro del método `onCreate()` de la actividad:

```
/src/MainActivity.java
```

```
package es.uam.eps.android.ccc7_1;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

## Segundo proyecto

En el siguiente proyecto, en lugar de fijar el valor de los atributos del botón en el fichero de diseño, lo hacemos con código Java mediante la clase `Resources`. Por simplicidad, solo utilizaremos el primer botón de los dos del proyecto anterior:

```
/res/layout/activity_main.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button1" />
</LinearLayout>
```

El método `getResources()` de la clase `Context` devuelve una referencia a un objeto de la clase `Resources`. Esta clase cuenta con métodos para acceder a los diferentes tipos de recursos de la aplicación:

- `getResources().getString(R.string.mensaje1)` para recuperar la cadena asociada al recurso con identificador `R.string.mensaje1`.
- `getResources().getDimension(R.dimen.tipo_grande)` para recuperar el número `float` asociado al recurso con identificador `R.dimen.tipo_grande`.
- `getResources().getColor(R.color.verde)` para recuperar el número `int` asociado al recurso con identificador `R.color.verde`.

Veamos cómo se fijan los atributos del texto del botón utilizando estos métodos de la clase `Resources`. Primero se recupera una referencia al botón mediante el método `findViewById()`, al que se pasa como argumento la referencia del botón, es decir, `R.id.button1`:

```
Button button1 = (Button) findViewById(R.id.button1);
```

A continuación se extrae la cadena correspondiente al recurso `R.string.mensaje1` y se asigna al texto del botón:

```
String mensaje1 = getResources().getString(R.string.mensaje1) ;
button1.setText(mensaje1);
```

De forma parecida se fijan el tamaño del texto y el color:

```
float dimension1 = getResources().getDimension(R.dimen.tipo_grande);
button1.setTextSize(dimension1);

int color1 = getResources().getColor(R.color.verde);
button1.setTextColor(color1);
```

El fichero Java al completo es el siguiente:

```
/src/MainActivity.java
```

```
package es.uam.eps.android.ccc7_2;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;

public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button button1 = (Button) findViewById(R.id.button1);

        String mensaje1 = getResources().getString(R.string.mensaje1) ;
        button1.setText(mensaje1);

        float dimension1 = getResources().getDimension(R.dimen.tipo_grande);
        button1.setTextSize(dimension1);

        int color1 = getResources().getColor(R.color.verde);
        button1.setTextColor(color1);
    }
}
```



## 7.2 Arrays de cadenas y de enteros

En el siguiente fichero definimos dos arrays, uno de cadenas y otro de enteros. El primero contiene el nombre de tres provincias y el segundo, sus distancias por carretera a Madrid. Estos arrays se declaran mediante elementos `<string-array>` e `<integer-array>`, respectivamente, dentro del fichero de recursos de nombre `arrays.xml`. El atributo `name` actúa como identificador del array, y los elementos se especifican mediante elementos `item` dentro del array:

```
/res/values/arrays.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="provincias">
        <item>Burgos</item>
        <item>Soria</item>
        <item>Valladolid</item>
    </string-array>
    <integer-array name="distancias">
        <item>238</item>
        <item>229</item>
        <item>211</item>
    </integer-array>
</resources>
```

A continuación vamos a escribir un proyecto sencillo para mostrar los elementos de estos arrays dentro de un `EditText`, primero el nombre de la provincia y, a su derecha, su distancia a Madrid.

El fichero de diseño de la actividad incluye una vista `TextView` dentro de un contenedor de tipo `LinearLayout`:

```
/res/layout/activity_main.xml
```

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

Para acceder programáticamente a un array de cadenas necesitamos un nuevo método de la clase `Resources`:

```
String[] provincias = getResources().getStringArray(R.array.provincias);
```

De forma similar accedemos al array de distancias:

```
int[] distancias = getResources().getIntArray(R.array.distancias);
```

Mediante un bucle `for` creamos una cadena (`mensaje`) con los pares ciudades/distancia, una línea por cada ciudad:

```
String mensaje = "";
for (int i=0; i<provincias.length; i++)
    mensaje += provincias[i] + " " + distancias[i] + "\n";
```

Finalmente, asignamos la cadena `mensaje` al `EditText` de la interfaz, todo ello dentro del método `onCreate()` de la actividad:

```
textView.setText(mensaje);
```

El código completo de la actividad es el siguiente, junto con el resultado gráfico:

```
/src/MainActivity.java
```

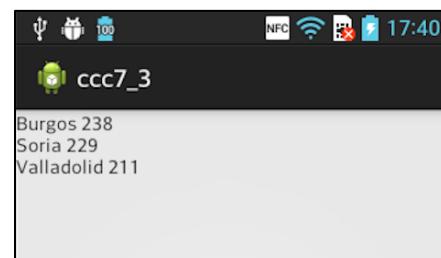
```
package es.uam.eps.android.ccc7_3;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView textView = (TextView) findViewById(R.id.textView);
        String[] provincias = getResources().getStringArray(R.array.provincias);
        int[] distancias = getResources().getIntArray(R.array.distancias);

        String mensaje = "";
        for (int i=0; i<provincias.length; i++)
            mensaje += provincias[i] + " " + distancias[i] + "\n";
```



```
        textView.setText(mensaje);  
    }  
}
```

# Jugando con Android

Aprende a programar tu primera App

Semana 2. Interfaz de usuario

## 8. La primera interfaz de usuario

Una aplicación Android es básicamente un conjunto de actividades. Cada actividad se implementa en una clase Java derivada de la clase base `Activity` de Android y, en general, mediante un fichero XML de diseño.

### 8.1 El fichero XML de diseño

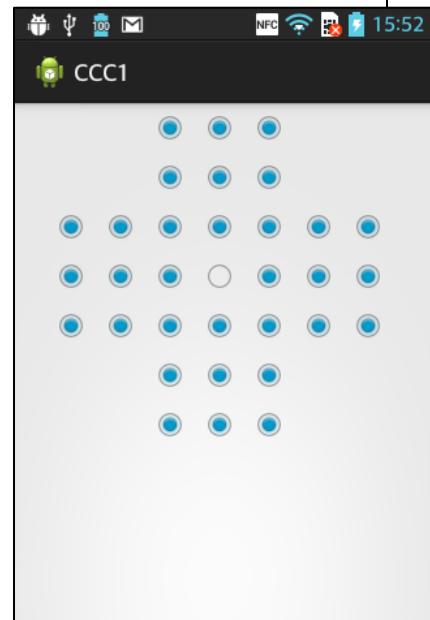
Las actividades interactúan con los usuarios a través de su interfaz de usuario. Esta interfaz se implementa con objetos de las clases `View` y `ViewGroup`, que podemos traducir como vistas y contenedores, respectivamente.

Por ejemplo, en la interfaz de nuestro juego vamos a utilizar vistas de tipo `RadioButton` y contenedores de tipo `LinearLayout`. Este tipo de contenedor organiza su contenido (los botones) en una sola fila o columna. Veremos mas adelante que existen otros contenedores como, por ejemplo, `TableLayout`, para organizar vistas de forma tabular.

La implementación de la interfaz de usuario puede hacerse en Java o mediante un fichero XML. En nuestro primer programa, la interfaz se especifica en el siguiente fichero XML:

```
/res/layout/activity_main.xml
```

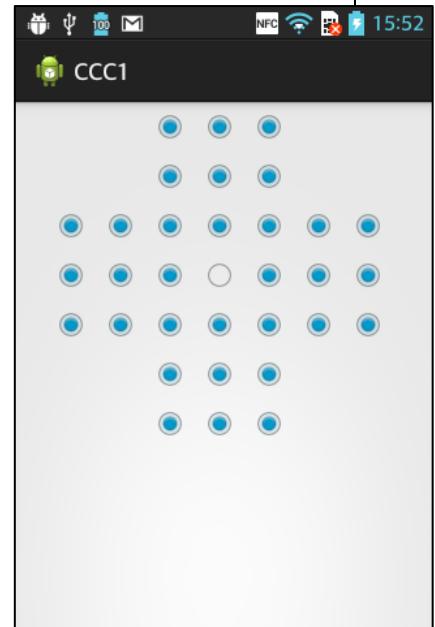
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center_horizontal">
    <LinearLayout
        android:layout_width="wrap_content"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



```

        android:layout_height="wrap_content" />
    </LinearLayout>
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >
        <RadioButton
            android:checked="true"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <RadioButton
            android:checked="true"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </LinearLayout>
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >
        <RadioButton
            android:checked="true"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <RadioButton
            android:checked="false"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <RadioButton
            android:checked="true"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <RadioButton
            android:checked="true"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <RadioButton
            android:checked="true"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </LinearLayout>
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >
        <RadioButton
            android:checked="true"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <RadioButton
            android:checked="true"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

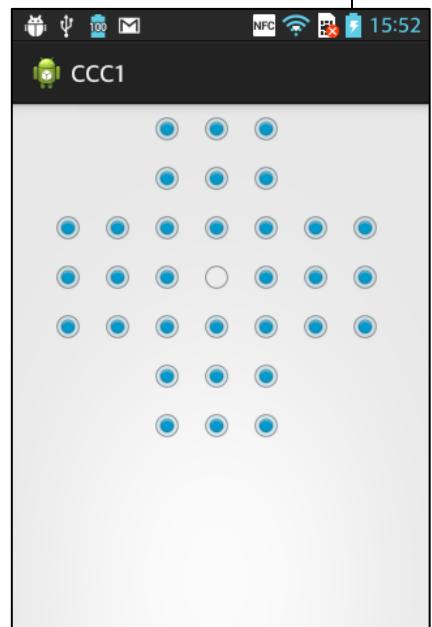
```



```

<RadioButton
    android:checked="true"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</LinearLayout>
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" >
    <RadioButton
        android:checked="true"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    </LinearLayout>
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <RadioButton
        android:checked="true"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    </LinearLayout>
</LinearLayout>

```



La etiqueta `<LinearLayout>` especifica un elemento de tipo `LinearLayout`. Este elemento es un contenedor que organiza su contenido en una sola fila o en una sola columna. Sus atributos se estudian a continuación.

El atributo `android:layout_width` especifica la anchura de la vista. Su valor puede ser una dimensión (tal y como "40dip") o una de las siguientes constantes:

- `fill_parent`: La vista será tan ancha como el padre, menos el padding del padre si es que tiene. El padding se especifica mediante el atributo `android:padding` como veremos mas adelante.
- `match_parent`: La vista será tan ancha como el padre, menos el padding del padre si es que tiene. Este valor remplaza a `fill_parent` desde el nivel API 8.

- `wrap_content`: La vista será lo suficientemente ancha para contener su propio contenido más padding.

El atributo `android:layout_height` especifica la altura de la vista. Su valor puede ser una dimensión (tal y como "40dip") o una de las constantes de la tabla de arriba (donde se ha de sustituir anchura por altura).

Los atributos `android:layout_width` y `android:layout_height` se utilizan tanto en los contenedores `LinearLayout` como en las vistas `RadioButton`. Todos los elementos utilizan el valor `wrap_content` salvo el contenedor principal, que utiliza `match_parent`.

El atributo `android:orientation` tiene el valor `vertical`, es decir, los hijos se organizan en una columna. En este caso, el elemento `LinearLayout` principal contiene siete hijos de tipo `LinearLayout` dispuestos verticalmente, uno por cada fila del tablero de nuestro juego. A su vez, cada uno de estos hijos contiene botones en fila, ya que la orientación por defecto es `horizontal`. El resultado final es siete filas de botones, con tres o siete botones por fila dependiendo de la posición.

El atributo `android:gravity` del contenedor principal tiene el valor `center_horizontal`, lo cual garantiza que sus hijos, es decir, cada una de las filas del tablero, quedarán centradas en la pantalla.

La etiqueta `<RadioButton>` especifica un elemento de tipo `RadioButton`, que puede ser pulsado por el usuario. Estos elementos van a jugar el papel de las fichas de nuestro juego. El atributo `android:checked` especifica el estado inicial del botón, que puede ser pulsado (`true`) o no (`false`). El único botón sin pulsar es el que ocupa la posición central del tablero para simular la ausencia de ficha en esa posición al iniciar el juego.

Esto completa el análisis del fichero de diseño. Echemos un vistazo al fichero Java.

## 8.2 El fichero Java

El fichero `MainActivity.java` contiene la definición de la clase `MainActivity` que extiende la clase `Activity` de Android. En nuestro caso, solo se sobrescribe el método `onCreate()` de `Activity`. Este método es uno de los métodos del ciclo de vida de la actividad. No se llama explícitamente, como todos los métodos que empiezan por `on`, sino que se ejecuta automáticamente en un cierto momento de la vida de la actividad.

Concretamente, `onCreate()` se ejecuta una vez instanciada la actividad pero cuando todavía no se ha mostrado en la pantalla. El código es el siguiente:

```
/src/MainActivity.java
```

```
package es.uam.eps.android.ccc8;

import android.os.Bundle;
import android.app.Activity;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

La anotación `@Override` hace que el compilador compruebe que el método que se está sobrescribiendo efectivamente existe en la superclase. De esta forma se evitan errores como el que ocurre si intentamos implementar el método:

```
protected void onCreates(Bundle savedInstanceState) {
    ...
}
```

Dentro del método, lo primero que hacemos es llamar a la versión de la superclase, lo cual es obligatorio. Además, ha de ser la primera instrucción del método sobrescrito:

```
super.onCreate(savedInstanceState);
```

Como ves, se pasa el objeto recibido de tipo `Bundle` al método de la superclase. El objeto de tipo `Bundle` contiene el estado de las vistas de la interfaz almacenado en forma de pares clave-valor. Este objeto sirve para que la actividad guarde y posteriormente recreé su interfaz gráfica. Como veremos, el programador puede añadir información a este objeto en forma de pares clave-valor extra, con el objetivo de restaurar información adicional cuando se infla la interfaz.

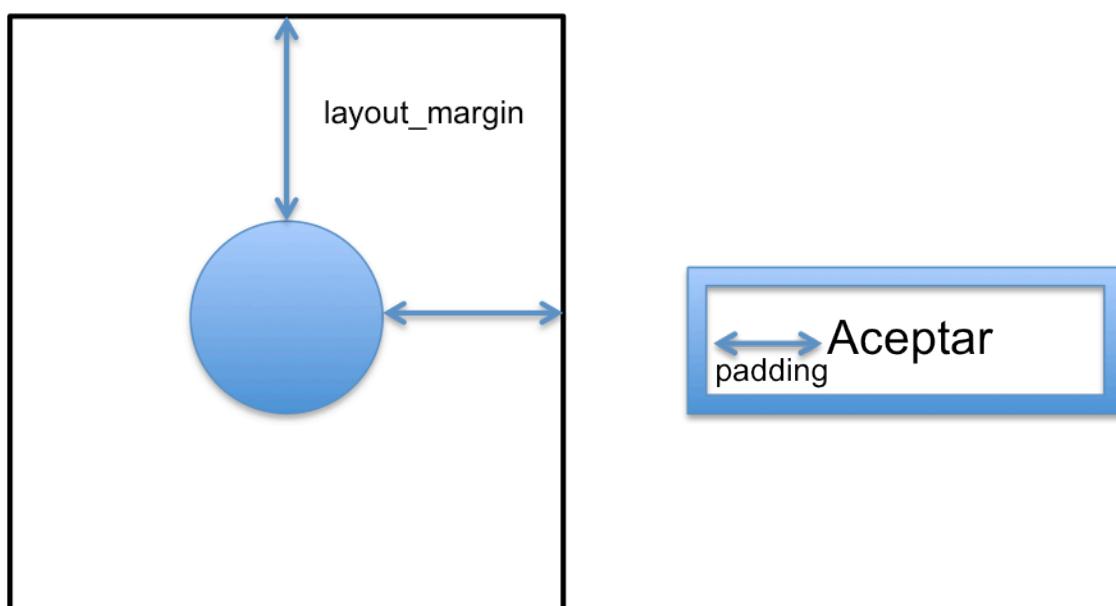
A continuación llamamos al método `setContentView()`, que infla la interfaz especificada por su argumento, que es un identificador de recurso. El identificador `R.layout.activity_main` representa el fichero `activity_main.xml` que acabamos de estudiar en la sección anterior.

Recuerda que la clase `R` es una clase constante generada automáticamente y que contiene valores enteros organizados en subclases como, por ejemplo, `id` y `layout`. Así, dentro de `layout` se sitúan los identificadores de los ficheros XML de diseño como `activity_main.xml` y, en el interior de `id`, se encuentran los identificadores de las vistas de la interfaz.

## 9. Márgenes y espaciado

En Android, el espaciado se especifica mediante los atributos `padding` y `layout_margin`. El atributo `layout_margin` espacia la vista con respecto a su contenedor u otras vistas, mientras que `padding` espacia el contenido de una vista respecto a los bordes de la vista.

Dicho de otra forma, `layout_margin` especifica el espaciado fuera de los bordes de la vista mientras que `padding` lo hace dentro de los bordes de la vista, como se muestra en el siguiente diagrama:



Se puede ser mas específico aún:

- `android:padding`: especifica el espacio vacío entre el contenido de un elemento y sus cuatro lados.
- `android:paddingTop`: especifica el espacio vacío entre el contenido de un elemento y su lado superior.
- `android:paddingBottom`: especifica el espacio vacío entre el contenido de un elemento y su lado inferior.
- `android:paddingLeft`: especifica el espacio vacío entre el contenido de un elemento y su lado izquierdo.
- `android:paddingRight`: especifica el espacio vacío entre el contenido de un elemento y su lado derecho.

El espacio entre una vista y su contenedor se especifica mediante los siguientes atributos:

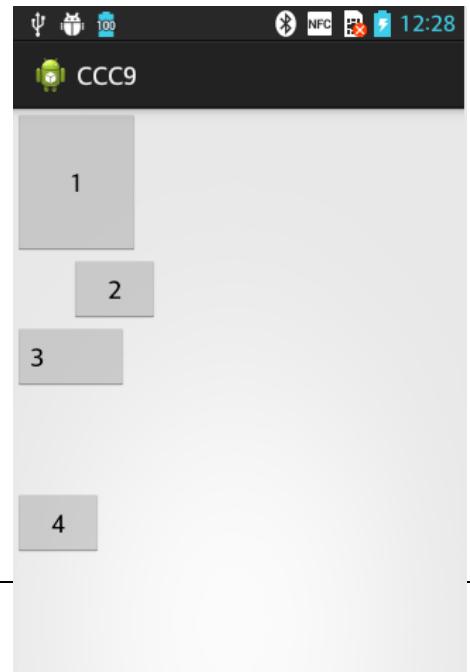
- `android:layout_margin`: especifica el espacio entre una vista y las vistas o el contenedor por sus cuatro lados.

- `android:layout_marginTop`: especifica el espacio entre el lado superior de una vista y otro elemento o el contenedor.
- `android:layout_marginBottom`: especifica el espacio entre el lado inferior de una vista y otro elemento o el contenedor.
- `android:layout_marginLeft`: especifica el espacio entre el lado izquierdo de una vista y otro elemento o el contenedor.
- `android:layout_marginRight`: especifica el espacio entre el lado derecho de una vista y otra vista o el contenedor.

El siguiente ejemplo ilustra estas diferencias:

/res/layout/activity\_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:text="1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="40dip"/>
    <Button
        android:text="2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="40dip"/>
    <Button
        android:text="3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingRight="60dip"/>
    <Button
        android:text="4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="70dip"/>
</LinearLayout>
```



El fichero Java es elemental pues solo se sobrescribe el método `onCreate()` para inflar la interfaz especificada en `activity_main.xml`:

/src/MainActivity.java

```
package es.uam.eps.dadm.ccc9;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

## 10. Gravedad

Los atributos de gravedad permiten controlar la alineación del contenido de las vistas (`gravity`), así como la alineación de las vistas dentro de sus contenedores (`layout_gravity`):

- `android:gravity`, en un contenedor tal y como `LinearLayout`, alinea los elementos situados dentro del contenedor según el valor asignado (derecha, izquierda, ...). Dentro de una vista como `Button`, alinea el texto dentro del botón.
- `android:layout_gravity` en una vista como `Button`, por ejemplo, alinea el botón dentro del contenedor en el que se encuentra, según el valor asignado.

Los valores que pueden tomar estos atributos son los siguientes:

- `top`
- `bottom`
- `left`
- `right`
- `center_vertical`
- `fill_vertical`
- `center_horizontal`
- `fill_horizontal`
- `center`
- `fill`
- `clip_vertical`
- `clip_horizontal`
- `start`
- `end`

A continuación se utilizan cuatro proyectos para ilustrar el significado de algunos de estos valores. Todos los proyectos comparten el fichero Java salvo por el nombre del paquete. Este fichero se encarga de inflar la interfaz gráfica especificada en el fichero de diseño `activity_main.xml`:

/src/MainActivity.java

```
package es.uam.eps.dadm.ccc10_1;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

En el siguiente proyecto (ccc10\_1) utilizamos el atributo `gravity` para alinear el texto de los botones: Burgos queda alineado a la izquierda y Madrid a la derecha. Fíjate en que el atributo `android:layout_width` de los botones debe tener el valor `match_parent` para conseguir el efecto deseado (utilizamos por claridad una cadena en lugar de un recurso en el valor del atributo `android:text`):

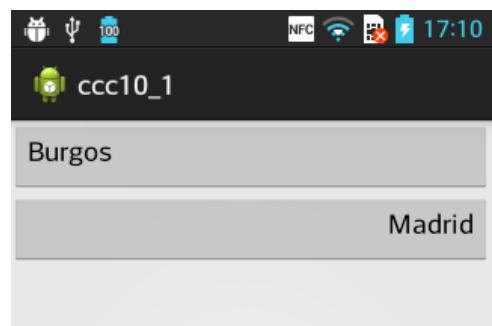
```
/res/layout/activity_main.xml
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="left"
        android:text="Burgos" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="right"
        android:text="Madrid" />

</LinearLayout>
```



En el siguiente proyecto (ccc10\_2) son los botones, no su contenido, los que se alinean a la izquierda y derecha dentro del contenedor `LinearLayout` gracias al atributo `android:layout_gravity`. El atributo `android:layout_width` de los botones se iguala a `wrap_content` en esta ocasión (comprueba lo que ocurre si dejas el valor `match_parent`):

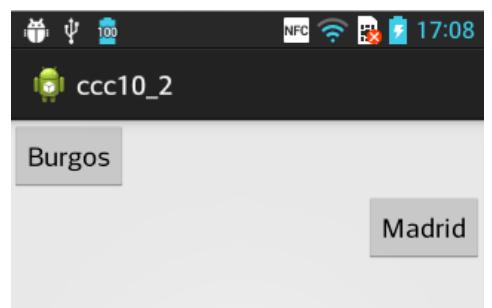
```
/res/layout/activity_main.xml
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="left"
        android:text="Burgos" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="Madrid" />

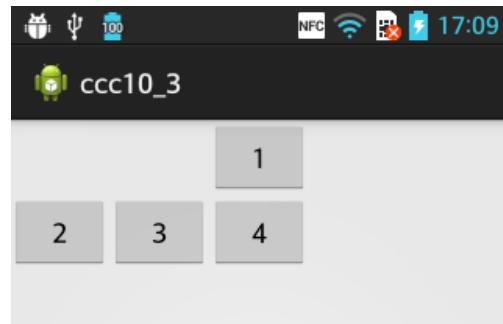
</LinearLayout>
```



El valor `android:layout_gravity="center_horizontal"` del botón número 1 del ejemplo ccc10\_3 hace que el botón quede centrado dentro del `LinearLayout` que lo contiene. El `LinearLayout` interior con `android:layout_gravity="left"` queda alineado a la izquierda del `LinearLayout` principal, con lo que el botón número 2 aparece a la izquierda del todo:

```
/res/layout/activity_main.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="1" />
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_gravity="left">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="2" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="3" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="4"/>
    </LinearLayout>
</LinearLayout>
```



En el último ejemplo (ccc10\_4), el valor `android:gravity="center_horizontal"` del `LinearLayout` principal fuerza a que tanto el botón número 1 como el `LinearLayout` interior, que contiene a los botones 2, 3 y 4, queden centrados horizontalmente:

```
/res/layout/activity_main.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center_horizontal">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="1" />
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="2" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="3" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="4" />
    </LinearLayout>
</LinearLayout>
```



## 11. Pesos

El atributo `android:layout_weight` reparte el espacio sobrante entre las vistas de un contenedor. Utilizado adecuadamente, constituye una forma sencilla y elegante de adaptar la interfaz a distintos tamaños de pantalla. El valor por omisión de `android:layout_weight` es 0, lo cual indica que el espacio sobrante no se debe utilizar. Para que entiendas su significado vamos a crear un proyecto sencillo, `ccc11`, con cuatro interfaces de usuario diferentes (`activity_main_i.xml`, donde `i` va de 1 a 4).

Empecemos con una interfaz que muestra tres botones dispuestos horizontalmente dentro de un contenedor `LinearLayout` sin repartir el espacio sobrante, marcado en la figura con una línea horizontal:

```
/res/layout/activity_main_2.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="0"
        android:text="w=0" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="0"
        android:text="w=0" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="0"
        android:text="w=0" />
</LinearLayout>
```



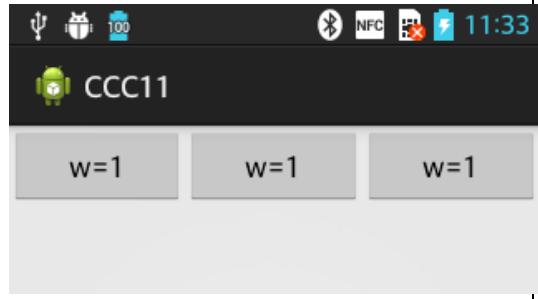
El contenedor `LinearLayout` hace dos pasadas antes de asignar la anchura de sus vistas en pantalla. Primero utiliza el atributo `layout_width`. En este caso, las vistas reciben espacio suficiente para su contenido, pues el valor asignado a este atributo es `wrap_content`. En el segundo paso, el contenedor utiliza la información del atributo `layout_weight` para repartir el espacio sobrante. Como en este caso todos los pesos valen 0, este espacio sobrante no se utiliza.

Si lo que se quiere es que el contenedor reparta el espacio de una sola vez, basándose exclusivamente en los pesos, basta con igualar la anchura de las vistas a 0 (`android:layout_width="0dp"`).

En el siguiente ejemplo, la asignación `android:layout_weight="1"` en todos los botones garantiza que el espacio sobrante se repartirá uniformemente, es decir, cada botón recibirá 1/3 del espacio sobrante. Cualquier otro valor entero o real surtirá el mismo efecto (1.0, 8, ...), con tal de ser el mismo para las tres vistas. La interfaz de usuario es la siguiente:

```
/res/layout/activity_main_2.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="w=1" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="w=1" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="w=1" />
</LinearLayout>
```



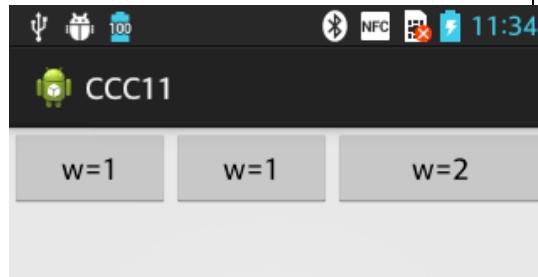
Si los dos primeros botones tienen un peso igual a 1 y el tercero igual a 2, el espacio sobrante se repartirá como  $\frac{1}{4}$ ,  $\frac{1}{4}$  y  $\frac{1}{2}$ , respectivamente. Lo mismo se consigue asignando los pesos de la siguiente manera: (25, 25, 50). En general, ajustando el valor de los pesos,  $w_i$ , conseguimos que los elementos del contenedor llenen el espacio libre en su interior de acuerdo con la siguiente fórmula:

$$\text{espacio sobrante}_i = \frac{w_i}{\sum_j w_j}$$

donde  $w_i$  es el peso del hijo número  $i$ , y el denominador es la suma de los pesos de cada hijo del contenedor. El resultado de esta ecuación es el espacio sobrante que se concede al hijo número  $i$ . De acuerdo con esta fórmula, si los pesos valen 0, no se reparte el espacio sobrante. Si todos los pesos se igualan a 1, el espacio sobrante se reparte uniformemente,  $1/3$  para cada botón. Los siguientes ejemplos corresponden a las distribuciones  $(\frac{1}{4}, \frac{1}{4}, \frac{1}{2})$  y  $(0, \frac{1}{2}, \frac{1}{2})$ , respectivamente:

```
/res/layout/activity_main_3.xml
```

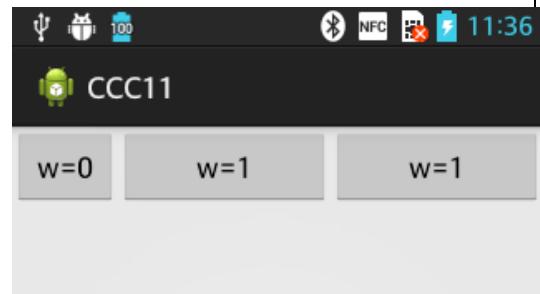
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="w=1" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="w=1" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:text="w=2" />
</LinearLayout>
```



```
</LinearLayout>
```

```
/res/layout/activity_main_4.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="0"
        android:text="w=0" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="w=1" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="w=1" />
</LinearLayout>
```



El fichero Java para el proyecto de esta unidad es sumamente sencillo, reduciéndose a inflar la interfaz correspondiente a cada uno de los ficheros de diseño:

```
/src/MainActivity.java
```

```
package es.uam.eps.dadm.ccc11;

import android.app.Activity;
import android.os.Bundle;

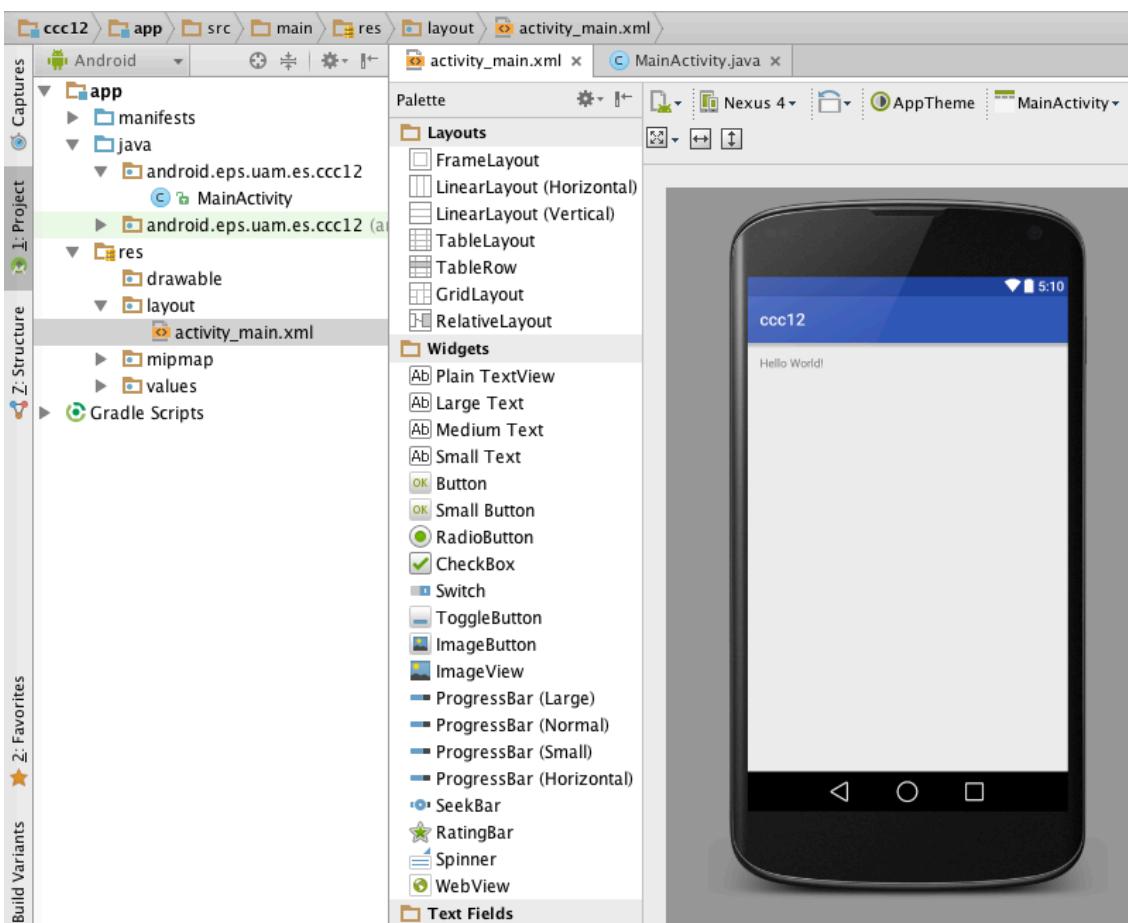
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main_1);
    }
}
```

## 12. Otras vistas

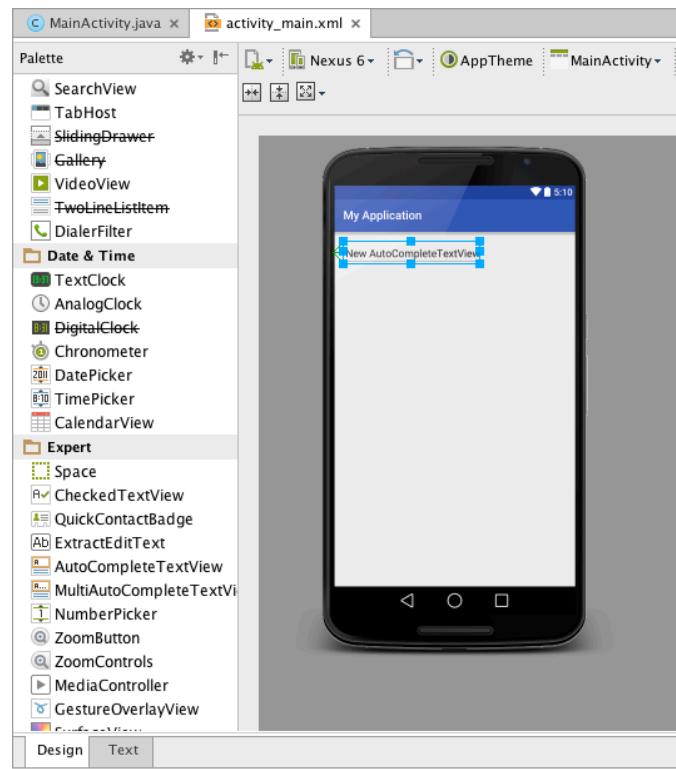
En las unidades anteriores hemos utilizado algunas vistas de Android como `Button` y `RadioButton`. Como puedes imaginar existen otras muchas vistas en Android, algunas de las cuales utilizaremos a lo largo del desarrollo de nuestro proyecto. En esta unidad vamos a ver una forma sencilla de experimentar con otras vistas en Android Studio.

Para ello crea un nuevo proyecto de nombre `ccc12` y selecciona el fichero de diseño `activity_main.xml` de la carpeta `res` de la vista Android:

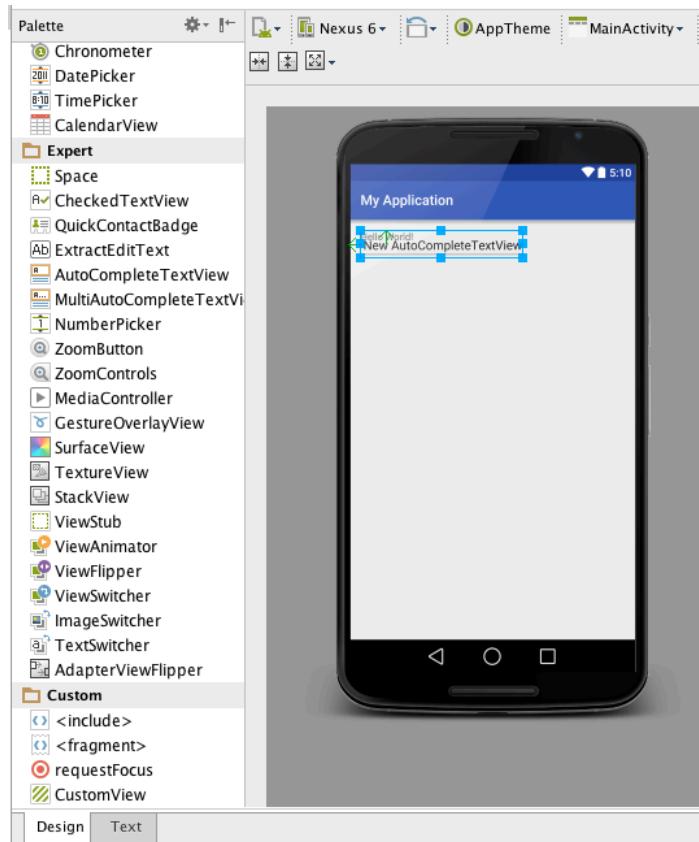


En la columna de nombre `Palette` tienes clasificadas las vistas y contenedores por categorías. Por ejemplo, dentro de la carpeta `Expert`, (tendrás que desplazarte hacia abajo en la columna `Palette`) nos encontramos con elementos como `AutoCompleteTextView`, que es una vista de texto que muestra sugerencias a medida que el usuario escribe en ella.

Antes de añadir gráficamente este elemento a la interfaz, vamos a eliminar la vista de tipo `TextView` que ha generado el entorno automáticamente. Para ello pulsa con el botón derecho del ratón y selecciona `Delete`:



A continuación, añade la nueva vista arrastrándola y dejándola caer sobre la pantalla del dispositivo. El resultado es el siguiente:



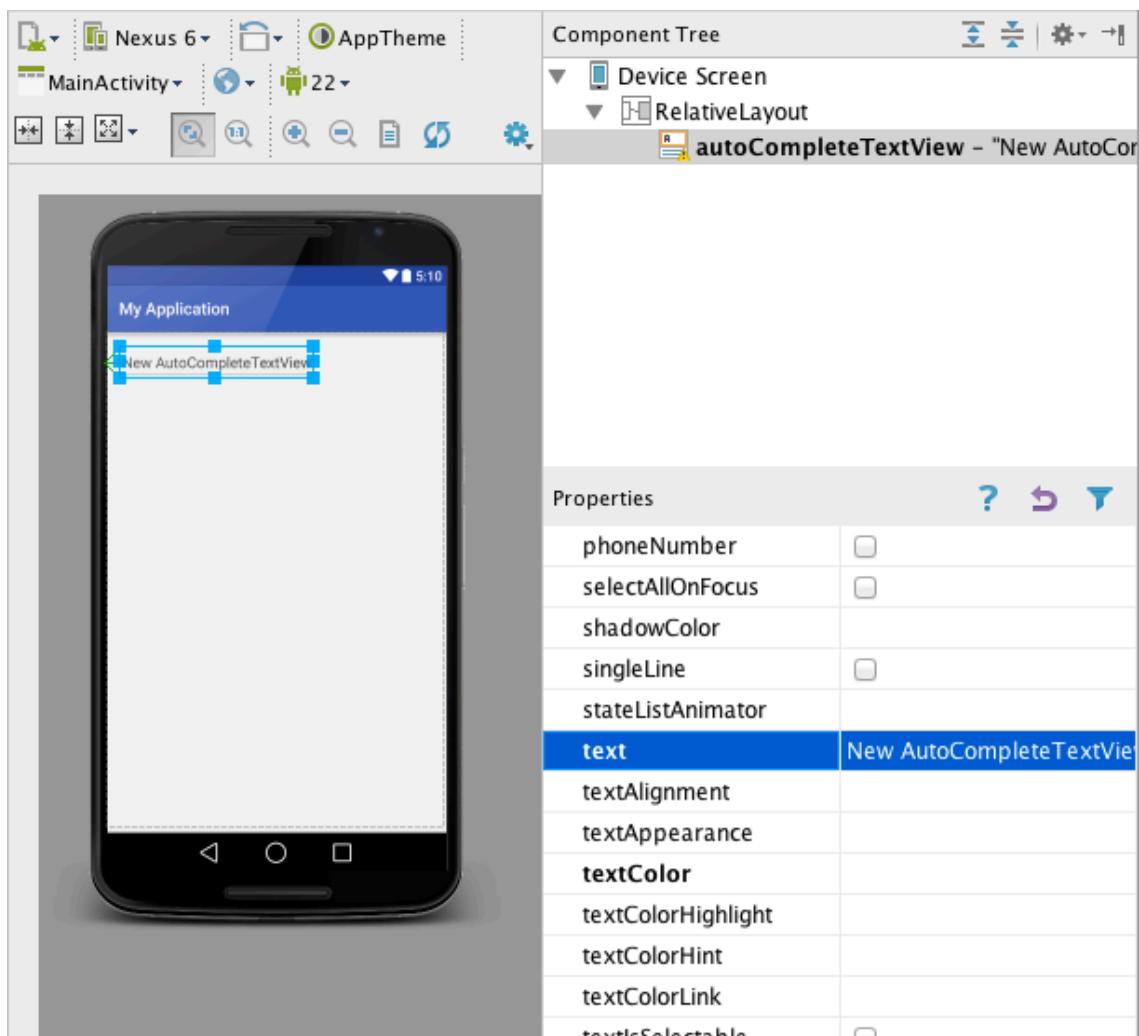
Android Studio añade automáticamente el elemento a nuestro fichero de diseño XML:

```
/res/layout/activity_main.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <AutoCompleteTextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New AutoCompleteTextView"
        android:id="@+id/autoCompleteTextView"
        android:layout_alignTop="@+id/textView"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />
</RelativeLayout>
```

Puedes consultar y actualizar las propiedades de la vista en la ventana de la derecha de nombre **Properties**:



Por ejemplo, puedes eliminar el valor que el atributo `android:text` tiene por defecto marcando y borrando el texto a la derecha de la propiedad `text`.

Las sugerencias que propone la vista a medida que se escribe se han de suministrar en el código como muestra la actividad `MainActivity.java`:

```
/src/MainActivity.java
```

```
package es.uam.eps.dadm.ccc12;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;

public class MainActivity extends Activity {
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.activity_main);

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_dropdown_item_1line, CIUDADES);

        AutoCompleteTextView textView = (AutoCompleteTextView)
            findViewById(R.id.autoCompleteTextView);

        textView.setAdapter(adapter);
    }

    private static final String[] CIUDADES = new String[] {
        "Burgos", "Soria", "Barcelona", "Sevilla", "Santander"
    };
}
```

Las sugerencias están contenidas en el array `CIUDADES` que se pasa como tercer argumento al constructor del adaptador de arrays de nombre `adapter`:

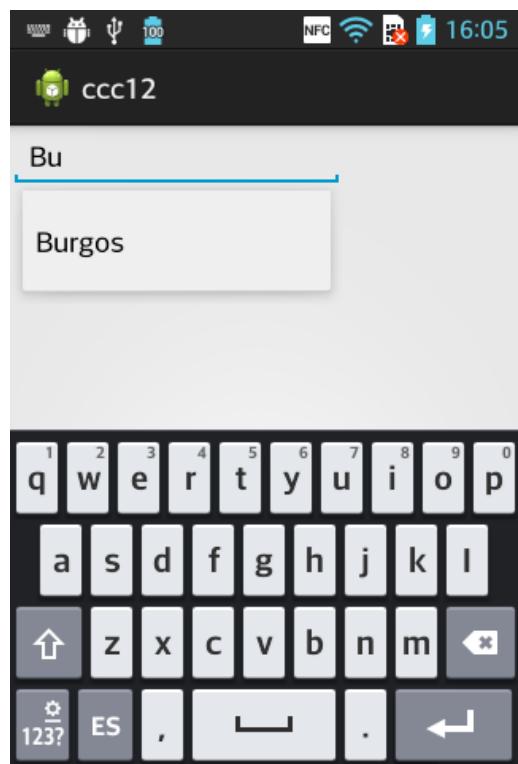
```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_dropdown_item_1line, CIUDADES);
```

Un adaptador es una especie de intermediario entre unos datos que deseamos mostrar y la vista que los ha de representar. El primer argumento del constructor es el contexto, la actividad en este caso. El segundo es el identificador de recurso del fichero de diseño en el que mostrar los datos. En este caso utilizamos un diseño predefinido consistente en un `TextView`. Podemos suministrar otro fichero de diseño siempre que tenga como raíz un elemento de tipo `TextView`.

El adaptador se ajusta al objeto de tipo `AutoCompleteTextView` mediante la llamada al método `setAdapter()`:

```
textView.setAdapter(adapter);
```

La actividad resultante mejora la experiencia del usuario frente a un simple elemento `EditText` gracias a las sugerencias:



## 13. RelativeLayout

RelativeLayout es un contenedor que organiza sus elementos en posiciones relativas. Por ejemplo, un elemento se puede colocar a la izquierda o debajo de otro, o pegado a un lado del contenedor o alineado con otro elemento. Se utiliza para evitar anidar contenedores, lo cual es costoso desde el punto de vista computacional.

Para utilizar este contenedor hemos de identificar los elementos de la interfaz mediante el atributo `android:id`. Estos identificadores se asocian a los elementos de la interfaz y nos permiten acceder a los botones desde el fichero Java. La sintaxis es `android:id="@+id/f2"`, donde `f2` es un identificador válido elegido por el programador y el signo `+` indica que `f2` es un nuevo identificador que se añade automáticamente al fichero `R.java`.

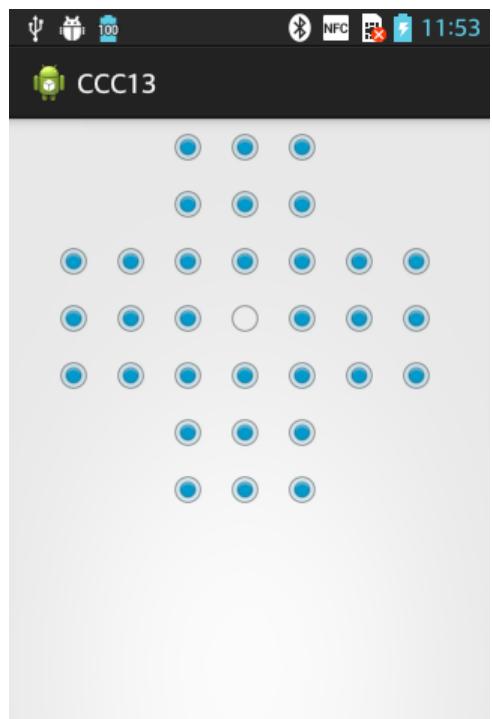
Como se puede comprobar a continuación, la clase `R` está formada por subclases, una de las cuales se denomina `id` y contiene los identificadores enteros como miembros públicos, estáticos y finales. Esto impide alterar el valor del identificador y, a la vez, permite su acceso como `R.id.f2`:

/gen/R.java

```
public final class R {
    public static final class attr {
    }
    public static final class dimen {
        public static final int activity_horizontal_margin=0x7f040000;
        public static final int activity_vertical_margin=0x7f040001;
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int action_settings=0x7f080021;
        public static final int f1=0x7f080001;
        public static final int f10=0x7f080006;
        public static final int f11=0x7f08000a;
        public static final int f12=0x7f08000b;
        public static final int f13=0x7f08000c;
        public static final int f14=0x7f08000d;
        public static final int f15=0x7f08000e;
        public static final int f16=0x7f08000f;
        public static final int f17=0x7f080010;
        public static final int f18=0x7f080011;
        public static final int f19=0x7f080012;
        public static final int f2=0x7f080000;
        public static final int f20=0x7f080013;
        public static final int f21=0x7f080014;
        public static final int f22=0x7f080015;
        public static final int f23=0x7f080016;
        public static final int f24=0x7f080017;
        public static final int f25=0x7f080018;
        public static final int f26=0x7f080019;
        public static final int f27=0x7f08001a;
        ...
    }
    public static final class layout {
        public static final int activity_main=0x7f030000;
    }
    ...
}
```

Veamos cómo reconstruir la interfaz de nuestro juego con `RelativeLayout` en lugar de `LinearLayout`. El código es bastante extenso, así que solo mostramos una parte a continuación, la correspondiente a las dos primeras filas del tablero:

```
/res/layout/activity_main.xml
```



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <RadioButton
        android:id="@+id/f2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:layout_centerHorizontal="true"/>
    <RadioButton
        android:id="@+id/f1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:layout_toLeftOf="@+id/f2"/>
    <RadioButton
        android:id="@+id/f3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:layout_toRightOf="@+id/f2"/>
    <RadioButton
        android:id="@+id/f4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:layout_below="@+id/f1"
        android:layout_alignLeft="@+id/f1"/>
    <RadioButton
        android:id="@+id/f5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:layout_below="@+id/f1"
        android:layout_toRightOf="@+id/f4"/>
    <RadioButton
        android:id="@+id/f6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:layout_below="@+id/f1"
        android:layout_toRightOf="@+id/f5"/>
    ...
</RelativeLayout>
```

El botón identificado como `f2` se centra horizontalmente:

```
<RadioButton
    android:id="@+id/f2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:layout_centerHorizontal="true"/>
```

El botón identificado como `f1` se sitúa a su izquierda:

```
<RadioButton
    android:id="@+id/f1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:layout_toLeftOf="@+id/f2"/>
```

Finalmente, el botón identificado como `f3` se sitúa la derecha del botón `f2`:

```
<RadioButton  
    android:id="@+id/f3"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:checked="true"  
    android:layout_toRightOf="@+id/f2"/>
```

Encontrarás el código completo del fichero de diseño en la carpeta `/res/layout/` del proyecto `ccc13` que acompaña a esta unidad. Aunque el aspecto final del tablero es indistinguible del de la unidad 8, esta versión del diseño es preferible pues hemos evitado anidar contenedores, lo cual hace que la interfaz se dibuje más rápidamente.

A continuación encontrarás la lista completa de atributos XML correspondientes a la clase `RelativeLayout.LayoutParams` y organizada por categorías:

Atributos utilizados para especificar la posición de una vista con respecto a su contenedor:

- `android:layout_alignParentTop`: si `true`, hace que el lado superior de la vista se alinee con el lado superior del contenedor.
- `android:layout_alignParentBottom`: si `true`, hace que el lado inferior de la vista se alinee con el lado inferior del contenedor.
- `android:layout_alignParentLeft`: si `true`, hace que el lado izquierdo de la vista se alinee con el lado izquierdo del contenedor.
- `android:layout_alignParentRight`: si `true`, hace que el lado derecho de la vista se alinee con el lado derecho del contenedor.
- `android:layout_centerHorizontal`: si `true`, centra la vista horizontalmente dentro del contenedor.
- `android:layout_centerVertical`: si `true`, centra la vista verticalmente dentro del contenedor.
- `android:layout_centerInParent`: si `true`, centra la vista horizontal y verticalmente dentro del contenedor.

Atributos utilizados para controlar la posición de una vista respecto a la de otra (el identificador de la segunda vista se especifica como en el siguiente ejemplo: `android:layout_toRightOf="@+id/f2"`):

- `android:layout_above`: coloca el lado inferior de la vista por encima de la vista especificada.
- `android:layout_below`: coloca el lado superior de la vista por debajo de la vista especificada.
- `android:layout_toLeftOf`: coloca el lado derecho de la vista a la izquierda de la vista especificada.
- `android:layout_toRightOf`: coloca el lado izquierdo de la vista a la derecha de la vista especificada.
- `android:layout_alignWithParentIfMissing`: si `true`, el padre se utilizará como referencia si la referencia a la segunda vista no se puede encontrar para `layout_toLeftOf`, `layout_toRightOf`, ...

Atributos utilizados para alinear una vista con otra (el identificador de la segunda vista se especifica como en este ejemplo: `android:layout_alignRight="@+id/f1"`):

- `android:layout_alignBaseline`: la línea base de las dos vistas se alinea.
- `android:layout_alignBottom`: el lado inferior de las dos vistas se alinea.
- `android:layout_alignTop`: el lado superior de las dos vistas se alinea.
- `android:layout_alignLeft`: el lado izquierdo de las dos vistas se alinea.
- `android:layout_alignRight`: el lado derecho de las dos vistas se alinea.

## 14. Otros contenedores

Los contenedores determinan la estructura visual de las actividades pues cada uno tiene una forma concreta de colocar en su interior a las vistas que lo componen.

Los contenedores extienden directa o indirectamente la clase `ViewGroup`, que es una vista especial (`ViewGroup` extiende `View`) que puede contener a otras en su interior. Estos son algunos de los contenedores que tienes a tu disposición en Android:

- `AbsoluteLayout`: cada hijo es asignado a una posición fija dentro del contenedor.
- `AdapterView`: los hijos que pueblan este contenedor se obtienen dinámicamente de un adaptador. Por ejemplo, `ListView` muestra a sus hijos en una lista vertical desplazable.
- `DrawerLayout`: permite extraer a sus hijos (cajones) desde los lados de la ventana.
- `FrameLayout`: sirve para reservar una zona de la pantalla para un solo hijo.
- `GridLayout`: organiza a sus hijos dentro de una red rectangular de forma flexible, es decir, sin necesidad de especificar elementos intermedios como filas, por ejemplo.
- `LinearLayout`: organiza a sus hijos en una sola fila o columna.
- `RelativeLayout`: la posición de los hijos se especifica en relación a la de los demás o la del contenedor.
- `ScrollView`: permite desplazar (*scroll*) a su único hijo, que suele ser otro contenedor, y así poder mostrar información que no cabe en una sola pantalla.
- `TableLayout`: organiza a sus hijos en una tabla mediante elementos de tipo `TableRow`.
- `ViewPager`: organiza páginas de datos entre las que nos podemos desplazar a la izquierda o a la derecha.
- `WebView`: permite mostrar páginas web.

Aunque no es muy frecuente, dada la gran variedad de contenedores que existen, también tienes la posibilidad de desarrollar tus propios contenedores extendiendo la clase `ViewGroup`.

Esta unidad viene acompañada por cinco proyectos (`ccc14_1`, ..., `ccc14_5`) que ilustran algunos de estos contenedores. Solo el segundo (`ccc14_2`) contiene código Java no trivial. El resto de los proyectos cuentan con un fichero Java como los utilizados hasta ahora, que básicamente infla la interfaz especificada en el fichero de diseño correspondiente (`activity_main.xml`).

## 14.1 AbsoluteLayout

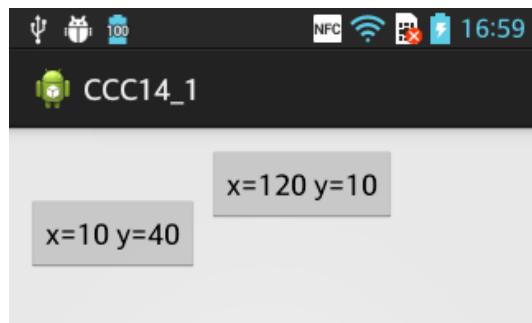
Los elementos de este contenedor se sitúan especificando sus posiciones X e Y exactas mediante los atributos `android:layout_x` y `android:layout_y`, respectivamente:

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_x="10dip"  
    android:layout_y="40dip"/>
```

Se trata de un contenedor no recomendable pues las interfaces no se adaptan adecuadamente al cambiar la resolución o el tamaño del dispositivo. Veamos un ejemplo sencillo:

/res/layout/activity\_main.xml

```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context=".Main" >  
  
    <Button  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_x="10dip"  
        android:layout_y="40dip"  
        android:text="@string/x_10_y_40" />  
  
    <Button  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_x="120dip"  
        android:layout_y="10dip"  
        android:text="@string/x_120_y_10" />  
  
</AbsoluteLayout>
```



## 14.2 FrameLayout

Este contenedor presenta un solo elemento en su lado superior izquierdo. Cualquier otra vista que añadamos en su interior se superpondrá a la anterior. Sin embargo, también se puede utilizar para alternar entre elementos de forma dinámica. Por ejemplo, el siguiente proyecto permite alternar entre dos imágenes sin mas que pulsar sobre ellas.

Necesitamos introducir una vista nueva de tipo `ImageView`, que se utiliza para mostrar imágenes en Android. El fichero con la imagen se debe colocar en las carpetas de recursos, `drawable`, y asociarlas al atributo `android:src` como a continuación:

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/img0169"  
    android:scaleType="fitCenter" />
```

Colocaremos dos imágenes en la carpeta `drawable` de nombre `circulo.png` y `cuadrado.png`, respectivamente. En el siguiente fichero de diseño se añaden dos elementos de tipo `ImageView` al `FrameLayout`, uno por cada imagen:

/res/layout/activity\_main.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <ImageView
        android:id="@+id/circulo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:contentDescription="@string/circulo"
        android:onClick="circulo"
        android:scaleType="fitCenter"
        android:src="@drawable/circulo" />

    <ImageView
        android:id="@+id/cuadrado"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:contentDescription="@string/cuadrado"
        android:onClick="cuadrado"
        android:scaleType="fitCenter"
        android:src="@drawable/cuadrado" />

</FrameLayout>
```

El atributo `android:scaleType` sirve para escalar la imagen dentro de la vista. Sus posibles valores incluyen:

- `fitXY`: escala independientemente cada dimensión para ajustarla al tamaño de la vista sin mantener la proporción original.
- `center`: centra la imagen en el contenedor sin escalarla.
- `centerInside`: escala la imagen proporcionalmente para ajustarla al tamaño de la vista.
- `fitCenter`: escala la imagen proporcionalmente de tal forma que o bien su anchura o su altura se ajustan al tamaño de la vista.

Los valores asignados al atributo `android:onClick` de cada vista garantizan que cuando pulsemos la primera imagen se ejecutará el método `circulo()`, y cuando pulsemos la segunda, se ejecutará el método `cuadrado()`. El código de ambos métodos se ha añadido en el fichero `MainActivity.java`:

/src/MainActivity.java

```
package es.uam.eps.dadm.ccc14_2;

import android.os.Bundle;
import android.app.Activity;
import android.view.View;
import android.widget.ImageView;

public class MainActivity extends Activity {
    ImageView primera;
    ImageView segunda;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        primera = (ImageView) findViewById(R.id.circulo);
```

```

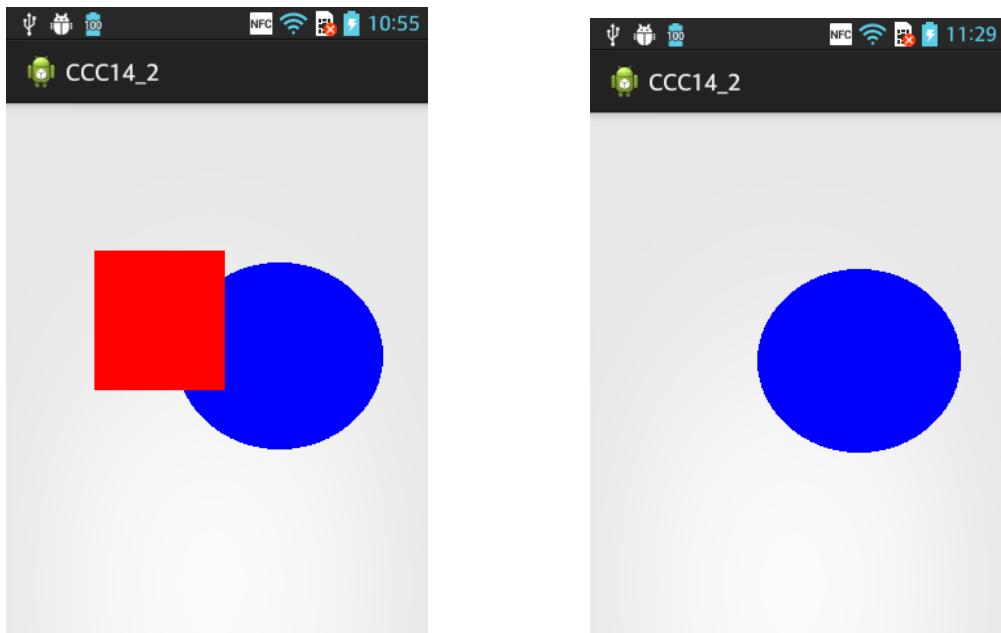
        segunda = (ImageView) findViewById(R.id.cuadrado);
    }

    public void circulo (View view){
        segunda.setVisibility(View.VISIBLE);
        primera.setVisibility(View.GONE);
    }

    public void cuadrado (View view){
        primera.setVisibility(View.VISIBLE);
        segunda.setVisibility(View.GONE);
    }
}

```

En el método `onCreate()` se consiguen referencias a cada una de las vistas, para poder actuar sobre su visibilidad dentro de los métodos `circulo()` y `cuadrado()`. Inicialmente las dos imágenes se ven superpuestas pero, al pulsar por primera vez, se ejecuta el método `cuadrado()`, con lo que se verá solo el círculo azul (a la derecha):



### 14.3 GridLayout

Este contenedor permite organizar los elementos en filas y columnas sin más que especificar la fila y columna donde se desea colocar cada elemento. Por ejemplo, el siguiente elemento de tipo `EditText` se situará en la primera fila y segunda columna:

```

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_row="0"
    android:layout_column="1"
    android:inputType="textPersonName"/>

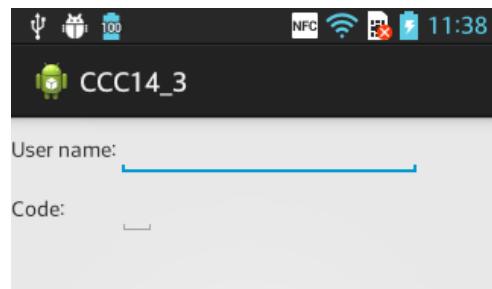
```

```
        android:minWidth="200dip"/>
```

Los elementos de tipo `EditText` permiten que el usuario introduzca texto. El atributo `inputType` permite especificar el tipo de contenido básico del campo de texto. Los elementos de tipo `TextView` sirven para mostrar mensajes. El siguiente fichero de diseño genera una interfaz para que los usuarios tecleen su nombre y código:

```
/res/layout/activity_main.xml
```

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_row="0"
        android:layout_column="0"
        android:text="@string/user_name" />
    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_row="0"
        android:layout_column="1"
        android:inputType="textPersonName"
        android:minWidth="200dip"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_row="1"
        android:layout_column="0"
        android:text="@string/code" />
    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_row="1"
        android:layout_column="1"
        android:inputType="textPassword"/>
</GridLayout>
```

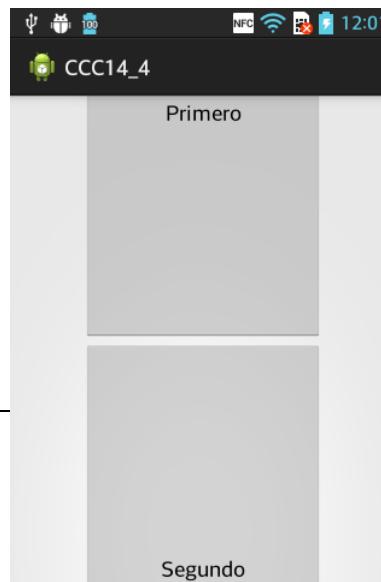


## 14.4 ScrollView

Este elemento permite mostrar contenido que no cabe en una sola pantalla mediante la utilización de una barra vertical de scroll. Solo puede contener un hijo, que suele ser otro contenedor como un `LinearLayout`, que a su vez contiene al resto de las vistas que deseamos acomodar en la pantalla. El siguiente ejemplo permite acomodar dos botones que se han dimensionado con una altura de 380dip, que no caben en una sola pantalla:

```
/res/layout/activity_main.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:gravity="center">
        <Button
            android:layout_width="200dip"
            android:layout_height="400dip"
            android:text="Primero" />
        <Button
            android:layout_width="200dip"
            android:layout_height="400dip"
            android:text="Segundo" />
    </LinearLayout>
</ScrollView>
```



## 14.5 TableLayout

El contenedor `TableLayout` organiza a sus hijos en filas y columnas. Cada fila se define mediante un elemento `TableRow` que puede tener 0 o más elementos. El número de columnas del contenedor es el máximo número de elementos en una fila.

La anchura de una columna es igual a la del elemento más ancho de esa columna. Sin embargo, se puede utilizar el atributo `android:stretchColumns` para especificar que ciertas columnas aprovecharán el espacio libre disponible. Por ejemplo, la siguiente instrucción especifica que lo harán las columnas primera y tercera:

```
android:stretchColumns="0,2"
```

Si en lugar de especificar el número de las columnas ponemos un asterisco, todas las columnas se ensanchan:

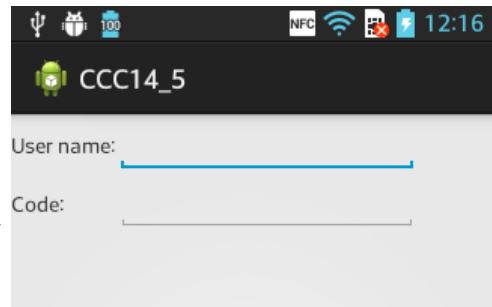
```
android:stretchColumns="*"
```

A continuación se muestra un fichero de diseño que reproduce la interfaz de la sección 14.3 mediante un `TableLayout` en lugar de un `GridLayout`:

```
/res/layout/activity_main.xml
```

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TableRow>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/user_name" />
        <EditText
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:minWidth="200dp"
            android:inputType="textPersonName"/>
    
    <TableRow>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/code" />
        <EditText
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:inputType="textPassword"/>
    

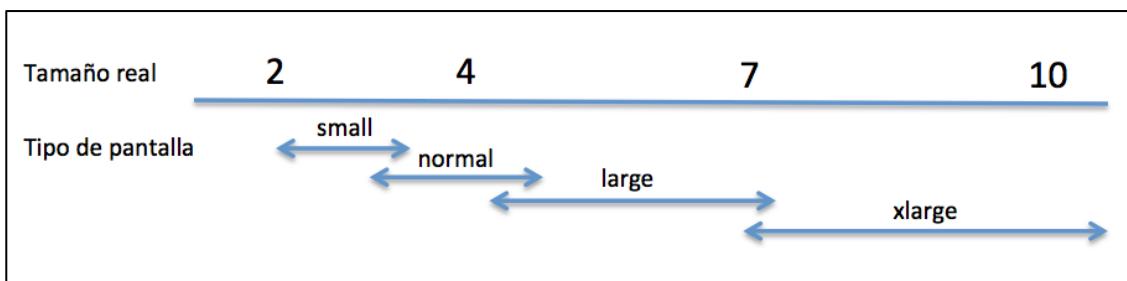
```



## 15. Tamaños de pantalla y unidades de medida

Una pantalla se caracteriza por su tamaño y su densidad. Android reconoce cuatro **tamaños**, medidos como el número de pulgadas de la diagonal:

- small: de 2 a 3 pulgadas
- normal: de 3 a 4,5 pulgadas
- large: de 4,5 a 7 pulgadas
- xlarge: de 7 a 10 pulgadas



La **densidad** de la pantalla es el número de píxeles por unidad de área (dpi o dots per inch, es decir, puntos por pulgada). Android reconoce cuatro densidades de pantalla (siguen aproximadamente las proporciones 3:4:6:8):

- baja (low density - ldpi): 120 dpi.
- media (medium density - mdpi): 160 dpi.
- alta (high density - hdpi): 240 dpi.
- extra alta (extra high density - xhdpi): 320 dpi.

Por ejemplo, el LG Optimus L5 E610 tiene una pantalla de 4 pulgadas y 320 x 480 píxeles. Podemos calcular su densidad como el número de píxeles por unidad de longitud de su diagonal:

$$\frac{\sqrt{320^2 + 480^2}}{4} \approx 144$$

Se trata, por lo tanto, de un dispositivo de densidad media.

Las siguientes unidades de medida se pueden utilizar para especificar el tamaño de los elementos de una interfaz Android (los dos primeros son los recomendados):

- **dp o dip (density independent pixel)**. 1 dip es equivalente a 1 pixel en una pantalla de densidad 160 dpi (dots per inch). Es la unidad recomendada para dimensiones de elementos en ficheros de diseño como, por ejemplo, `layout_margin`, `padding`, etc.
- **sp (scale independent pixel)**. Son píxeles independientes de densidad que tienen en cuenta el tamaño del font. Es la unidad recomendada para tamaños de texto.
- **pt (point)**. Un punto (point) es 1/72 de una pulgada (inch) de una pantalla física.

- `px` (*pixel*). Esta unidad se corresponde con los píxeles de una pantalla.

Como los `dips` son independientes de densidad y 1 `dip` equivale a 1 píxel en una pantalla de densidad media (160 dpi), 1 `dip` debería equivaler a 2 píxeles en una pantalla con el doble de densidad (320 dpi). En general, esta es la fórmula que convierte `dips` en píxeles:

$$px = dp \frac{dpi}{160}$$

donde `dpi` es la densidad de pantalla del dispositivo.

Para crear interfaces de usuario en Android debemos especificar distintos ficheros de diseño para cada tamaño de pantalla (utilizando recursos alternativos) y bitmaps específicos para cada densidad.

Los calificadores para especificar recursos con tamaños específicos son: `small`, `normal`, `large` y `xlarge`. Por ejemplo, el fichero de recurso de nombre `/res/layout-small/activity_main.xml` será seleccionado por Android en dispositivos con pantallas pequeñas. A partir del API 13, en lugar de estos calificadores, se recomienda utilizar el calificador `sw<N>dp` (anchura mínima). Por ejemplo, el recurso `/res/layout-sw600dp/activity_main.xml` se utilizará en dispositivos con una anchura de pantalla, cualquiera que sea la orientación del dispositivo, de al menos 600 `dps`. Esta es la forma más sencilla de especificar recursos alternativos independientemente de la orientación del dispositivo. Los siguientes ejemplos pueden servir de orientación:

- 320 dp: pantalla de un teléfono típico.
- 600 dp: pantalla de una tableta de 7".
- 720 dp: pantalla de una tableta de 10".

Es fundamental expresar las dimensiones de los elementos de la interfaz en `dips` en lugar de `px`. Para entender mejor esta cuestión, vamos a simular dos dispositivos de tamaño normal con densidades diferentes: Nexus 4 (xhdpi) y Nexus S (hdpi). El fichero de diseño contiene simplemente un botón cuya anchura especificamos primero en píxeles (`px`) y luego en píxeles independientes de densidad (`dip`):

`/res/layout/activity_main.xml`

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
    <Button
        android:layout_width="200px"
        android:layout_height="wrap_content"
        android:text="@string/hello_world"/>
</RelativeLayout>
```

En el grupo de la izquierda, con la anchura del botón expresada en píxeles, el tamaño del botón disminuye al aumentar la densidad (de izquierda a derecha).

En el grupo de la derecha, este efecto se compensa al especificar la dimensión en píxeles independientes de densidad:



En el caso de bitmaps no es suficiente con dar su tamaño en dips, ya que un dip se traduce en un número de píxeles distinto para cada densidad de pantalla. Como las imágenes se definen en píxeles, esto puede dar lugar a imágenes borrosas al contraer o ampliar. Para evitarlo deberíamos añadir una versión de cada imagen en distintas densidades.

Por ejemplo, si tenemos una imagen de 100x100 píxeles que se ve correctamente para una densidad media (mdpi), teniendo en cuenta que las densidades siguen aproximadamente las proporciones 3:4:6:8, deberíamos añadir las siguientes versiones del bitmap:

- bitmap de 75 x 75 píxeles en la carpeta `/res/drawable-ldpi`
- bitmap de 150 x 150 píxeles en la carpeta `/res/drawable-hdpi`
- bitmap de 200 x 200 píxeles en la carpeta `/res/drawable-xhdpi`

# Jugando con Android

Aprende a programar tu primera App

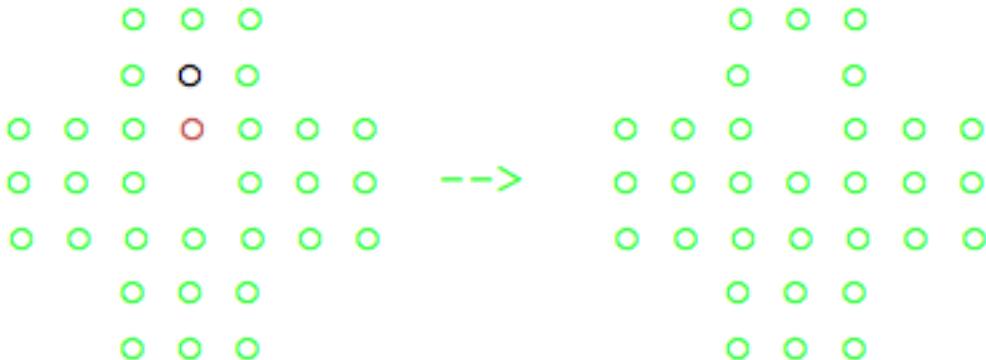
Semana 3. Interacción y modelo

## 16. El atributo android:onClick

¿Cómo podemos hacer que nuestra aplicación ejecute una tarea en respuesta a eventos de la interfaz de usuario tal y como la pulsación de uno de los botones? Existen distintas formas de resolver esta cuestión:

- Asignando el atributo `android:onClick` del botón. El valor de este atributo es el nombre del método que se ejecuta al pulsar el botón.
- Implementando un escuchador de eventos (*event listener*) y registrándolo en el botón.
- Haciendo que la actividad implemente la interfaz escuchador de eventos.

La forma más sencilla es la primera y se ilustra en el proyecto de esta unidad, CCC16, en el que vamos a empezar a gestionar los clicks del jugador sobre el tablero programando un método y asignándosele al botón número 5 (el de color negro en la figura inferior). Como resultado de la pulsación el estado de los botones 5 y 10 debe pasar a `false`, y a `true` el del botón 17, como se ve en la figura de la derecha:



El fichero de diseño del tablero es prácticamente igual al de la unidad 13 (proyecto CCC13) salvo por estas dos diferencias:

- Hemos eliminado las líneas correspondientes al atributo `android:checked`, que especifica el estado del botón, pues ahora fijaremos el estado de los botones con código Java.
- El botón con identificador `f5` tiene su atributo `android:onClick` asignado al método `onRadioButtonClick()`. Este es el método que se ejecutará al pulsar el botón:

```
<RadioButton  
    android:id="@+id/f5"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/f1"  
    android:layout_toRightOf="@+id/f4"  
    android:onClick="onRadioButtonClick"/>
```

Este es el aspecto de las dos primeras líneas de botones del nuevo fichero de diseño:

```
/res/layout/activity_main.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <RadioButton
        android:id="@+id/f2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"/>
    <RadioButton
        android:id="@+id/f1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/f2"/>
    <RadioButton
        android:id="@+id/f3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/f2"/>
    <RadioButton
        android:id="@+id/f4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/f1"
        android:layout_alignLeft="@id/f1"/>
    <RadioButton
        android:id="@+id/f5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/f1"
        android:layout_toRightOf="@id/f4"
        android:onClick="onRadioButtonClick"/>
    <RadioButton
        android:id="@+id/f6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/f1"
        android:layout_toRightOf="@id/f5"/>
    ...
</RelativeLayout>
```

El código de `MainActivity.java` es el siguiente:

```
/src/MainActivity.java
```

```
package es.uam.eps.android.ccc3;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.RadioButton;

public class MainActivity extends Activity {
    int SIZE = 7;

    private RadioButton button5;
    private RadioButton button10;
    private RadioButton button17;

    private final int[][] ids = {
        {0, 0, R.id.f1, R.id.f2, R.id.f3, 0, 0},
        {0, 0, R.id.f4, R.id.f5, R.id.f6, 0, 0},
        {R.id.f7, R.id.f8, R.id.f9, R.id.f10, R.id.f11, R.id.f12, R.id.f13},
        {R.id.f14, R.id.f15, R.id.f16, R.id.f17, R.id.f18, R.id.f19, R.id.f20},
```

```

{R.id.f21, R.id.f22, R.id.f23, R.id.f24, R.id.f25, R.id.f26, R.id.f27},
{0, 0, R.id.f28, R.id.f29, R.id.f30, 0, 0},
{0, 0, R.id.f31, R.id.f32, R.id.f33, 0, 0}};

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    setFigure();

    button5 = (RadioButton) findViewById(R.id.f5);
    button10 = (RadioButton) findViewById(R.id.f10);
    button17 = (RadioButton) findViewById(R.id.f17);
}

private void setFigure () {
    RadioButton button;

    for (int i=0; i<SIZE; i++)
        for (int j=0; j<SIZE; j++)
            if (ids[i][j]!=0){
                button = (RadioButton) findViewById(ids[i][j]);
                if (ids[i][j] != R.id.f17)
                    button.setChecked(true);
            }
}

public void onRadioButtonClick(View view) {
    button5.setChecked(false);
    button10.setChecked(false);
    button17.setChecked(true);
}
}

```

El movimiento de las fichas se va a simular mediante el cambio de su estado que ya no se fija en el fichero de diseño. Para cambiar el estado dinámicamente se utiliza el método `setChecked()`. Este método hemos de invocarlo desde una referencia al botón correspondiente. Estas referencias se consiguen llamando al método `findViewById()` de la actividad, al que hay que pasar el identificador del botón correspondiente.

Los identificadores de los botones se definen en el fichero de diseño y también se almacenan en un array de enteros de nombre `ids`. Cada elemento de este array, de dimensión  $7 \times 7$ , se corresponde con una posición del tablero, lo cual nos permitirá en unaidades posteriores obtener fácilmente las coordenadas de un botón en el tablero a partir de su identificador.

En el método `onCreate()` de `MainActivity`, una vez inflada la interfaz especificada en el fichero de diseño, llamamos al método `setFigure()` y conseguimos referencias a los botones 5, 10 y 17:

```

button5 = (RadioButton) findViewById(R.id.f5);
button10 = (RadioButton) findViewById(R.id.f10);
button17 = (RadioButton) findViewById(R.id.f17);

```

La referencia devuelta por el método `findViewById()` es de tipo `View` y la convertimos mediante un cast al tipo `RadioButton`.

El método `setFigure()` es el encargado de poner el estado de todos los botones a `true` salvo el central (`R.id.f17`):

```
private void setFigure () {
    RadioButton button;

    for (int i=0; i<SIZE; i++)
        for (int j=0; j<SIZE; j++)
            if (ids[i][j] !=0) {
                button = (RadioButton) findViewById(ids[i][j]);
                if (ids[i][j] != R.id.f17)
                    button.setChecked(true);
            }
}
```

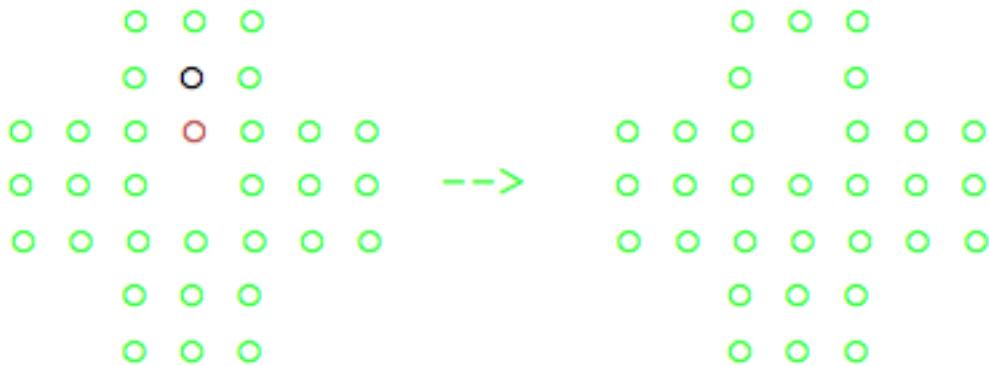
Una vez que disponemos de una referencia al objeto `RadioButton`, ajustamos su estado a `true` (marcado), siempre que no se trate del botón central:

```
if (ids[i][j] != R.id.f17)
    button.setChecked(true);
```

Solo nos falta por estudiar el método `onRadioButtonClicked()`, el método que se ejecuta al pulsar el botón con identificador `R.id.f5`. Este método ajusta a `false` los estados de los botones 5 y 10, y a `true` el estado del botón 17:

```
Public void onRadioButtonClick(View view) {
    button5.setChecked(false);
    button10.setChecked(false);
    button17.setChecked(true);
}
```

De esta manera tan sencilla simulamos el salto de la ficha 5 sobre la ficha 10, como se observa en la parte derecha de la siguiente figura:



Como ves, todavía estamos lejos de poder echar una partida al solitario. Para poder jugar de verdad hemos de dotar de funcionalidad al resto de los botones e implementar la lógica del juego, como veremos en unidades posteriores.

## 17. Escuchadores de eventos

En la unidad anterior estudiamos una forma sencilla de ligar un método a la pulsación de un botón mediante el atributo `android:onClick` del botón. En esta unidad veremos una forma más general de añadir funcionalidad a las vistas mediante la implementación de interfaces.

La clase `View` posee una colección de interfaces denominadas escuchadores de eventos. Como sabes, las interfaces en Java son clases cuyos métodos no tienen cuerpo. Las clases que implementen la interfaz deben suministrar necesariamente una versión de cada método de la interfaz.

Los escuchadores de eventos contienen un único método, denominado método callback. Por ejemplo, la interfaz `View.OnClickListener` declara el método `onClick()` que se llama automáticamente cuando el usuario toca la vista, por ejemplo.

### 17.1 Escuchando mediante clases con nombre

Para ilustrar la utilización de los escuchadores, vamos a programar una versión de nuestro juego funcionalmente equivalente a la de la unidad anterior pero con un escuchador en lugar de utilizar el atributo `android:onClick`. El fichero Java es el siguiente:

/src/MainActivity.java

```
package es.uam.eps.android.ccc17_1;

import android.app.Activity;
...
import android.widget.RadioButton;

public class MainActivity extends Activity {

    private int SIZE = 7;
    private RadioButton button5;
    private RadioButton button10;
    private RadioButton button17;

    private final int ids [][] = {
        {0, 0, R.id.f1, R.id.f2, R.id.f3, 0, 0},
        {0, 0, R.id.f4, R.id.f5, R.id.f6, 0, 0},
        {R.id.f7, R.id.f8, R.id.f9, R.id.f10, R.id.f11, R.id.f12, R.id.f13},
        {R.id.f14, R.id.f15, R.id.f16, R.id.f17, R.id.f18, R.id.f19, R.id.f20},
        {R.id.f21, R.id.f22, R.id.f23, R.id.f24, R.id.f25, R.id.f26, R.id.f27},
        {0, 0, R.id.f28, R.id.f29, R.id.f30, 0, 0},
        {0, 0, R.id.f31, R.id.f32, R.id.f33, 0, 0}};

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        setFigure();

        button5 = (RadioButton) findViewById(R.id.f5);
        button10 = (RadioButton) findViewById(R.id.f10);
        button17 = (RadioButton) findViewById(R.id.f17);
        button5.setOnClickListener(listener);
    }
}
```

```

    }

    private void setFigure () {
        RadioButton button;

        for (int i=0; i<SIZE; i++)
            for (int j=0; j<SIZE; j++) {
                if (ids[i][j]!=0) {
                    button = (RadioButton) findViewById(ids[i][j]);
                    if (ids[i][j] != R.id.f17)
                        button.setChecked(true);
                }
            }
        private OnClickListener listener = new OnClickListener() {
            public void onClick(View v) {
                RadioButton button = (RadioButton) v;
                button.setChecked(false);
                button10.setChecked(false);
                button17.setChecked(true);
            }
        };
    }
}

```

La diferencia entre la versión del método `onCreate()` de esta unidad y la versión de la unidad anterior es la asignación de un escuchador al botón `button5`. En general, para que el método callback del escuchador de eventos se ejecute al hacer click en un elemento de la interfaz, el escuchador debe registrarse para dicho elemento:

```
button5.setOnClickListener(listener);
```

El escuchador `listener` es una interfaz de tipo `View.OnClickListener` y, por lo tanto, a la vez que se instancia es necesario implementar el método callback `onClick()`, que lleva a cabo la misma tarea que el método `onRadioButtonClick()` de la unidad anterior. Básicamente, como ya vimos en la unidad anterior, el método `setChecked()` se utiliza para ajustar el estado de los botones y simular el salto de la ficha 5 sobre la ficha 10:

```

private OnClickListener listener = new OnClickListener() {
    public void onClick(View v) {
        RadioButton button = (RadioButton) v;
        button.setChecked(false);
        button10.setChecked(false);
        button17.setChecked(true);
    }
};

```

El fichero de diseño del proyecto `ccc17_1` es el mismo que el de la unidad anterior (`activity_main.xml` del proyecto `CCC16`).

## 17.2 Escuchando mediante clases anónimas

Podemos ahorrarnos la declaración de la interfaz `listener` instanciándola directamente en la lista de argumentos de `setOnClickListener()`:

```
button5.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        // TODO Auto-generated method stub  
        RadioButton button = (RadioButton) v;  
        button.setChecked(false);  
        button10.setChecked(false);  
        button17.setChecked(true);  
    }  
});
```

Quizá recuerdes que en Java a este tipo de clases se les denomina internas anónimas. Aunque al principio esta sintaxis resulta un poco intrincada, es una práctica bastante habitual en Android que tiene la ventaja de situar el código callback justo donde se necesita. Es el estilo recomendado cuando el escuchador solo se registra en una vista.

El proyecto `ccc17_2` utiliza este enfoque. El código completo de la actividad principal es el siguiente:

/src/MainActivity.java

```
package es.uam.eps.android.ccc17_2;  
  
import android.app.Activity;  
import android.os.Bundle;  
import android.view.View;  
import android.view.View.OnClickListener;  
import android.widget.RadioButton;  
  
public class MainActivity extends Activity {  
  
    private int SIZE = 7;  
    private RadioButton button5;  
    private RadioButton button10;  
    private RadioButton button17;  
  
    private final int ids [][] = {  
        {0, 0, R.id.f1, R.id.f2, R.id.f3, 0, 0},  
        {0, 0, R.id.f4, R.id.f5, R.id.f6, 0, 0},  
        {R.id.f7, R.id.f8, R.id.f9, R.id.f10, R.id.f11, R.id.f12, R.id.f13},  
        {R.id.f14, R.id.f15, R.id.f16, R.id.f17, R.id.f18, R.id.f19, R.id.f20},  
        {R.id.f21, R.id.f22, R.id.f23, R.id.f24, R.id.f25, R.id.f26, R.id.f27},  
        {0, 0, R.id.f28, R.id.f29, R.id.f30, 0, 0},  
        {0, 0, R.id.f31, R.id.f32, R.id.f33, 0, 0}};  
  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        setFigure();  
  
        button5 = (RadioButton) findViewById(R.id.f5);  
        button10 = (RadioButton) findViewById(R.id.f10);  
        button17 = (RadioButton) findViewById(R.id.f17);  
    }
```

```

        button5.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                RadioButton button = (RadioButton) v;
                button.setChecked(false);
                button10.setChecked(false);
                button17.setChecked(true);
            }
        });
    }

    private void setFigure() {
        RadioButton button;

        for (int i = 0; i < SIZE; i++)
            for (int j = 0; j < SIZE; j++)
                if (ids[i][j] != 0) {
                    button = (RadioButton) findViewById(ids[i][j]);
                    if (ids[i][j] != R.id.f17)
                        button.setChecked(true);
                }
    }
}

```

### 17.3 La actividad también escucha

Otra forma de esquivar la declaración del objeto `listener` consiste en hacer que la actividad implemente la interfaz `OnClickListener`. Veamos cómo hacer esto con el proyecto `CCC17_3`:

/src/MainActivity.java

```

package es.uam.eps.android.ccc17_3;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.RadioButton;

public class MainActivity extends Activity implements OnClickListener {

    private int SIZE = 7;
    private RadioButton button5;
    private RadioButton button10;
    private RadioButton button17;

    private final int ids [][] = {
        {0, 0, R.id.f1, R.id.f2, R.id.f3, 0, 0},
        {0, 0, R.id.f4, R.id.f5, R.id.f6, 0, 0},
        {R.id.f7, R.id.f8, R.id.f9, R.id.f10, R.id.f11, R.id.f12, R.id.f13},
        {R.id.f14, R.id.f15, R.id.f16, R.id.f17, R.id.f18, R.id.f19, R.id.f20},
        {R.id.f21, R.id.f22, R.id.f23, R.id.f24, R.id.f25, R.id.f26, R.id.f27},
        {0, 0, R.id.f28, R.id.f29, R.id.f30, 0, 0},
        {0, 0, R.id.f31, R.id.f32, R.id.f33, 0, 0}};
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        setFigure();

        button5 = (RadioButton) findViewById(R.id.f5);
        button10 = (RadioButton) findViewById(R.id.f10);
        button17 = (RadioButton) findViewById(R.id.f17);
        button5.setOnClickListener(this);
    }
}

```

```

private void setFigure () {
    RadioButton button;

    for (int i=0; i<SIZE; i++)
        for (int j=0; j<SIZE; j++) {
            if (ids[i][j]!=0) {
                button = (RadioButton) findViewById(ids[i][j]);
                if (ids[i][j] != R.id.f17)
                    button.setChecked(true);
            }
        }

    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        RadioButton button = (RadioButton) v;
        button.setChecked(false);
        button10.setChecked(false);
        button17.setChecked(true);
    }
}

```

Ahora es la actividad la que juega el papel del escuchador `listener` del proyecto `CCC17_1`, es decir, la que implementa el método `onClick()` de la interfaz y, por lo tanto, la que se registra como escuchador del botón 5:

```
button5.setOnClickListener(this);
```

## 18. La lógica del juego

Ya estamos listos para escribir un proyecto (ccc18) que nos permitirá jugar una partida completa al solitario y nos avisará con un mensaje cuando ya no podamos hacer mas movimientos.

El proyecto va a contar con dos clases:

- `Game.java`: con los métodos que implementan las reglas del juego como, por ejemplo, el método que comprueba si un salto es posible.
- `MainActivity.java`: que gestiona las pulsaciones del usuario y se las comunica al objeto de la clase `Game`.

Esta separación entre las reglas del juego y la interfaz gráfica permite mantener estos dos módulos de forma independiente. Más adelante podríamos por ejemplo modificar la interfaz gráfica sin necesidad de tocar la clase `Game`. Este es el enfoque utilizado en el patrón de diseño conocido como **modelo-vista-controlador**:

- El modelo es todo lo que tiene que ver con los datos, es decir, las reglas del juego en nuestro caso.
- La vista es la interfaz gráfica, es decir, el tablero del juego. En nuestra app la interfaz se especifica en el fichero de diseño `activity_main.xml`.
- El controlador es la actividad que sirve de intermediario entre la interfaz y el modelo: captura los eventos que tienen lugar en la vista, se los comunica al modelo para que se actualice y luego a la vista para que se redibuje.

El fichero de diseño utilizado en el proyecto ccc18 es el mismo que el utilizado en la unidad anterior. Empecemos estudiando la actividad `MainActivity` que, como puedes comprobar a continuación, contiene muchos elementos conocidos.

### 18.1 La clase `MainActivity`

```
/src/MainActivity.java
```

```
package es.uam.eps.android.ccc18;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.RadioButton;
import android.widget.Toast;

public class MainActivity extends Activity implements OnClickListener{
    Game game;
    static final int SIZE = 7;

    private final int ids [][] = {
        {0, 0, R.id.f1, R.id.f2, R.id.f3, 0, 0},
        {0, 0, R.id.f4, R.id.f5, R.id.f6, 0, 0},
        {R.id.f7, R.id.f8, R.id.f9, R.id.f10, R.id.f11, R.id.f12, R.id.f13},
        {R.id.f14, R.id.f15, R.id.f16, R.id.f17, R.id.f18, R.id.f19, R.id.f20},
        {R.id.f21, R.id.f22, R.id.f23, R.id.f24, R.id.f25, R.id.f26, R.id.f27},
        {0, 0, R.id.f28, R.id.f29, R.id.f30, 0, 0},
        {0, 0, R.id.f31, R.id.f32, R.id.f33, 0, 0}};
```

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    registerListeners();
    game = new Game();
    setFigureFromGrid();
}

private void registerListeners () {
    RadioButton button;

    for (int i=0; i<SIZE; i++)
        for (int j=0; j<SIZE; j++)
            if (ids[i][j]!=0){
                button = (RadioButton) findViewById(ids[i][j]);
                button.setOnClickListener(this);
            }
}

public void onClick (View v){
    int id = ((RadioButton) v).getId();

    for (int i=0; i<SIZE; i++)
        for (int j=0; j<SIZE; j++)
            if (ids[i][j] == id) {
                game.play(i, j);
                break;
            }

    setFigureFromGrid();
    if (game.isGameFinished())
        Toast.makeText(this, R.string.gameOverTitle, Toast.LENGTH_LONG).show();
}

private void setFigureFromGrid () {
    RadioButton button;

    for (int i=0; i<SIZE; i++)
        for (int j=0; j<SIZE; j++)
            if (ids[i][j] != 0){
                int value = game.getGrid(i, j);
                button = (RadioButton) findViewById(ids[i][j]);

                if (value == 1)
                    button.setChecked(true);
                else
                    button.setChecked(false);
            }
}
}

```

El método `onCreate()`, una vez inflada la interfa gráfica especificada en el fichero de diseño, llama a `registerListeners()`, que se encarga de registrar la actividad como escuchador para todos los botones, incluido el central. También instancia un objeto de tipo `Game`, que se utilizará dentro del método `onClick()`. Finalmente, sitúa las fichas sobre el tablero con el método `setFigureFromGrid()` (poniendo a `true` el estado de los botones necesarios). Este método utiliza la información del miembro `grid` de la clase `Game`, que es un array 7x7 de enteros: 1 indica que la posición correspondiente del tablero está ocupada por una ficha y 0 refleja que la posición está vacía.

El método `onClick()` identifica las coordenadas del botón pulsado por el jugador y se las pasa al método `play()` de la clase `Game`. Este método se encarga de actualizar el array `grid` de `Game`, de tal forma que la actividad pueda redibujar el tablero con el método `setFigureFromGrid()`. Finalmente, si el método `isGameFinished()` devuelve `true`, se muestra un mensaje que indica el final del juego mediante la clase `Toast`.

Un toast es una ventana flotante que presenta rápidamente un pequeño mensaje. El método `makeText()` construye el objeto `Toast` y toma tres argumentos:

1. El contexto.
2. Una referencia a un objeto `CharSequence` que contiene el mensaje que se desea mostrar.
3. La duración de la vista: `Toast.LENGTH_SHORT` o `Toast.LENGTH_LONG`.

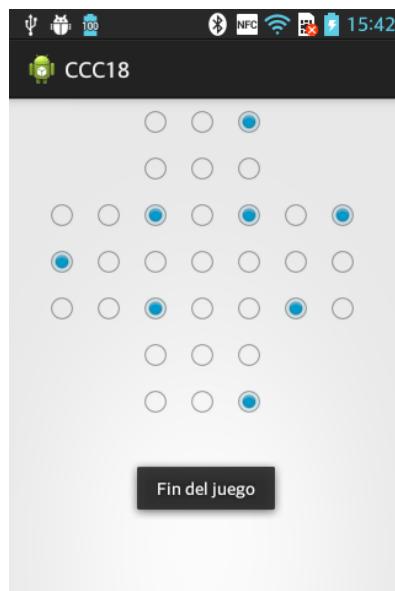
El método `makeText()` devuelve el objeto `Toast`. Es entonces cuando llamamos al método `show()` de `Toast` para mostrar el mensaje. El recurso `gameOverTitle` almacena el siguiente mensaje: Fin del juego.

```
/res/values/strings.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">CCC18</string>
    <string name="gameOverTitle">Fin del juego</string>

</resources>
```



Mas adelante aprenderemos cómo mostrar mensajes distintos dependiendo del idioma del dispositivo.

## 18.2 La clase Game

Veamos ahora cómo implementa `Game` las reglas del juego:

```
/src/Game.java
```

```
package es.uam.eps.android.ccc18;

public class Game {
    static final int SIZE = 7;
    private int grid[][];
    private static final int CROSS[][]={{0,0,1,1,1,0,0},
                                      {0,0,1,1,1,0,0},
```

```

        {1,1,1,1,1,1,1},
        {1,1,1,0,1,1,1},
        {1,1,1,1,1,1,1},
        {0,0,1,1,1,0,0},
        {0,0,1,1,1,0,0}};
private static final int BOARD[][]={{0,0,1,1,1,0,0},
        {0,0,1,1,1,0,0},
        {1,1,1,1,1,1,1},
        {1,1,1,1,1,1,1},
        {1,1,1,1,1,1,1},
        {0,0,1,1,1,0,0},
        {0,0,1,1,1,0,0}};

private int pickedI, pickedJ;
private int jumpedI, jumpedJ;
private enum State {READY_TO_PICK, READY_TO_DROP, FINISHED};
private State gameState;

public Game(){
    grid = new int [SIZE][SIZE];

    for (int i=0; i<SIZE; i++)
        for (int j=0; j<SIZE; j++)
            grid[i][j] = CROSS[i][j];

    gameState = State.READY_TO_PICK;
}

public int getGrid(int i, int j){
    return grid[i][j];
}

public boolean isAvailable(int i1, int j1, int i2, int j2) {

    if (grid[i1][j1]==0 || grid[i2][j2] == 1)
        return false;

    if (Math.abs(i2-i1) == 2 && j1 == j2)
    {
        jumpedI = i2 > i1 ? i1+1: i2+1;
        jumpedJ = j1;
        if (grid[jumpedI][jumpedJ] == 1)
            return true;
    }

    if (Math.abs(j2-j1) == 2 && i1 == i2)
    {
        jumpedI = i1;
        jumpedJ = j2 > j1 ? j1+1: j2+1;
        if (grid[jumpedI][jumpedJ] == 1)
            return true;
    }

    return false;
}

public void play (int i, int j) {
    if (gameState == State.READY_TO_PICK) {
        pickedI = i;
        pickedJ = j;
        gameState = State.READY_TO_DROP;
    } else if (gameState == State.READY_TO_DROP) {
        if (isAvailable(pickedI, pickedJ, i, j)) {
            gameState = State.READY_TO_PICK;

            grid[pickedI][pickedJ] = 0;
            grid[jumpedI][jumpedJ] = 0;
            grid[i][j] = 1;

            if (isGameFinished())
                gameState = State.FINISHED;
        }
        else {
            pickedI=i;
            pickedJ=j;
        }
    }
}

```

```

public boolean isGameFinished () {
    for (int i=0; i<SIZE; i++)
        for (int j=0; j<SIZE; j++)
            for (int p=0; p<SIZE; p++)
                for (int q=0; q<SIZE; q++)
                    if (grid[i][j]==1 && grid[p][q]==0 && BOARD[p][q]==1)
                        if (isAvailable(i, j, p, q))
                            return false;
    return true;
}

```

Para empezar hay que tener en cuenta que el juego puede estar en uno de los tres estados siguientes:

- `READY_TO_PICK`: el juego está en este estado antes de pulsar una ficha que se desea mover.
- `READY_TO_DROP`: el juego está en este estado cuando espera que indiquemos dónde deseamos dejar caer una ficha previamente seleccionada.
- `FINISHED`: el juego está en este estado si no podemos mover ninguna ficha.

Veamos algunos de los miembros de `Game`. Tenemos tres arrays bidimensionales de enteros de dimensión 7x7:

- El miembro `grid` es el único que se modifica a lo largo del juego ya que indica en cada momento si hay una ficha (1) o no (0) en cada posición.
- El miembro `CROSS` almacena la figura inicial, es decir, las posiciones del tablero que tienen ficha inicialmente (1).
- El miembro `BOARD` tiene a 1 las posiciones accesibles del tablero. Se diferencia de `CROSS` en la posición central, que es accesible pero no tiene ficha inicialmente.

Además, (`pickedI`, `pickedJ`) son las coordenadas de la última ficha que quiere mover el jugador y (`jumpedI`, `jumpedJ`) son las coordenadas de la última ficha que hemos intentado saltar.

El constructor de `Game` inicializa el estado del juego con el valor `READY_TO_PICK` y el array `grid` con el array `CROSS`. Mas adelante tendremos ocasión de inicializar el tablero con otras figuras iniciales a elección del jugador.

El método `isAvailable()` devuelve `true` si la ficha con coordenadas (`i1,j1`) puede saltar a la posición (`i2,j2`). Para ello el método ejecuta las siguientes tareas:

- Comprueba que la posición de origen tiene ficha y que la posición de destino está vacía.
- Comprueba que las posiciones de origen y destino están a dos unidades de distancia en fila (segundo `if`) o en columna (primer `if`).
- Calcula la posición intermedia (`jumpedI`, `jumpedJ`) y comprueba que tiene ficha.

El método `play()` funciona de la siguiente manera:

- Si el estado del juego es `READY_TO_PICK`, guarda las coordenadas de la ficha seleccionada en `pickedI` y `pickedJ` para su uso mas adelante, en otra llamada a `play()`.
- Si el estado del juego es `READY_TO_DROP`, comprueba que el salto de la ficha seleccionada en una llamada anterior (`pickedI`, `pickedJ`) a la posición ( $i$ ,  $j$ ), la última pulsada, es posible. En ese caso, devuelve el estado del juego al valor `READY_TO_PICK` y actualiza el array `grid` para reflejar el salto, y que luego lo utilice la actividad para redibujar el tablero. Si el salto no es posible, actualiza (`pickedI`, `pickedJ`) con las coordenadas de la última posición, es decir, el destino seleccionado se convierte en origen para un movimiento posterior del jugador.

Por último el método `isGameFinished()` comprueba si existe algún salto válido con las fichas que quedan sobre el tablero. Si encuentra alguno, devuelve `false`. En caso contrario, devuelve `true`. Para ello recorre todos los posibles pares de posiciones inicial y final, incluso con repetición para simplificar el código. Antes de comprobar si el salto entre ( $i, j$ ) y ( $p, q$ ) es posible, comprueba que la casilla ( $i, j$ ) está llena, que la casilla ( $p, q$ ) está vacía y, además, es una casilla válida, lo cual está registrado en el array `BOARD`.

# Jugando con Android

Aprende a programar tu primera App

Semana 4. Ciclo de vida e intenciones

## 19. El ciclo de vida de una actividad

Las actividades atraviesan una serie de estados desde el momento en que se crean hasta su destrucción. A lo largo de este recorrido se invocan automáticamente un conjunto de métodos estándar. La clase base `Activity` proporciona versiones por omisión de estos métodos que, opcionalmente, el programador puede sobrecargar. De hecho, ya has sobrecargado el método `onCreate()` para inflar la interfaz desde el fichero de diseño. Enseguida veremos cómo y para qué sobrecargar el resto de los métodos del ciclo de vida.

Una actividad puede estar:

- **Activa:** la actividad está en primer plano de la pantalla y tiene el foco.
- **En pausa:** la actividad sigue visible pero sin foco como, por ejemplo, cuando un diálogo la tapa parcialmente. Toda la información de estado se mantiene pero el sistema puede decidir matar la actividad en ciertas situaciones extremas.
- **Parada:** la actividad está cubierta completamente por otra actividad. Se mantiene la información de estado pero es probable que el sistema la mate si necesita memoria.

Una actividad en pausa o parada puede volver a estar activa. Todas estas transiciones vienen acompañadas por llamadas a un conjunto estándar de métodos como se puede ver en la Figura 1:

- `onCreate()`: llamado cuando la actividad se crea por primera vez. Este método se utiliza habitualmente para crear la interfaz de usuario, capturar referencias a las vistas de la interfaz y asignar escuchadores. Cuenta con un argumento de tipo `Bundle` (conjunto de pares clave-valor) con información del estado anterior de la actividad, que se utiliza para recrearla tras una rotación, por ejemplo. Siempre le sigue el método `onStart()`.
- `onStart()`: llamado cuando la actividad se vuelve visible para el usuario. En general le sigue el método `onResume()` u `onRestoreInstanceState()` cuando la actividad se recrea por ejemplo tras una rotación del dispositivo.
- `onRestoreInstanceState()`: llamado cuando la actividad se recrea a partir de un estado guardado en el argumento de tipo `Bundle`.
- `onResume()`: llamado cuando la actividad empieza a interactuar con el usuario. Sobrecarga este método para arrancar código que necesite ejecutarse cuando la actividad está en primer plano. La actividad pasa a estar *Activa*.
- `onPause()`: llamado cuando el sistema está a punto de reanudar una actividad previa o lanzar una nueva. También se llama cuando la actividad pasa a estar parcialmente visible, por ejemplo al abrir un diálogo. En estos casos la actividad pasa al estado *En pausa*. A este método le puede seguir `onResume()` u `onStop()`, dependiendo de lo que ocurra. Puedes sobrecargar `onPause()` para guardar información y detener código que debe ejecutarse tan solo cuando la actividad está en primer plano (animaciones, música,

etc). En casos extremos, la aplicación puede ser destruida sin que se realice la llamada a `onDestroy()`, como se puede ver en la Figura 1. Por lo tanto, el método `onPause()` es el adecuado para liberar recursos pues puede ser el último método del ciclo de vida que llegue a ejecutarse. Hay que tener en cuenta que, como la siguiente actividad no será reanudada hasta que `onPause()` termine, conviene que la ejecución del método sea lo más rápida posible.

- `onSaveInstanceState()`: llamado en ocasiones después de `onPause()`, como por ejemplo cuando se gira el dispositivo, y antes de que la actividad se mate de tal forma que se pueda recuperar su estado en `onCreate()` u `onRestoreInstanceState()`. El objeto de tipo `Bundle` creado se pasa a ambos métodos.
- `onStop()`: llamado cuando la actividad deja de ser visible completamente al usuario porque otra actividad ha sido reanudada y la está cubriendo.
- `onDestroy()`: llamado cuando la actividad se destruye. El método `isFinishing()` permite averiguar si la destrucción se debe a que se invocó el método `finish()` o bien fue el sistema quien mató la actividad.
- `onRestart()`: llamado cuando la actividad ha sido parada y se reanuda posteriormente.

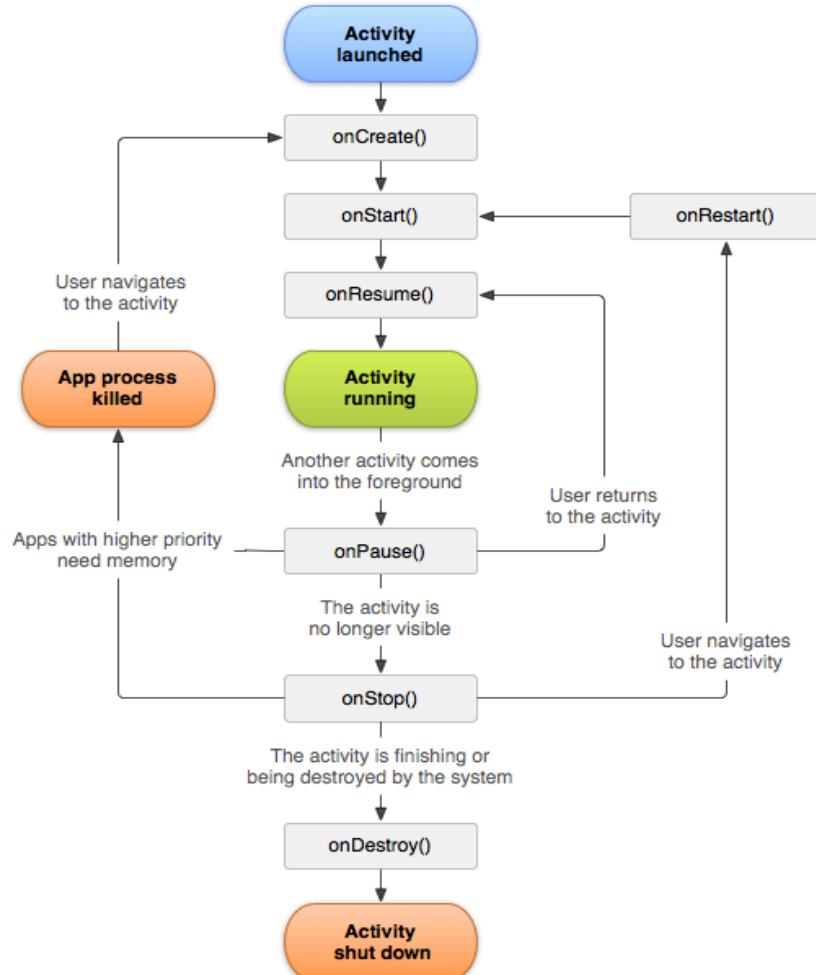
Cuando se sobrecargan estos métodos hay que llamar a la implementación en la superclase, de lo contrario se lanzará una excepción.

## 19.1 Dos ciclos de ejecución habituales

En la Figura 1 se pueden apreciar tres ciclos que ocurren durante la ejecución de una actividad. Dos de ellos son los más frecuentes: El ciclo `onPause() -> onResume()` y el ciclo `onPause() -> onStop() -> onRestart() -> onStart() -> onResume()`. Veamos qué se suele hacer en cada uno de estos ciclos:

- **Pausando y reanudando una actividad:** El método `onPause()` se invoca cuando la aplicación pasa a segundo plano. En este método se deben parar acciones que consuman CPU como videos, animaciones, etc. y liberar recursos que consuman batería (como accesos a GPS, cámara, etc.). Además se deben guardar los cambios que el usuario espera que sean permanentes. El método `onResume()` es la contrapartida de `onPause()` y se deben inicializar elementos que se liberan en `onPause()`. Hay que tener en cuenta que `onResume()` se ejecuta también al crear una actividad y no siempre tras un `onPause()`.
- **Parando y reiniciando una actividad:** El método `onStop()` se invoca cuando la actividad está completamente oculta al usuario. Este método debe usarse para operaciones que requieran mayor uso de CPU como guardar información en la base de datos. El método `onStart()` se puede usar para hacer comprobaciones sobre las capacidades del sistema (como comprobar si el GPS está activado). Algunos casos habituales en los que la actividad se para y se reinicia son:
  - Cuando el usuario navega a otra aplicación y luego vuelve.

- Cuando la actividad abre otra actividad y luego el usuario pulsa el botón back.
- Cuando se recibe una llamada y luego se cuelga.



**Figura 1.** Ciclo de vida de una actividad<sup>1</sup>.

## 19.2 El ciclo de vida en LogCat

El siguiente proyecto, CCC19, sirve para entender en qué situaciones se llama a cada uno de los métodos del ciclo de vida de una actividad. Puedes utilizar como fichero de diseño el generado automáticamente. El código Java es el siguiente:

---

<sup>1</sup> Figura perteneciente al proyecto [Android Open Source](#) y utilizada de acuerdo a los términos descritos en [Creative Commons 2.5 Attribution License](#).

```
/src/MainActivity.java
```

```
package es.uam.eps.android.ccc19;

import android.os.Bundle;
import android.app.Activity;
import android.util.Log;

public class MainActivity extends Activity {

    private void log(String text) {
        Log.d("LifeCycleTest", text);
    }
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        log("created");
    }
    public void onStart() {
        super.onStart();
        log("started");
    }
    public void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);
        log("onRestoreInstanceState() called");
    }
    protected void onResume() {
        super.onResume();
        log("resumed");
    }
    protected void onPause() {
        super.onPause();
        log("paused");
    }
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        log("onSaveInstanceState() called");
    }
    protected void onStop() {
        super.onStop();
        log("stopped");
    }
    protected void onDestroy() {
        super.onDestroy();
        log("destroyed");
    }
    protected void onRestart() {
        super.onRestart();
        log("restarted");
    }
}
```

La primera instrucción de los métodos sobrecargados ha de ser una llamada al método implementado en la clase `Activity`:

```
super.onCreate(savedInstanceState);
```

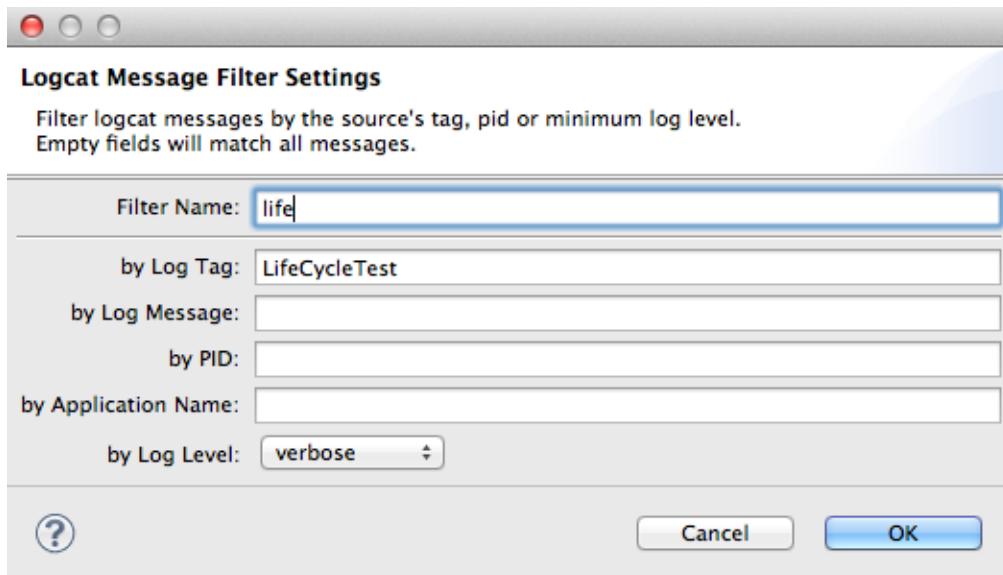
Por ejemplo, en `onCreate()`, el método de la superclase utiliza el estado almacenado de las vistas en el objeto de tipo `Bundle` para recrear la jerarquía de vistas de la actividad. El único método del código anterior no heredado de la clase base `Activity` es el método privado `log()`, el cual envía un mensaje marcado como `LifeCycleTest`:

```
Log.d ("LifeCycleTest", text);
```

Para ver estos mensajes, abre la vista `LogCat` de Eclipse pulsando `Window->Show View->LogCat`. Debería aparecer una nueva pestaña denominada `LogCat`:

Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.					
Level	Time	PID	Application	Tag	Text
E	06-19 16:22:05.775	567	com.programming.a...	AndroidRuntime	at com.ap...
E	06-19 16:22:05.775	567	com.programming.a...	AndroidRuntime	at dalvik...
E	06-19 16:22:05.775	567	com.programming.a...	AndroidRuntime	Caused by: ji...
E	06-19 16:22:05.775	567	com.programming.a...	AndroidRuntime	at android...
E	06-19 16:22:05.775	567	com.programming.a...	AndroidRuntime	at android...
E	06-19 16:22:05.775	567	com.programming.a...	AndroidRuntime	at android...
E	06-19 16:22:05.775	567	com.programming.a...	AndroidRuntime	at android...
E	06-19 16:22:05.775	567	com.programming.a...	AndroidRuntime	at android...
E	06-19 16:22:05.775	567	com.programming.a...	AndroidRuntime	at android...
E	06-19 16:22:05.775	567	com.programming.a...	AndroidRuntime	at android...
E	06-19 16:22:05.775	567	com.programming.a...	AndroidRuntime	at android...
E	06-19 16:22:05.775	567	com.programming.a...	AndroidRuntime	at android...
I	06-19 16:22:08.516	567	com.programming.a...	Process	Sending sign...

Como puedes ver el número de mensajes es muy elevado. Para centrarnos en los que emite nuestro programa resulta cómodo añadir un filtro pulsando el signo + en verde. Tienes que elegir un nombre de filtro (*Filter Name*), una marca (*tag*) (`LifeCycleTest` en nuestro caso) y pulsar **OK**. Este filtro selecciona los mensajes marcados como `LifeCycleTest`:



Ejecuta la aplicación siguiendo los pasos 1 a 5 y responde a las cuestiones estudiando los mensajes de la ventana LogCat:

1. ¿Qué métodos se ejecutan cuando se llama a la aplicación por primera vez?
2. Pulsa el botón Hacia Atrás (*Back*). ¿Qué métodos se ejecutan? ¿Se destruye la actividad?
3. Ejecuta de nuevo la aplicación.
4. Baja la cortina de notificaciones y pulsa en cualquier notificación pendiente. ¿Qué métodos se ejecutan en este caso?
5. Vuelve a la actividad pulsando el botón Hacia Atrás. ¿Qué métodos se ejecutan en esta ocasión?
6. Gira el dispositivo y observa los métodos ejecutados.

## 20. Música en tu dispositivo

Para reproducir una pieza de música vamos a añadir la siguiente clase `Music` a nuestro proyecto:

/src/Music.java

```
package es.uam.eps.android.ccc20;

import android.content.Context;
import android.media.MediaPlayer;

public class Music {
    private static MediaPlayer player;

    public static void play (Context context, int id){
        player = MediaPlayer.create(context, id);
        player.setLooping(true);
        player.start();
    }

    public static void stop (Context context){
        if(player != null){
            player.stop();
            player.release();
            player = null;
        }
    }
}
```

La clase `MediaPlayer` se utiliza para controlar la ejecución de ficheros de audio y vídeo. El método `create()` crea un objeto `MediaPlayer` asociado a un identificador que apunta a un cierto fichero como, por ejemplo, un fichero mp3.

Si el argumento del método `setLooping()` es `true`, la reproducción del fichero se repite indefinidamente.

El método `start()` arranca o reanuda la reproducción. Tras parar la reproducción, es importante llamar inmediatamente al método `release()` de tal forma que se liberen los recursos utilizados por el motor del reproductor interno asociado con el objeto `MediaPlayer`.

Si deseamos reproducir música mientras que la actividad se encuentre en primer plano, deberemos situar la llamada `play()` dentro del método `onResume()`, pues este método del ciclo de vida de la actividad se ejecuta cuando ésta empieza a interactuar con el usuario:

```
protected void onResume () {
    super.onResume ();
    Music.play(this, R.raw.sampleaudio);
}
```

Recuerda que el método `onPause()` del ciclo de vida de la actividad se ejecuta automáticamente cuando el sistema está a punto de lanzar otra actividad o reanudar una actividad previa.

Por lo tanto, si queremos que la música se detenga cuando la actividad deje el primer plano, el objeto `MediaPlayer` deberá ser parado en el método `onPause()`:

```
protected void onPause() {
    super.onPause();
    Music.stop(this);
}
```

El fichero `MainActivity.java` del proyecto `CCC20` añade los métodos `onResume()` y `onPause()` al correspondiente fichero Java del proyecto de la unidad 18 (`CCC18`):

/src/MainActivity.java

```
package es.uam.eps.android.ccc20;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.RadioButton;
import android.widget.Toast;

public class MainActivity extends Activity implements OnClickListener{
    Game game;
    static final int SIZE = 7;

    private final int ids [][] = {
        {0, 0, R.id.f1, R.id.f2, R.id.f3, 0, 0},
        {0, 0, R.id.f4, R.id.f5, R.id.f6, 0, 0},
        {R.id.f7, R.id.f8, R.id.f9, R.id.f10, R.id.f11, R.id.f12, R.id.f13},
        {R.id.f14, R.id.f15, R.id.f16, R.id.f17, R.id.f18, R.id.f19, R.id.f20},
        {R.id.f21, R.id.f22, R.id.f23, R.id.f24, R.id.f25, R.id.f26, R.id.f27},
        {0, 0, R.id.f28, R.id.f29, R.id.f30, 0, 0},
        {0, 0, R.id.f31, R.id.f32, R.id.f33, 0, 0}};

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        registerListeners();
        game = new Game();
        setFigureFromGrid();
    }

    private void registerListeners () {
        RadioButton button;

        for (int i=0; i<SIZE; i++)
            for (int j=0; j<SIZE; j++)
                if (ids[i][j]!=0)
                    button = (RadioButton) findViewById(ids[i][j]);
                    button.setOnClickListener(this);
    }

    public void onClick (View v){
        int id = ((RadioButton) v).getId();

        for (int i=0; i<SIZE; i++)
            for (int j=0; j<SIZE; j++)
                if (ids[i][j] == id) {
                    game.play(i, j);
                    break;
                }

        setFigureFromGrid();
        if (game.isGameFinished())
            Toast.makeText(this, R.string.gameOverTitle, Toast.LENGTH_LONG).show();
    }

    private void setFigureFromGrid (){
        RadioButton button;

        for (int i=0; i<SIZE; i++)
            for (int j=0; j<SIZE; j++)
```

```

        if (ids[i][j] != 0) {
            int value = game.getGrid(i, j);
            button = (RadioButton) findViewById(ids[i][j]);

            if (value == 1)
                button.setChecked(true);
            else
                button.setChecked(false);
        }
    }

    protected void onResume() {
        super.onResume();
        Music.play(this, R.raw.funkandblues);
    }

    protected void onPause() {
        super.onPause();
        Music.stop(this);
    }
}

```

El archivo `funkandblues.mp3` debe situarse en una carpeta de nombre `/res/raw/`. Su identificador se pasa como segundo argumento del método `play()` en `onResume()`. Puedes encontrar archivos sin derechos de autor en el siguiente enlace:

<http://incompetech.com/m/c/royalty-free/>.

Resumiendo, el proyecto `CCC20` cuenta ya con tres archivos Java:

- `Game.java`: lógica del juego.
- `MainActivity.java`: gestión de las pulsaciones del tablero.
- `Music.java`: evoltorio de la clase `MediaPlayer` de Android.

El archivo de diseño `activity_main.xml` se puede copiar del proyecto `CCC18`.

## 21. Orientación del dispositivo

Android permite dos orientaciones de la pantalla: retrato (*portrait*) y apaisado (*landscape*). Como sabes, el fichero `activity_main.xml` dentro de `/res/layout/` especifica la interfaz de usuario de la actividad en modo retrato. Si creas una carpeta alternativa de nombre `/res/layout-land/` con una nueva versión de `activity_main.xml`, este fichero especificará la interfaz de la actividad en modo apaisado. Android cargará automáticamente el fichero adecuado dependiendo de la orientación.

Android recrea las actividades, es decir, las destruye y vuelve a crear, cuando tiene lugar un cambio de orientación, cuando se conecta un teclado o una pantalla externa, etc. Al volver a ejecutarse el método `onCreate()` es cuando se infla el fichero alternativo si es que se ha suministrado.

Para evitar que la actividad se recree después de una o varias de estas circunstancias tenemos que indicarlas en el atributo `android:configChanges` del elemento `activity` del fichero de manifiesto, concatenadas mediante el símbolo `|`. Por ejemplo, esto es lo que hay que hacer para evitar que la actividad se recree después de una rotación, evitando que se actualice la interfaz:

```
<activity name=".MainActivity" android:configChanges="orientation|screenSize"/>
```

Para completar nuestro proyecto vamos a añadir el siguiente fichero de diseño para el modo apaisado:

`/res/values/layout-land/activity_main.xml`

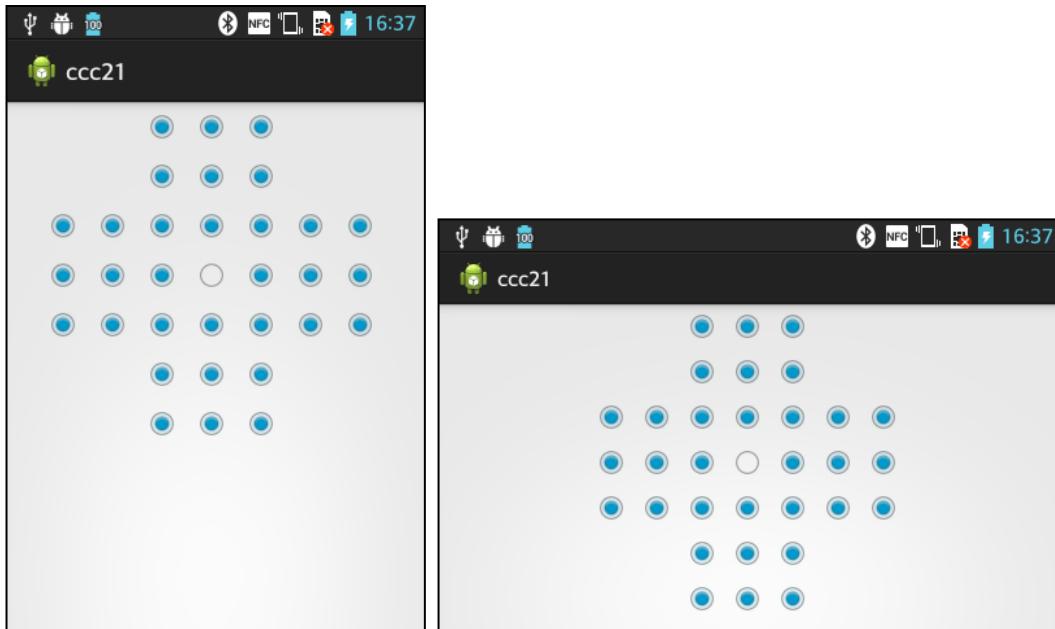
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <RadioButton
        android:id="@+id/f2"
        android:layout_width="@dimen/button_side"
        android:layout_height="@dimen/button_side"
        android:layout_centerHorizontal="true"/>
    <RadioButton
        android:id="@+id/f1"
        android:layout_width="@dimen/button_side"
        android:layout_height="@dimen/button_side"
        android:layout_toLeftOf="@id/f2"/>
    ...
    <RadioButton
        android:id="@+id/f33"
        android:layout_width="@dimen/button_side"
        android:layout_height="@dimen/button_side"
        android:layout_alignBaseline="@id/f31"
        android:layout_toRightOf="@id/f32"/>
</RelativeLayout>
```

Como ves la única diferencia entre este archivo y el del modo retrato es el tamaño de los botones. Aquí utilizamos un recurso de dimensión definido en el archivo `dimens.xml`, de tal forma que quepa el tablero completo en una pantalla de tamaño normal:

```
/res/values/dimens.xml
```

```
<resources>
    <!-- Default screen margins, per the Android Design guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="button_side">35dp</dimen>
</resources>
```

A continuación puedes comparar el aspecto de la interfaz en modo retrato y apaisado en un teléfono LG OPTIMUS L5 E610:



Cuando una actividad se recrea, por ejemplo, después de un cambio de orientación, su estado actual puede perderse. Para asegurarnos de la conservación del estado, debemos sobrecargar los métodos `onPause()` u `onSaveInstanceState()`:

- Puedes sobrecargar `onPause()` para guardar los datos de tu actividad en una base de datos o en un fichero. La llamada al método `onPause()` está garantizada antes de la destrucción de la actividad, así que es el momento en el que debemos guardar datos necesarios para las siguientes ejecuciones de la aplicación, como un correo en borrador.
- Para evitar el uso de una base de datos, también puedes sobrecargar `onSaveInstanceState()`, que viene con un objeto `Bundle` como argumento y también se llama cuando una actividad está a punto de ser abortada. Sin embargo, al revés que `onPause()`, este método puede no ser llamado cuando la actividad se destruye. Por ejemplo, este método nunca se llama cuando el usuario pulsa el botón Back.

El objeto `Bundle` nos permite almacenar datos en pares clave-valor. Por ejemplo, el siguiente método almacena el miembro `grid` de `Game` en la cadena `GRID` mediante una llamada a `putString()`:

```
public void onSaveInstanceState (Bundle outState) {  
    outState.putString("GRID", game.gridToString());  
    super.onSaveInstanceState(outState);  
}
```

Para ello hemos añadido el método `gridToString()` a la clase `Game`, que transforma el array `grid` en una cadena de caracteres:

```
public String gridToString (){  
    String str = "";  
    for (int i=0; i<SIZE; i++)  
        for (int j=0; j<SIZE; j++)  
            str += grid[i][j];  
    return str;  
}
```

Cuando la actividad se recrea, primero se ejecuta `onCreate()`, luego `onStart()` y luego `onRestoreInstanceState()` (repasa la unidad 19), al que se le pasa el objeto de tipo `Bundle` salvado por el método `onSaveInstanceState()`. Esto nos permite recuperar el estado del tablero guardado en la cadena `GRID`:

```
public void onRestoreInstanceState (Bundle savedInstanceState){  
    super.onRestoreInstanceState(savedInstanceState);  
    String grid = savedInstanceState.getString("GRID");  
    game.stringToGrid(grid);  
    setFigureFromGrid();  
}
```

En este método utilizamos el método `stringToGrid()` añadido para este propósito a la clase `Game`:

```
public void stringToGrid (String str){  
    for (int i=0, cont=0; i<SIZE; i++)  
        for (int j=0; j<SIZE; j++)  
            grid[i][j] = str.charAt(cont++)-'0';  
}
```

De forma parecida podemos almacenar mas información como, por ejemplo, el estado del juego o alguno de los miembros necesarios en la implementación de la lógica. Puedes encontrar el código completo de `MainActivity` en el fichero del proyecto asociado a esta unidad: CCC21.

## 22. Intenciones

Una aplicación puede contener cero o más actividades. Cuando una aplicación posee más de una actividad, es bastante frecuente arrancar una de ellas desde otra. Esto se consigue mediante una intención (*Intent* en inglés).

Vamos a utilizar una intención para iniciar nuestro juego desde una pantalla inicial que muestra una imagen. Hasta ahora todas nuestras aplicaciones tenían una única actividad. La aplicación de esta unidad posee dos: `Initial` y `MainActivity`, que deben especificarse en el fichero manifiesto:

`AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="es.uam.eps.android.CCC22"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="15"
        android:targetSdkVersion="19" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".Initial"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".MainActivity">
            <intent-filter>
                <action android:name="es.uam.eps.android.CCC22.MAINACTIVITY" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Veamos lo que significan los atributos del elemento `activity`:

- El atributo `android:name` especifica el nombre de la clase, que extiende actividad. Se puede indicar el nombre completo de la clase como, por ejemplo, `es.uam.eps.android.CCC22.Initial`, o como alternativa más sencilla, si se empieza por punto, podemos poner solo el nombre de la clase (`.Initial`, por ejemplo).
- El atributo `android:label` es el texto que se muestra en la barra de acción de la pantalla cuando la actividad está visible. En caso de que no se especifique, como en el caso de `MainActivity`, se mostrará la etiqueta especificada en el elemento `application`.

Dentro del elemento `activity` se encuentra el elemento `intent-filter` que especifica los tipos de intenciones a los que responde la actividad. El elemento `intent-filter` contiene:

- Al menos un elemento `action`. El atributo `android:name` indica a qué acciones responde esta actividad. En el caso de `Initial` este atributo toma el valor `android.intent.action.MAIN`, que identifica a esta actividad como el punto de entrada para la ejecución de nuestra aplicación. En el caso de `MainActivity` el valor es `es.uam.eps.android.CCC22.MAINACTIVITY`, que se trata de una acción definida por el programador. El nombre de dominio invertido reduce la probabilidad de colisión con otros nombres.
- Elementos `category` que contienen información adicional sobre el tipo de actividad. En nuestro caso, el nombre de la categoría del filtro es `android.intent.category.DEFAULT`. Este valor permite que otras actividades puedan arrancar a `MainActivity` con el método `startActivity()`. El valor `android.intent.category.LAUNCHER` indica que se añadirá un ícono en el menú de aplicaciones del sistema.

A continuación mostramos los ficheros `Initial.java` e `initial.xml`, correspondientes a la nueva actividad de nuestro juego:

/src/Initial.java

```
package es.uam.eps.android.CCC22;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.MotionEvent;

public class Initial extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.initial);
    }

    public boolean onTouchEvent(MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            startActivity(new Intent("es.uam.eps.android.CCC22.MAINACTIVITY"));
        }
        return true;
    }
}
```

El método `onTouchEvent()` de la actividad se ejecuta cuando el usuario toca la pantalla y ninguna otra vista gestiona el evento. Concretamente, tras comprobar que el código de acción es `ACTION_DOWN`, se instancia un objeto de la clase `Intent` al que se le pasa como argumento el nombre del filtro de la actividad que deseamos invocar: `MainActivity`. El objeto de tipo `Intent` se pasa como argumento al método `startActivity()`, el cual finalmente invoca la actividad `MainActivity`. Se debe devolver `true` cuando el evento fue gestionado y `false` en caso contrario.

Si la actividad que se invoca se encuentra en el mismo paquete que la actividad invocadora, se puede utilizar esta otra llamada a `startActivity()`:

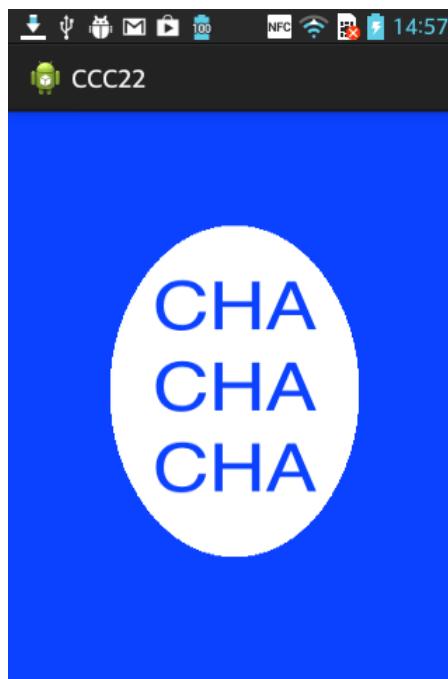
```
startActivity(new Intent(this, MainActivity.class));
```

En este caso no es necesario incluir el elemento `intent-filter` en el fichero de manifiesto. El archivo `initial.xml` utiliza un `ImageView` para mostrar una imagen que, si el usuario toca, arranca la actividad `MainActivity` como hemos visto en el fichero `Initial.java`:

```
/res/layout/initial.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:src="@drawable/initial"
        android:scaleType="fitXY"
        android:contentDescription="@string/initialImage" />
</LinearLayout>
```

El archivo `initial.png` se ha colocado en la carpeta `drawable-mdpi` del proyecto. La pantalla que se observa al arrancar la aplicación es la siguiente:



Al pulsar sobre esta pantalla se arrancará la actividad `MainActivity`, que mostrará el tablero que ya conocemos de unidades anteriores.

## 23. Arrancar aplicaciones integradas de Android

En la unidad anterior hemos aprendido a utilizar intenciones para ejecutar actividades desde otras del mismo paquete. Una de las características más destacadas de Android es que también permite llamar a actividades de otras aplicaciones. Incluso podemos utilizar intenciones para invocar y utilizar aplicaciones integradas en Android, como el navegador web o el teléfono.

Por ejemplo, para arrancar el navegador web haremos lo siguiente:

```
Intent intent = new Intent (android.content.Intent.ACTION_VIEW,
                            Uri.parse("http://www.eps.uam.es"));
startActivity(intent);
```

En lugar de utilizar una constante, también podemos pasar la acción al constructor como una cadena de caracteres:

```
Intent intent = new Intent ("android.intent.action.VIEW",
                            Uri.parse("http://www.eps.uam.es"));
startActivity(intent);
```

Como puedes ver, pasamos dos argumentos al constructor del objeto `Intent`:

- El primer argumento, `android.content.Intent.ACTION_VIEW`, es la acción, que, combinada con el `http` del segundo argumento, hace que los datos se muestren con el navegador web. Si la uri fuera de tipo `mailto:`, se abriría el gestor de correo; si fuera de tipo `tel:`, se abriría el marcador de teléfonos; si fuera de tipo `market:`, se abriría Google Play, y así sucesivamente.
- El segundo argumento corresponde a los datos. Utilizamos el método `parse()` de la clase `Uri` para convertir la cadena `http://www.eps.uam.es` en un objeto `Uri`. El siguiente código abre un nuevo correo con el destinatario indicado:

```
Intent intent = new Intent ("android.intent.action.VIEW",
                            Uri.parse("mailto:pepe@uam.es"));
startActivity(intent);
```

También podemos pasar al constructor la acción y posteriormente añadir los datos con el método `setData()`:

```
Intent intent = new Intent ("android.intent.action.VIEW");
intent.setData(Uri.parse("http://www.eps.uam.es"));
startActivity(intent);
```

A veces no es necesario especificar datos. Por ejemplo, para seleccionar un contacto de la aplicación de contactos, hay que hacer lo siguiente:

```
Intent intent = new Intent(android.content.Intent.ACTION_PICK);
intent.setType(ContactsContract.Contacts.CONTENT_TYPE);
startActivity(intent);
```

En general, un objeto de tipo `Intent` puede contener acción, datos, tipo y categoría. En la página de desarrollo de Android encontrarás más detalles al respecto.

A continuación, veamos cómo permitir que nuestro juego envíe información a nuestros contactos y nos permita solicitar la aplicación a través de la cual deseamos enviarla. Vamos a crear un proyecto nuevo CCC23 que, más tarde, cuando veamos cómo se implementan los menús, añadiremos a nuestro juego. A continuación estudiamos los archivos `MainActivity.java` y `activity_main.xml`:

/src/MainActivity.java

```
package es.uam.eps.android.CCC23;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity implements OnClickListener {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.send);
        button.setOnClickListener(this);
        button = (Button) findViewById(R.id.call);
        button.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {

        if (v.getId() == R.id.send) {
            Intent intent = new Intent(android.content.Intent.ACTION_SEND);
            intent.setType("text/plain");
            intent.putExtra(Intent.EXTRA_SUBJECT, "Cha cha cha score");
            intent.putExtra(Intent.EXTRA_TEXT,
                "Hola ..., he llegado a ... puntos en cha cha cha ...");
            startActivity(intent);
        } else if (v.getId() == R.id.call) {
            // Incluye android.permission.CALL_PHONE en el manifiesto

            Intent intent = new Intent(android.content.Intent.ACTION_CALL);
            intent.setData(Uri.parse("tel:+620254234"));
            startActivity(intent);
        }
    }
}
```

/res/layout/activity\_main.xml

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

    <Button
        android:id="@+id/send"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/sendString" />

    <Button
        android:id="@+id/call"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/callString" />

</LinearLayout>
```

Si la aplicación tiene que marcar un número directamente, es necesario añadir el permiso CALL\_PHONE al fichero `AndroidManifest.xml`:

/res/AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="es.uam.eps.android.CCC23"
    android:versionCode="1"
    android:versionName="1.0" >

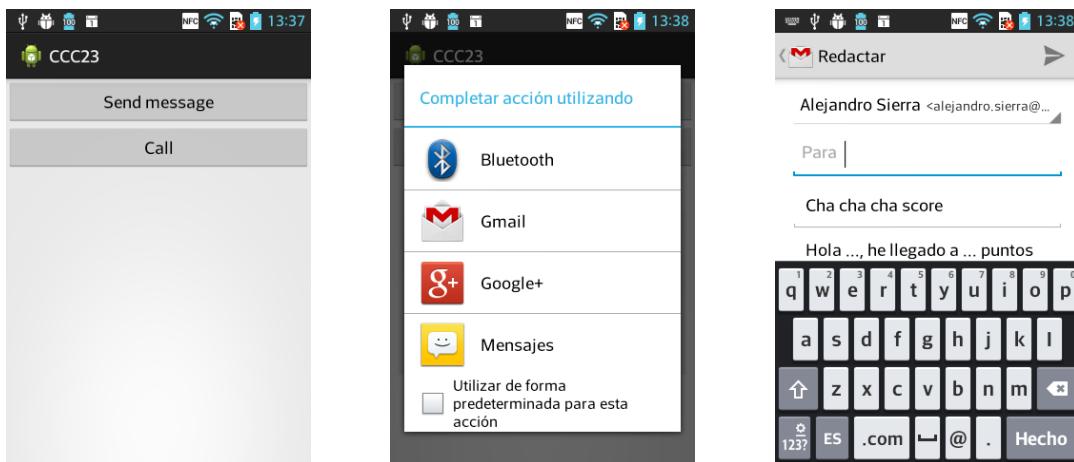
    <uses-sdk
        android:minSdkVersion="15"
        android:targetSdkVersion="19" />

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

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Esta es la sucesión de pantallas que vemos al arrancar la aplicación, pulsar el botón Send message y seleccionar Gmail:



### 23.1 Ciclo de vida de las actividades

Veamos ahora cómo se ejecutan los distintos métodos del ciclo de vida de las actividades cuando se arranca una actividad desde otra. Supongamos que tenemos una actividad principal A que arranca una actividad B mediante una intención. Este sería el proceso completo de llamadas:

```
----- Se crea la app
A onCreate()
A onStart()
A onResume()

----- Se lanza la actividad B desde A
A onSaveInstanceState()
A onPause()
B onCreate()
B onStart()
B onResume()
A onStop()

----- Se aprieta Back con la actividad B visible
B onPause()
A onRestart()
A onStart()
A onResume()
B onStop()
B onDestroy()
```

Como se puede observar, antes de lanzar y de ejecutar ningún método del ciclo de vida de la actividad B, se ejecuta el método `onPause()` de la actividad A, a continuación se ejecutan los métodos `onCreate()`, `onStart()` y `onResume()` de la actividad B, que pasa a estar visible tapando completamente la actividad A.

Una vez que la actividad B está visible, se llama al método `onStop()` de la actividad A. Este proceso es igual cuando cerramos la actividad B al apretar el botón back. Esto es, primero se pausa B, a continuación se rearranca A (`onRestart()`, `onStart()` y `onResume()`) y finalmente se para y destruye B (`onStop()` y `onDestroy()`).

Viendo estas secuencias, es importante resaltar que la velocidad con que se ejecuta el método `onPause()` de una actividad influye directamente en el tiempo que tardará la siguiente actividad en mostrarse al usuario. Por lo tanto, hay que evitar tareas en el método `onPause()` que lleven demasiado tiempo y dejarlas para el método `onStop()`, una vez que la siguiente actividad ya está en pantalla.

# Jugando con Android

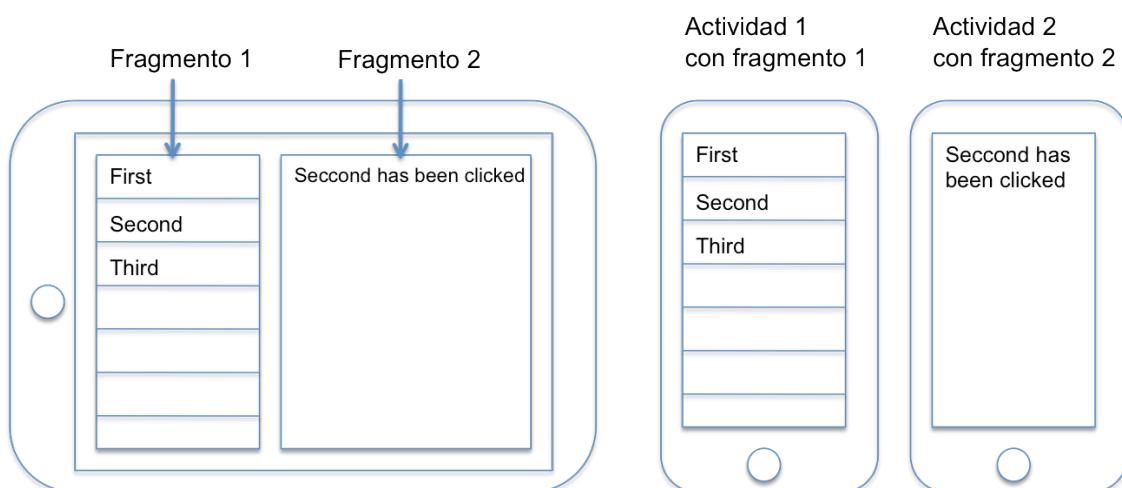
Aprende a programar tu primera App

Semana 5. Fragmentos

## 24. Fragmentos

Informalmente, puedes imaginar un fragmento como una subactividad, con su propia interfaz, comportamiento y ciclo de vida, que puedes reutilizar en distintas actividades. A pesar de su carácter modular, un fragmento siempre debe estar ligado a una actividad y su ciclo de vida depende del de la actividad a la que está ligado. Por ejemplo, cuando la actividad se destruye mediante una llamada a su método callback `onDestroy()`, el fragmento también se destruye.

Al romper el diseño de una actividad en fragmentos, su apariencia se puede modificar dinámicamente cuando sea necesario, sin la necesidad de cambios complejos de la jerarquía de vistas. Por ejemplo, un juego puede utilizar un fragmento (fragmento 1) para mostrar un menú de acciones a la izquierda y otro, fragmento 2, para mostrar la actividad correspondiente a la derecha. En una pantalla grande, ambos fragmentos pueden formar parte de la misma actividad, actividad 1. Sin embargo, en un teléfono más pequeño, necesitaremos dos actividades, una ligada al fragmento 1 del menú, y otra al 2, que se arrancará cada vez que pulsemos algún botón del fragmento 1.



**Figura 1.** Una única actividad en una pantalla grande o dos en una pantalla más pequeña.

### 24.1 La biblioteca de apoyo

Android introdujo los fragmentos en Android 3.0 (API 11) principalmente para flexibilizar el diseño de interfaces gráficas en pantallas grandes como las de las tabletas. Sin embargo, puedes utilizarlos incluso aunque el nivel mínimo del API de tu aplicación sea inferior a 11. En este caso es necesario recurrir a la biblioteca de apoyo.

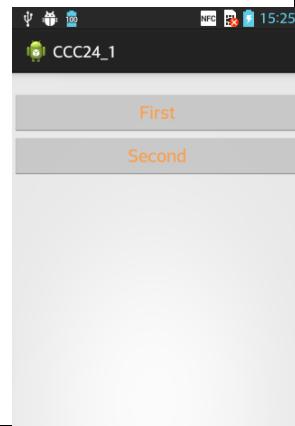
Esta biblioteca utiliza un paquete específico con algunos nombres ligeramente diferentes a los utilizados en la plataforma. Para instalar esta biblioteca debes arrancar el gestor del SDK, localizar la carpeta Extras al final del listado de paquetes, seleccionar Android Support Library, y pulsar install package. Debes asegurarte de no utilizar accidentalmente APIs nuevos en tu proyecto comprobando que tanto tus fragmentos como tus actividades provienen del paquete de apoyo:

```
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentActivity;
```

Para ilustrar la utilización de la biblioteca de apoyo veamos un ejemplo, CCC24\_1, con un nivel API mínimo igual a 8 y con un solo fragmento. Al igual que las actividades, los fragmentos necesitan de un fichero de diseño y otro Java. El fichero de diseño de nuestro primer fragmento, fragment1.xml, especifica dos botones llamados First y Second:

/res/layout/fragment1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:id="@+id/fragment1Button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:textColor="#FF9E49"
        android:text="@string/first_button_text"
        android:textSize="20sp" />
    <Button
        android:id="@+id/fragment1Button2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="#FF9E49"
        android:text="@string/second_button_text"
        android:textSize="20sp" />
</LinearLayout>
```



Los recursos first\_button\_text y second\_button\_text están definidos en strings.xml:

/res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">CCC24_1</string>
    ...
    <string name="first_button_text">First</string>
    <string name="second_button_text">Second</string>
</resources>
```

La clase del fragmento debe extender Fragment y, como mínimo, sobrescribir el método onCreateView(), que se llama cuando llega el momento de dibujar el layout del fragmento. Este método debe devolver a la actividad asociada una vista que será la raíz del diseño del fragmento. Solo existe una excepción: cuando el fragmento extiende ListFragment, la implementación por defecto devuelve un

objeto de tipo `ListView` y no es necesario implementar `onCreateView()`. También existe un método `onCreate()` pero, a diferencia de las actividades, no se utiliza para inflar la interfaz sino para configurar el fragmento.

Para implementar el ejemplo de esta sección, asegúrate de incluir la clase `Fragment` de la biblioteca de apoyo:

/src/Fragment1.java

```
package es.uam.eps.android.CCC24_1;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class Fragment1 extends Fragment {
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment1, container);
        return view;
    }
}
```

Como hemos mencionado anteriormente, el método `onCreateView()` devuelve la vista correspondiente al fragmento. El método `inflate()` de `LayoutInflater` acepta dos argumentos: el identificador del diseño del fragmento que se desea inflar (`R.layout.fragment1`) y la vista en la que se insertará el fragmento, argumento pasado por `onCreateView()`. El método `onCreateView()`, como el resto de los del ciclo de vida de los fragmentos, es público para poder ser invocado por cualquier actividad a la que se ligue el fragmento. Esto les diferencia de los métodos del ciclo de vida de la actividad.

Si bien los fragmentos pueden ser utilizados por varias actividades, siempre deben ligarse a una de ellas. Un fragmento no puede por si solo colocar una vista en la pantalla si previamente no se le ha asignado una zona de la jerarquía de vistas de una actividad. Para ligar un fragmento a una actividad podemos utilizar el fichero XML de diseño de la actividad, o bien, código Java, como veremos más adelante. En nuestro sencillo ejemplo, `MainActivity` incluye el fragmento `Fragment1` en su fichero de diseño mediante el atributo `class` de su elemento `<fragment>`:

/res/layout/activity\_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal"
    android:baselineAligned="false">
    <fragment
        android:id="@+id/fragment1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        class="es.uam.eps.android.CCC24_1.Fragment1" />
</LinearLayout>
```

Cuando el sistema infla este diseño, instancia el fragmento especificado, llama a su método `onCreateView()` e inserta la vista devuelta en el lugar indicado por el elemento `<fragment>` del archivo de diseño de la actividad. Asegúrate de extender `FragmentActivity` en lugar de `Activity`, cuando utilices la biblioteca de apoyo, pues las actividades anteriores al API 11 no saben gestionar fragmentos:

```
/src/MainActivity.java
```

```
package es.uam.eps.android.CCC24_1;

import android.os.Bundle;
import android.support.v4.app.FragmentActivity;

public class MainActivity extends FragmentActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

## 24.2 Añadiendo fragmentos a una actividad en su fichero XML

El ejemplo de esta sección utiliza dos fragmentos y prescinde de la biblioteca de apoyo. El primer fragmento, el de los dos botones, va a estar ligado a la actividad principal, y el segundo se va a ligar a una nueva actividad llamada `Detail`.

El fichero de diseño del primer fragmento, `fragment1.xml`, es el mismo que el de la sección anterior, es decir, especifica dos botones: `First` y `Second`. También reutilizaremos el fichero `Fragment1.java`, esta vez utilizando la clase `Fragment` original en lugar de la clase de la biblioteca de apoyo:

```
/src/Fragment1.java
```

```
package es.uam.eps.android.CCC24_2;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class Fragment1 extends Fragment {
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
                           savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment1, container);
        return view;
    }
}
```

El fichero de diseño del segundo fragmento, Fragment2, cuenta con un elemento de tipo TextView sobre un fondo de color naranja:

```
/res/layout/fragment2.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#FF9E49"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/fragment2TextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:textSize="20sp" />
</LinearLayout>
```

El fichero Fragment2.java es equivalente al del otro fragmento:

```
src/Fragment2.java
```

```
package es.uam.eps.android.CCC24_2;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class Fragment2 extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment2, container);
        return view;
    }
}
```

La actividad MainActivity incluye el fragmento Fragment1 en su fichero de diseño mediante el atributo class:

```
/res/layout/activity_main.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:baselineAligned="false">
    <fragment
        android:id="@+id/fragment1"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        class="es.uam.eps.android.CCC24_2.Fragment1" />
</LinearLayout>
```

El fichero de diseño de la segunda actividad, Detail, cuenta con un elemento fragment para el segundo fragmento:

```
/res/layout/detail.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:baselineAligned="false"
    android:orientation="horizontal" >
    <fragment
        android:id="@+id/fragment2"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        class="es.uam.eps.android.CCC24_2.Fragment2" />
</LinearLayout>
```

La actividad `MainActivity` infla el fichero `activity_main.xml` y extiende la clase `Activity` en lugar de la clase `FragmentActivity`. Además, implementa `OnClickListener` para dar funcionalidad a los dos botones del fragmento que aloja. Al pulsarlos se arranca la actividad `Detail` a la que además se pasa un mensaje indicando qué botón se ha pulsado:

```
/src/MainActivity.java
```

```
package es.uam.eps.android.CCC24_2;

import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.app.Activity;
import android.content.Intent;

public class MainActivity extends Activity implements OnClickListener{
    Fragment1 fragment1;
    Fragment2 fragment2;
    TextView textView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button button1 = (Button) findViewById(R.id.fragment1Button1);
        Button button2 = (Button) findViewById(R.id.fragment1Button2);

        button1.setOnClickListener(this);
        button2.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        String str;

        if (v.getId() == R.id.fragment1Button1)
            str = "First button clicked";
        else
            str = "Second button clicked";

        Intent intent = new Intent(getApplicationContext(), Detail.class);
        intent.putExtra("message", str);
        startActivity(intent);
    }
}
```

Finalmente, la actividad `Detail`, después de inflar su fichero de diseño, extrae el texto con etiqueta `message` del objeto de tipo `Intent` creado en `MainActivity` y, si no es nulo, lo muestra en la vista de tipo `TextView` de su fragmento:

```
/src/Detail.java
```

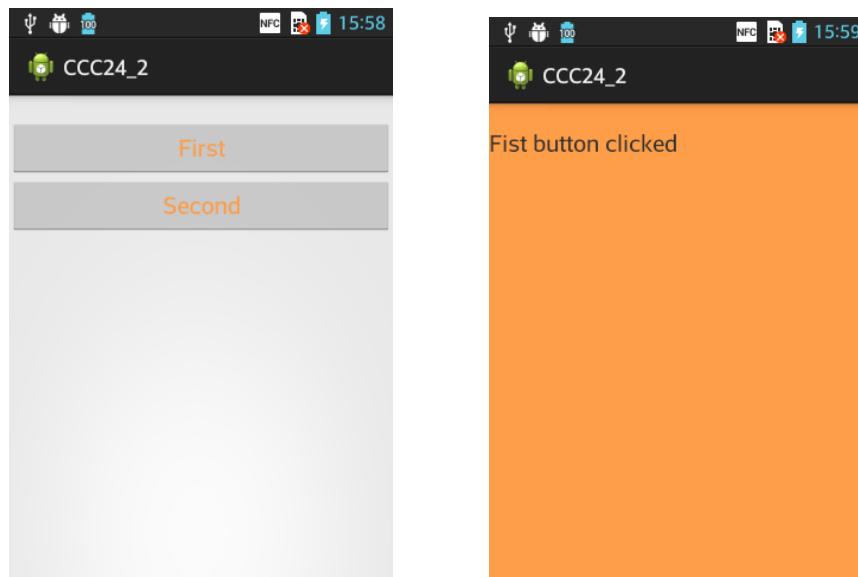
```
package es.uam.eps.android.CCC24_2;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class Detail extends Activity {
    public void onCreate (Bundle savedInstanceState){
        super.onCreate(savedInstanceState);

        setContentView(R.layout.detail);
        Bundle bundle = getIntent().getExtras();
        if (bundle != null){
            String str = bundle.getString("message");
            TextView textView = (TextView) findViewById(R.id.fragment2TextView);
            textView.setText(str);
        }
    }
}
```

No olvides añadir un elemento `activity` para la actividad `Detail` en el fichero de manifiesto. El resultado es el siguiente antes y después de pulsar el botón First:



## 24.3 Añadiendo fragmentos a una actividad durante su ejecución

Los fragmentos especificados en el fichero XML de la actividad no se pueden eliminar durante la ejecución de la app. Para crear interfaces dinámicas de este tipo es necesario añadir los fragmentos dinámicamente. En esta sección vamos a generar una interfaz similar a la de la sección anterior creando los fragmentos con código Java.

Podemos reutilizar sin cambios los ficheros XML de los dos fragmentos de la sección anterior. En los ficheros Java debemos añadir un tercer argumento al método `inflate()`, que indica si la vista inflada debe añadirse al padre (`true`) o no (`false`). Como vamos a hacerlo desde el código Java, debemos pasar `false` como tercer argumento. Por ejemplo, el primer fragmento queda así:

```
/src/Fragment1.java
```

```
package es.uam.eps.android.CCC24_3;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class Fragment1 extends Fragment {
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment1, container, false);
        return view;
    }
}
```

El fichero de diseño de la clase `MainActivity` ya no especifica el elemento de tipo `fragment`, sino simplemente un elemento de tipo `LinearLayout` vacío:

```
/res/layout/activity_main.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:baselineAligned="false">
    <LinearLayout
        android:id="@+id/fragment1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"/>
</LinearLayout>
```

El método `onCreate()` de `MainActivity` va a ser el encargado de crear el fragmento. Primero conseguimos una referencia al gestor de fragmentos que Android añadió a las actividades en el API 11 y que, entre otras cosas, mantiene:

- una lista de fragmentos identificados por los recursos `id` de sus contenedores: `R.id.fragment1` y `R.id.fragment2` en nuestra app.
- una pila de fragmentos (*back stack*) ligados a la actividad.

Para acceder al gestor llamamos al método `getFragmentManager()` o, si utilizas la biblioteca de apoyo, `getSupportFragmentManager()`:

```
FragmentManager fm = getFragmentManager();
```

Y a partir del gestor, conseguimos una referencia a `FragmentTransaction`:

```
FragmentTransaction ft = fragmentManager.beginTransaction();
```

A continuación se instancia el fragmento y se añade mediante el método `add()`, que tiene dos argumentos: el identificador del contenedor del fragmento (`R.id.fragment1`) y el fragmento que se desea añadir (`fragment1`):

```
Fragment1 fragment1 = new Fragment1();
ft.add(R.id.fragment1, fragment1);
```

Para que los cambios tengan lugar es necesario ejecutar el método `commit()` de la transacción:

```
ft.commit();
```

Antes de añadir los fragmentos conviene comprobar si ya se encuentran en la pila. Cuando la actividad se destruye, por rotación por ejemplo, la lista de fragmentos se guarda para ser recreados cuando se recree la actividad. Esta es una razón por la que ya podríamos tener a los fragmentos en la pila. Por lo tanto, para evitar instanciar fragmentos innecesariamente, utilizaremos el siguiente código:

```
FragmentManager fm = getFragmentManager();
if (fm.findFragmentById(R.id.fragment1) == null) {
    Fragment1 fragment1 = new Fragment1();
    fm.beginTransaction().add(R.id.fragment1, fragment1).commit();
}
```

Hemos utilizado el hecho de que el método `add()` devuelve una referencia a la transacción. El fichero `MainActivity.java` es el siguiente:

```
/src/MainActivity.java
```

```
package es.uam.eps.android.CCC24_3;

import android.app.Activity;
import android.app.FragmentManager;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity implements OnClickListener{

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        FragmentManager fm = getFragmentManager();
        if (fm.findFragmentById(R.id.fragment1) == null) {
            Fragment1 fragment1 = new Fragment1();
            fm.beginTransaction().add(R.id.fragment1, fragment1).commit();
        }
    }

    @Override
    protected void onStart() {
        super.onStart();

        Fragment1 fragment = (Fragment1)
            getFragmentManager().findFragmentById(R.id.fragment1);

        Button button1 = (Button)
            fragment.getView().findViewById(R.id.fragment1Button1);
        Button button2 = (Button)
            fragment.getView().findViewById(R.id.fragment1Button2);
    }
}
```

```

        button1.setOnClickListener(this);
        button2.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        String str;

        if (v.getId() == R.id.fragment1Button1)
            str = "First button clicked";
        else
            str = "Second button clicked";

        Intent intent = new Intent ("es.uam.eps.android.CCC24_3.DETAIL");
        intent.putExtra("message", str);
        startActivity(intent);
    }
}

```

En el método `onStart()` es donde se captura una referencia al primer fragmento mediante el método `findFragmentById()`:

```
Fragment1 fragment = (Fragment1) getSupportFragmentManager().findFragmentById(R.id.fragment1);
```

A continuación se recuperan sendas referencias a sus dos botones, para registrar la actividad como escuchador. Fíjate en que los fragmentos no cuentan con un método `findViewById()` sino que debemos utilizar el de la clase `View`. Para ello primero recuperamos la vista del fragmento con el método `getView()`. También podemos acceder a la actividad ligada al fragmento mediante el método `getActivity()` y así, por ejemplo, acceder a una vista determinada de su diseño:

```
TextView textView = (TextView) getActivity().findViewById(R.id.textView);
```

Finalmente, el método `onClick()` lleva a cabo la misma tarea que el método `onClick()` del ejemplo anterior, es decir, arrancar la actividad `Detail`.

La actividad `Detail`, por su parte, crea el fragmento número 2 como sigue:

/src/Detail.java

```

package es.uam.eps.android.CCC24_3;

import android.app.Activity;
import android.app.FragmentManager;
import android.os.Bundle;
import android.widget.TextView;

public class Detail extends Activity {
    public void onCreate (Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.detail);

        FragmentManager fm = getFragmentManager();
        if (fm.findFragmentById(R.id.fragment2) == null) {
            Fragment2 fragment2 = new Fragment2();
            fm.beginTransaction().add(R.id.fragment2, fragment2).commit();
        }
    }

    @Override
    protected void onStart (){
        super.onStart();

        Bundle bundle = getIntent().getExtras();
        if (bundle != null){

```

```

        String str = bundle.getString("message");
        TextView textView = (TextView) findViewById(R.id.fragment2TextView);
        textView.setText(str);
    }
}

```

Además del método `add()`, también podemos utilizar los métodos `replace()` y `remove()`. El método `addToBackStack()` permite añadir la transacción a la pila de transacciones (*back stack*). La actividad, a través del gestor de fragmentos, gestiona esta pila que permite al usuario volver a estados anteriores pulsando el botón back del dispositivo. Los fragmentos de las transacciones de la *back stack* se detienen y luego reanudan al pulsar el botón back, en lugar de destruirse directamente.

## 24.4 Comunicación con la actividad a través de una interfaz

Veamos una forma de conseguir el mismo resultado de la sección anterior sin que los fragmentos se comuniquen directamente, manteniendo así su independencia. El proyecto CCC24\_4 comienza modificando la clase `Fragment2.java` al añadir el método `showText()`, que se encarga de mostrar la cadena pasada como argumento en el `TextView` del segundo fragmento:

/src/Fragment2.java

```

package es.uam.eps.android.CCC24_4;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class Fragment2 extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment2, container, false);
        return view;
    }

    public void showText(String text) {
        TextView view = (TextView) getView().findViewById(R.id.fragment2TextView);
        view.setText(text);
    }
}

```

El primer fragmento define la interfaz `onButtonSelectedListener`

```

public interface OnButtonSelectedListener {
    public void onButtonSelected(String link);
}

```

La interfaz contiene un único método, `onButtonSelected()`, que tendrá que implementar la actividad en la que se incoruste el fragmento. Esta es una solución que evita que los fragmentos se comuniquen directamente, manteniendo así su independencia:

### /src/Fragment1.java

```
package es.uam.eps.android.CCC24_4;

import android.app.Activity;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.Button;

public class Fragment1 extends Fragment {
    private OnButtonSelectedListener listener;

    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
                           savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment1, container, false);

        Button button1 = (Button)view.findViewById(R.id.fragment1Button1);
        Button button2 = (Button)view.findViewById(R.id.fragment1Button2);

        button1.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                listener.onButtonSelected("First button clicked");
            }
        });

        button2.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                listener.onButtonSelected("Second button clicked");
            }
        });
        return view;
    }

    public interface OnButtonSelectedListener {
        public void onButtonSelected (String str);
    }

    @Override
    public void onAttach (Activity activity){
        super.onAttach(activity);
        if (activity instanceof OnButtonSelectedListener)
            listener = (OnButtonSelectedListener) activity;
        else {
            throw new ClassCastException(activity.toString() +
                " does not implement OnButtonSelectedListener");
        }
    }

    @Override
    public void onDetach(){
        super.onDetach();
        listener = null;
    }
}
```

Asignamos la actividad como escuchador en el método que se ejecuta cuando el fragmento se liga a la actividad: `onAttach()`. Este método lanza una excepción si la actividad pasada como argumento no implementa la interfaz. En el método `onDettach()` se asigna el escuchador a `null`.

La clase `MainActivity.java` se encarga de crear el primer fragmento con una transacción y de implementar la interfaz definida en el primer fragmento, es decir, implementar el método `onButtonSelected()`. Este método arranca la actividad `Detail` con un extra de clave `message`:

```
/src/MainActivity.java
```

```
package es.uam.eps.android.CCC24_4;

import android.app.Activity;
import android.app.FragmentManager;
import android.content.Intent;
import android.os.Bundle;

import es.uam.eps.android.CCC24_4.Fragment1.OnButtonSelectedListener;

public class MainActivity extends Activity implements OnButtonSelectedListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        FragmentManager fm = getFragmentManager();
        if (fm.findFragmentById(R.id.fragment1) == null) {
            Fragment1 fragment1 = new Fragment1();
            fm.beginTransaction().add(R.id.fragment1, fragment1).commit();
        }
    }

    @Override
    public void onButtonSelected(String str) {
        Intent intent = new Intent(getApplicationContext(), Detail.class);
        intent.putExtra("message", str);
        startActivity(intent);
    }
}
```

La actividad `Detail` crea el fragmento número 2 y, en el método `onStart()`, captura una referencia al fragmento y pasa al método `showText()` el mensaje extraído del objeto de tipo `Intent`:

```
/src/Detail.java
```

```
package es.uam.eps.android.CCC24_4;

import android.app.Activity;
import android.app.FragmentManager;
import android.os.Bundle;

public class Detail extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.detail);

        FragmentManager fm = getFragmentManager();
        if (fm.findFragmentById(R.id.fragment2) == null) {
            Fragment2 fragment2 = new Fragment2();
            fm.beginTransaction().add(R.id.fragment2, fragment2).commit();
        }
    }

    @Override
    protected void onStart() {
        super.onStart();

        Bundle bundle = getIntent().getExtras();
```

```

        if (bundle != null){
            String str = bundle.getString("message");
            Fragment2 fragment2 = (Fragment2)
                getFragmentManager().findFragmentById(R.id.fragment2);
            fragment2.showText(str);
        }
    }
}

```

## 24.5 Diseño para tabletas

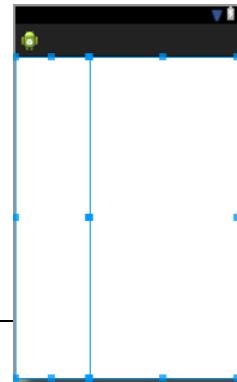
Hasta ahora, nuestra app muestra cada fragmento en una actividad diferente: `MainActivity` y `Detail`. En una tableta ambos fragmentos se pueden visualizar simultáneamente ligándolos a la actividad principal. Vamos a crear un último proyecto, `CCC24_5`, con un fichero de diseño con dos contenedores para fragmentos:

`/res/layout/activity_tablet.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:baselineAligned="false">
    <LinearLayout
        android:id="@+id/fragment1"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:orientation="vertical" />
    <LinearLayout
        android:id="@+id/fragment2"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:orientation="vertical" />
</LinearLayout>

```



Además, añadiremos un recurso alias de nombre `activity_master_detail` y tipo `layout` que inflará la actividad principal. Un recurso alias es un recurso que apunta a otro recurso. En este caso el alias apuntará a `activity_main.xml` en teléfonos o `activity_tablet.xml` en tabletas. Los recursos alias se colocan en la carpeta `/res/values/` dentro del fichero `refs.xml`:

`/res/values/refs.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <item name="activity_master_detail" type="layout">@layout/activity_main</item>
</resources>

```

Para que el recurso apunte a `activity_tablet` en tabletas, crearemos una versión alternativa de `refs.xml` en la carpeta `/res/values-sw600dp`:

```
/res/values-sw600dp/refs.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <item name="activity_master_detail" type="layout">@layout/activity_tablet</item>
</resources>
```

La actividad principal infla el fichero apuntado por el recurso alias de nombre activity\_master\_detail.xml y, además, antes de crear el fragmento de tipo Fragment2, comprueba que la anchura en dps es superior a 600, de tal forma que solo se instancie en tabletas:

```
/src/MainActivity.java
```

```
package es.uam.eps.android.CCC24_5;

import android.app.Activity;
import android.app.FragmentManager;
import android.content.Intent;
import android.os.Bundle;
import android.util.DisplayMetrics;
import android.view.Display;

import es.uam.eps.android.CCC24_5.Fragment1.OnButtonSelectedListener;

public class MainActivity extends Activity implements OnButtonSelectedListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_master_detail);

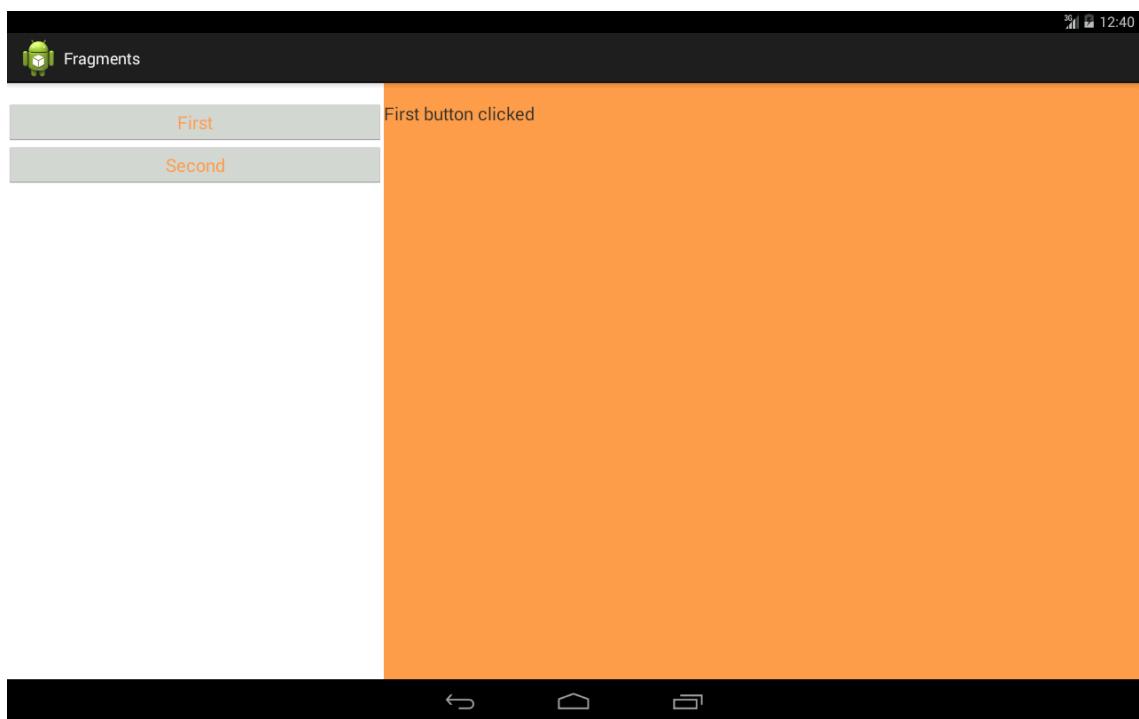
        Display display = getWindowManager().getDefaultDisplay();
        DisplayMetrics outMetrics = new DisplayMetrics ();
        display.getMetrics(outMetrics);
        float density   = getResources().getDisplayMetrics().density;
        float dpWidth   = outMetrics.widthPixels / density;

        FragmentManager fm = getFragmentManager();
        if (fm.findFragmentById(R.id.fragment1) == null) {
            Fragment1 fragment1 = new Fragment1();
            fm.beginTransaction().add(R.id.fragment1, fragment1).commit();
        }

        if (dpWidth > 600){
            if (fm.findFragmentById(R.id.fragment2) == null) {
                Fragment2 fragment2 = new Fragment2();
                fm.beginTransaction().add(R.id.fragment2, fragment2).commit();
            }
        }
    }

    @Override
    public void onButtonSelected(String str) {
        Fragment2 fragment = (Fragment2) getFragmentManager().findFragmentById(
                R.id.fragment2);
        if (fragment != null) {
            fragment.showText(str);
        } else {
            Intent intent = new Intent(getApplicationContext(), Detail.class);
            intent.putExtra("message", str);
            startActivity(intent);
        }
    }
}
```

Ahora, este es el resultado en una tableta:



## 25. Menús

Los menús muestran gráficamente opciones que, al ser seleccionadas, arrancan la tarea correspondiente. Básicamente, existen dos tipos de menús en Android:

- El menú de **opciones** de la actividad actual que se activa cuando pulsas el botón MENU. Las opciones se pueden mostrar como texto, casillas de selección (check boxes) o botones de selección (radio buttons).
- El menú de **contexto** de una vista concreta, que se muestra cuando mantienes la pulsación sobre la vista. Las opciones se pueden mostrar como casillas de selección o botones de selección. Este menú también permite añadir teclas de atajo (shortcutkeys) y submenús.

Un submenú se muestra como una ventana flotante que aparece cuando se pulsa una cierta opción del menú que lo lleva aparejado.

Para mostrar el menú de opciones de una actividad determinada hay que implementar dos métodos de la actividad:

- `onCreateOptionsMenu()`: este método se ejecuta cuando pulsas el botón MENU. Aquí debes suministrar código de interfaz de usuario.
- `onOptionsItemSelected()`: este método se ejecuta cuando se selecciona un ítem del menú creado en `onCreateOptionsMenu()`. El código del método debe llevar a cabo la tarea correspondiente al ítem seleccionado.

El proyecto de esta unidad, CCC25, añade un menú de opciones a nuestro juego. Recordemos que nuestro juego cuenta ya con cuatro clases: Game, Initial, MainActivity y Music, como puedes comprobar en el proyecto CCC22. Vamos a añadir la clase About y a modificar MainActivity para incluir el menú de opciones que va a permitirnos enviar mensajes como vimos en la unidad 23 y mostrar información sobre nuestra aplicación:

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="es.uam.eps.android.CCC25"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="15"
        android:targetSdkVersion="19" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".Initial"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

```

</activity>
<activity
    android:name=".MainActivity"
    android:label="" >
    <intent-filter>
        <action android:name="es.uam.eps.android.CCC25.MAINACTIVITY" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
<activity
    android:name=".About"
    android:theme="@android:style/Theme.Holo.Dialog"
    android:label="" >
    <intent-filter>
        <action android:name="es.uam.eps.android.CCC25.ABOUT" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
</application>

</manifest>

```

Una forma sencilla de especificar la interfaz de usuario del menú es crear un fichero XML y colocarlo en la carpeta /res/menu (añádela a res pulsando con el botón derecho del ratón sobre la carpeta res):

/res/menu/ccc\_menu.xml

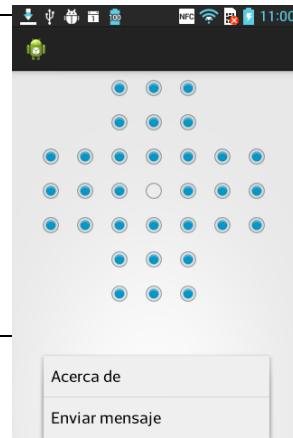
```

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

    <item
        android:id="@+id/menuAbout"
        android:icon="@android:drawable/ic_menu_info_details"
        android:title="@string/aboutText"/>
    <item
        android:id="@+id/sendMessage"
        android:icon="@android:drawable/ic_dialog_email"
        android:title="@string/sendMessageText"/>

</menu>

```



El elemento `menu` corresponde al conjunto del menú y cada elemento `item` especifica un ítem del menú con las siguientes propiedades:

- título (`android:title`)
- identificador (`android:id`)
- icono (`android:icon`)

El título recibe un valor mediante un recurso especificado en el fichero `strings.xml` del que, mas adelante, haremos una versión en inglés para internacionalizar el juego:

/res/values/strings.xml

```

<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">CCC25</string>
    <string name="action_settings">Ajustes</string>
    <string name="gameOverTitle">Fin del juego</string>
    <string name="initialImage">Imagen inicial del juego</string>
    <string name="aboutText">Acerca de</string>
    <string name="sendMessageText">Enviar mensaje</string>

```

```

<string name="searchText">Buscar</string>
<string name="searchByNameText">Buscar por nombre</string>
<string name="searchByBirthDateText">Buscar por fecha de nacimiento</string>
<string name="searchByLastNameText">Buscar por apellido</string>
<string name="aboutMessage">Chacha es un juego solitario
    en el que hemos de comer una a una las fichas
    saltando por encima de ellas, solo en vertical u
    horizontal, hacia la casilla adyacente que se
    encuentre vacía. El objetivo es dejar
    la última ficha en el centro del tablero, el
    lugar que comenzó libre.</string>
</resources>

```

La interfaz gráfica del menú se infla en el método `onCreateOptionsMenu()` de la clase `MainActivity`:

`/src/MainActivity.java`

```

...
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.ccc_menu, menu);
    return true;
} ...

```

La tarea ejecutada al seleccionar un ítem determinado se codifica en el método `onOptionsItemSelected()`:

`/src/MainActivity.java`

```

...
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menuAbout:
            startActivity(new Intent(this, About.class));
            return true;
        case R.id.sendMessage:
            Intent intent = new Intent(android.content.Intent.ACTION_SEND);
            intent.setType("text/plain");
            intent.putExtra(Intent.EXTRA_SUBJECT, "CHA CHA CHA");
            intent.putExtra(Intent.EXTRA_TEXT, "Hola ..., he llegado a ... puntos en cha
cha cha ...");
            startActivity(intent);
            return true;
    }
    return super.onOptionsItemSelected(item);
} ...

```

Por ejemplo, si el usuario pulsa el ítem `About`, se ejecuta la actividad `About`:

```
startActivity(new Intent(this, About.class));
```

El método `onCreate()` de `About` infla el interfaz especificado en `about.xml`:

`/src/About.java`

```

package es.uam.eps.android.CCC25;

import android.app.Activity;
import android.os.Bundle;

public class About extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.about);
    }
}

```

El fichero `about.xml` es el siguiente:

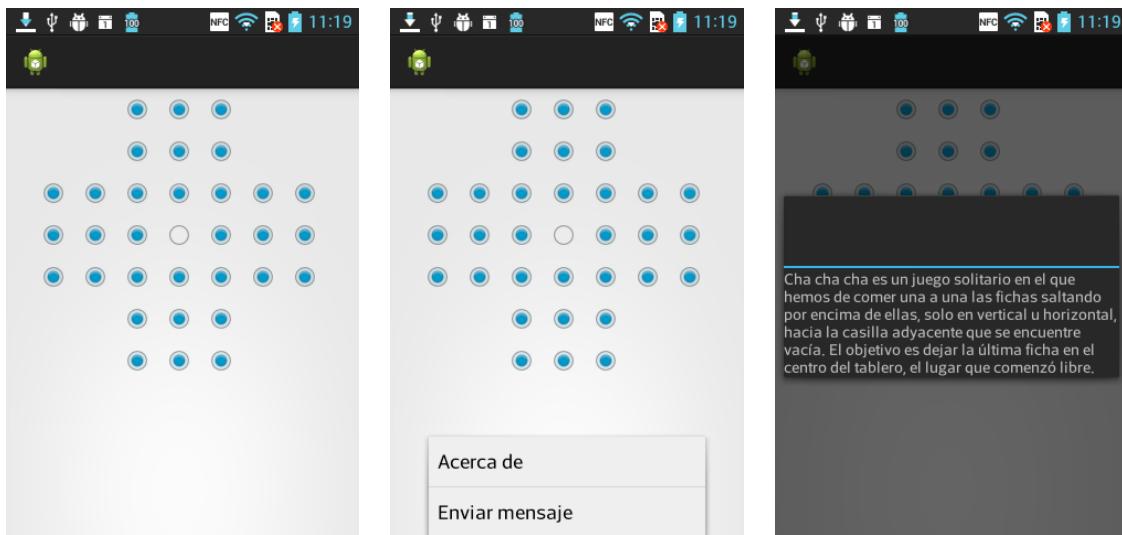
```
/res/layout/about.xml
```

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

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/aboutMessage" />
</LinearLayout>
```

A continuación se muestra un ejemplo de sucesión de acciones hasta ejecutar About:

- Durante la ejecución de `MainActivity` pulsamos la tecla del menú de opciones.
- Aparece el menú de opciones.
- Pulsamos el ítem Acerca de.
- Aparece una ventana ligera mostrando las reglas del juego.



Como ves, la actividad `About` no tapa por completo al tablero sino que se muestra como si fuera una ventana de diálogo gracias al tema especificado en el fichero de manifiesto:

```
<activity
    android:name=".About"
    android:theme="@android:style/Theme.Holo.Dialog"
    android:label="" >
    <intent-filter>
        <action android:name="es.uam.eps.android.ccc25.ABOUT" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

## 25.1 Submenús

Para programar un submenú basta anidar un elemento `menu` dentro de uno de los ítems del fichero de diseño. Por ejemplo, añadamos un submenú a un nuevo ítem del menú anterior que llamaremos Search. El nuevo fichero de diseño es el siguiente:

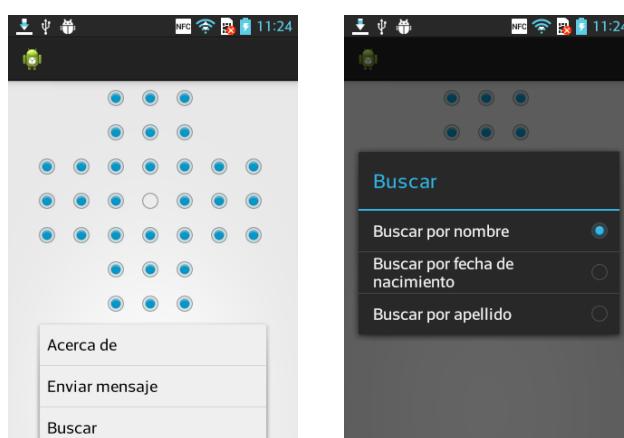
/res/menu/ccc\_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:title="@string/aboutText"
        android:id="@+id/menuAbout"
        android:icon="@android:drawable/ic_menu_info_details"/>
    <item
        android:title="@string/sendMessageText"
        android:id="@+id/sendMessage"
        android:icon="@android:drawable/ic_dialog_email"/>
    <item
        android:title="@string/searchText"
        android:id="@+id/menu_search"
        android:icon="@android:drawable/ic_menu_search">
        <menu>
            <group android:checkableBehavior="single">
                <item
                    android:id="@+id/search_by_name"
                    android:title="@string/searchByNameText"
                    android:checked="true"/>
                <item
                    android:id="@+id/search_by_birth_date"
                    android:title="@string/searchByBirthDateText"/>
                <item
                    android:id="@+id/search_by_last_name"
                    android:title="@string/searchByLastNameText"/>
            </group>
        </menu>
    </item>
</menu>
```

El elemento `<group>` sirve para agrupar opciones y aplicarles ciertos atributos colectivamente. Por ejemplo, `android:checkableBehavior` sirve para especificar si los ítems deben aparecer como ítems sencillos, casillas o botones. Sus valores pueden ser:

- `single`: botones donde solo un ítem se puede seleccionar
- `all`: casillas donde se pueden seleccionar varios ítems
- `none`: cada ítem se muestra como texto sin casillas ni botones.

Esto es lo que vemos al pulsar la tecla del menú de opciones y luego la opción Buscar:



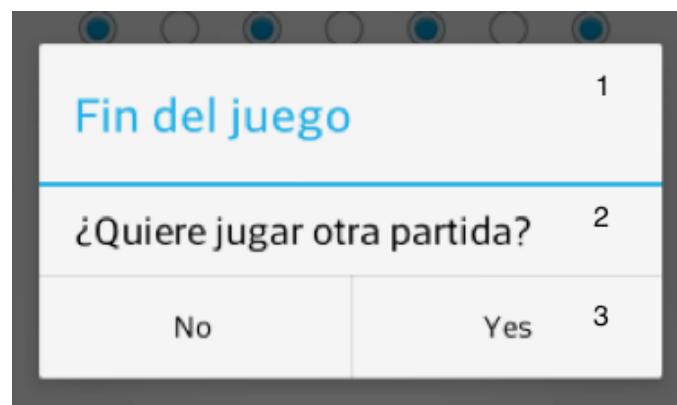
## 26. Diálogos

No solo podemos interactuar con los usuarios a través de actividades y menús con opciones. Cuando de lo que se trata es de pedir al usuario confirmación o de mostrar mensajes de alerta o error, los diálogos son preferibles pues se muestran como ventanas flotantes, más ligeras que una actividad. Mientras el diálogo está abierto, los usuarios solo pueden interactuar con sus botones, si bien la actividad que lo contiene sigue ejecutándose normalmente.

El SDK proporciona los siguientes tipos de diálogos:

- `Dialog`: clase base de todos los diálogos.
- `AlertDialog`: diálogo que muestra uno, dos o tres botones.
- `CharacterPickerDialog`: diálogo que permite seleccionar caracteres acentuados, por ejemplo.
- `DatePickerDialog`: diálogo que permite al usuario seleccionar y fijar una fecha.
- `TimePickerDialog`: diálogo que permite al usuario seleccionar y fijar una hora.
- `ProgressDialog`: diálogo que muestra un indicador del desarrollo de una actividad.

Un diálogo de alerta está compuesto por tres regiones como se muestra en la siguiente figura:



1. El **título** (1 en la figura) es opcional y solo se debe poner cuando el área de contenido, marcada como 2 en la figura, contiene un mensaje detallado, una lista u otras vistas añadidas por el programador. Cuando solo necesitamos mostrar un mensaje, basta con una alerta sin título.
2. El **área de contenido** (2 en la figura) puede contener un mensaje, una lista u otras vistas añadidas por el programador.
3. Los **botones de acción** (3 en la figura), que no pueden ser más de tres.

En el proyecto CCC26 vamos a añadir a nuestro solitario un diálogo de alerta al final de cada partida para preguntar al jugador si desea echar otra o no. Para ello extenderemos la clase `DialogFragment`, que es un fragmento que muestra una ventana de diálogo flotando sobre la ventana de la actividad ligada al fragmento. Este fragmento contiene un objeto de tipo `Dialog`, cuya gestión debe hacerse a través del fragmento en lugar de hacer llamadas directas al diálogo. El código de la clase `AlertDialogFragment` es el siguiente:

/src/AlertDialogFragment.java

```
package es.uam.eps.android.CCC26;

import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
import android.content.DialogInterface;
import android.os.Bundle;

public class AlertDialogFragment extends DialogFragment{
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        final MainActivity main = (MainActivity) getActivity();

        AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(getActivity());
        alertDialogBuilder.setTitle(R.string.gameOverTitle);
        alertDialogBuilder.setMessage(R.string.gameOverMessage);
        alertDialogBuilder.setPositiveButton("Yes",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                    main.game.restart();
                    main.setFigureFromGrid();
                    dialog.dismiss();
                }
            });
        alertDialogBuilder.setNegativeButton("No",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                    main.finish();
                    dialog.dismiss();
                }
            });
        return alertDialogBuilder.create();
    }
}
```

Es frecuente sobrescribir el método `onCreateDialog()` cuando se trata de generar diálogos adaptados por el programador. Dentro de este método callback, la forma más sencilla de generar el diálogo es instanciar un objeto de tipo `AlertDialog.Builder` y llamar a sus métodos `set` como sigue:

- `setTitle()` asigna el título de la cabecera de la ventana: Fin del juego.
- `setMessage()` asigna el mensaje debajo del título en el área de contenido: ¿Quiere jugar otra partida?
- `setPositiveButton()` asigna un escuchador cuyo método `onClick()` se ejecuta cuando se pulsa el botón positivo del diálogo. En nuestro proyecto, `onClick()` llama a `restart()` de la clase `Game`, que reinicia el grid, y a `setFigureFromGrid()` de `MainActivity` para redibujar el tablero. Finalmente se elimina el diálogo.
- `setNegativeButton()` asigna un escuchador cuyo método `onClick()` se ejecuta cuando se pulsa el botón negativo del diálogo. En nuestro proyecto,

`onClick()` termina la actividad y elimina el diálogo, con lo que volvemos a la actividad `Initial`.

La visibilidad de `setFigureFromGrid()` ha de ser `public` para que este código funcione. Los recursos de `strings.xml` nos van a permitir más adelante adaptar los textos de los diálogos al idioma especificado en el dispositivo:

```
/res/values/strings.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

<string name="app_name">CCC26</string>
<string name="action_settings">Ajustes</string>
...
<string name="gameOverTitle">Fin del juego</string>
<string name="gameOverMessage">¿Quiere jugar otra partida?</string>
</resources>
```

La actividad `MainActivity` se encarga de instanciar y mostrar el fragmento donde antes se mostraba el mensaje mediante la clase `Toast`. Concretamente, si el juego ha terminado se invoca el método `show()`:

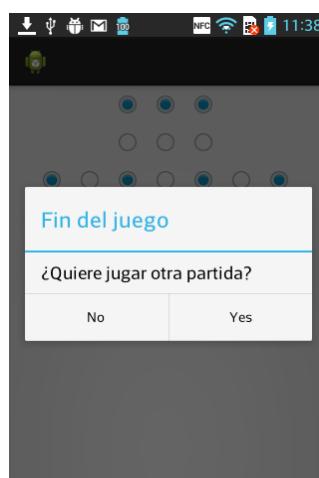
```
/src/MainActivity.java
```

```
...
public void onClick (View v){
    int id = ((RadioButton) v).getId();

    for (int i=0; i<SIZE; i++)
        for (int j=0; j<SIZE; j++)
            if (ids[i][j] == id) {
                game.play(i, j);
                break;
            }

    setFigureFromGrid();
    if (game.isGameFinished()) {
        new AlertDialogFragment () .show(getFragmentManager(), "ALERT_DIALOG");
    }
} ...
```

El método `show()` tiene dos argumentos: el `FragmentManager` al que añadimos el fragmento y el TAG con el que se añade. Aunque no es obvio, en realidad, lo que la actividad está haciendo es una transacción como las explicadas en la unidad 24 (que no se añade al back stack, por cierto). Este es el resultado al final de una partida:



## 27. Preferencias

Android dispone de un mecanismo para almacenar pequeñas cantidades de datos en forma de pares clave/valor: la clave ha de ser de tipo `String` y el valor de uno de los siguientes tipos: `Boolean`, `Integer`, `Long`, `Float`, `String`, o `set<String>` (un conjunto de valores de tipo `String` desde el API 11). El almacenamiento es persistente, es decir, los datos no se pierden aunque se detenga la app o se apague el dispositivo.

### 27.1 Lectura y escritura de preferencias

Los datos pueden ser privados de la actividad o pueden compartirse entre todas las actividades de una aplicación. En este último caso las preferencias se almacenan por defecto en un archivo, que agrega el sufijo `_preferences.xml` al nombre del paquete. Para el proyecto `CCC27` de esta unidad, el archivo de preferencias es:

```
es.uam.eps.android.CCC27_preferences.xml
```

En cuanto se modifique el valor de una de las preferencias, el archivo se añade a la siguiente carpeta:

```
/data/data/es.uam.eps.android.CCC27/shared_prefs
```

Veamos cómo recuperar el valor de una de estas preferencias sabiendo que su clave es `music`, por ejemplo. La forma más sencilla es utilizar el siguiente método desde cualquier actividad:

```
public Boolean music(){
    Boolean play = false;

    SharedPreferences sharedpreferences =
        PreferenceManager.getDefaultSharedPreferences(this);
    if (sharedpreferences.contains(music))
        play = sharedpreferences.getBoolean(music, false);

    return play;
}
```

Primero conseguimos una instancia de la clase `SharedPreferences` especificando el contexto (`this`). A continuación, siempre que exista la preferencia en el archivo, recuperamos su valor booleano mediante el método `getBoolean()`, al que pasamos la clave que identifica la preferencia como primer argumento, y su valor por omisión como segundo argumento. Más adelante situaremos todas las claves en una única clase para mejorar la organización.

Por otro lado, para modificar el valor de una preferencia compartida utilizaremos un método como el siguiente:

```

public void setMusic (Boolean value) {
    SharedPreferences preferences =
        PreferenceManager.getDefaultSharedPreferences(this);
    SharedPreferences.Editor editor = preferences.edit();
    editor.putBoolean(music, value);
    editor.commit();
}

```

Primero conseguimos una instancia de la clase `SharedPreferences` especificando el contexto (`this`). A continuación creamos un objeto de tipo `Editor` mediante el método `edit()`. Utilizamos el método `putBoolean()` para cambiar el valor de preferencias de tipo `Boolean`. Finalmente, para guardar los cambios en el fichero de preferencias, utilizamos el método `commit()`.

## 27.2 El menú de preferencias

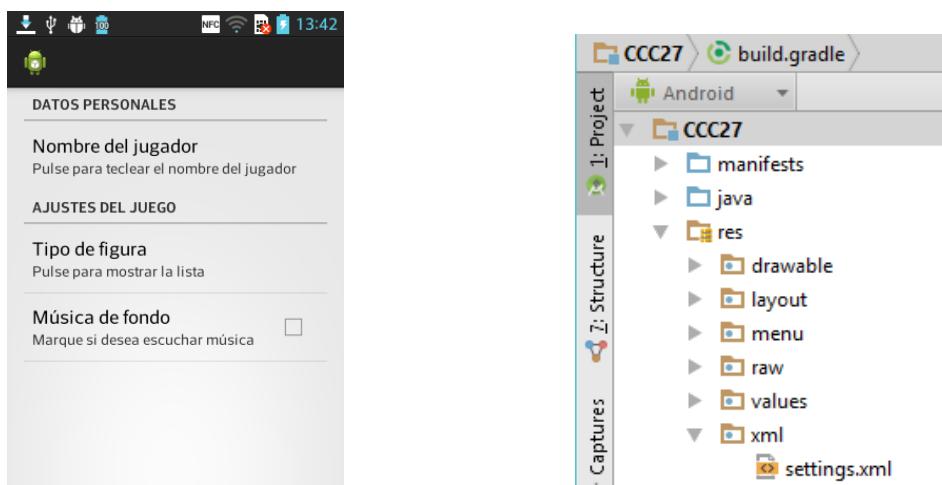
Veamos ahora cómo solicitar en nuestro proyecto, `ccc27`, el valor de ciertas preferencias (nombre del jugador, tipo de figura inicial y música) mediante una interfaz de usuario especificada en un fichero XML que colocaremos en una subcarpeta de nombre `xml` dentro de la carpeta `res` (estudia la interfaz gráfica a la que da lugar el fichero):

`/res/xml/settings.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory android:title="Datos personales">
        <EditTextPreference
            android:key="playerName"
            android:title="Nombre del jugadro"
            android:summary="Pulse para teclear el nombre del jugador" />
    </PreferenceCategory>
    <PreferenceCategory android:title="Ajustes del juego">
        <ListPreference
            android:key="figure"
            android:title="Tipo de figura"
            android:summary="Pulse para mostrar la lista"
            android:entries="@array/figures"
            android:entryValues="@array/figureCodes"
            android:dialogTitle="Elija la figura inicial"
            android.defaultValue="c0011100001100111111111011111111100111000011100" />
        <CheckBoxPreference
            android:key="music"
            android:title="Música de fondo"
            android:summary="Marque si desea escuchar música"
            android.defaultValue="false" />
    </PreferenceCategory>
</PreferenceScreen>

```



El elemento `PreferenceScreen` del fichero `settings.xml` corresponde al menú de preferencias dentro del cual se especifican otros elementos como `ListPreference`, `CheckBoxPreference` y `EditTextPreference`. Todos estos elementos se corresponden con vistas normales de Android a las que hemos añadido el sufijo `Preference`.

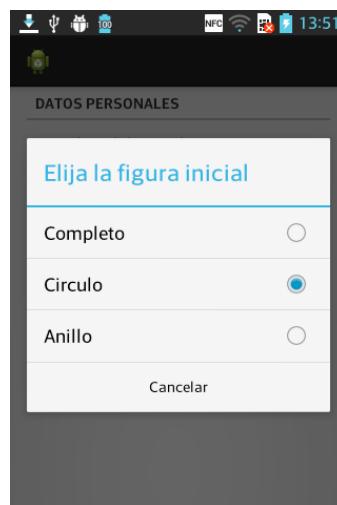
El atributo `android:key` es la clave de la preferencia, que utilizaremos en el código Java para recuperar su valor. El significado de `android:title` y `android:summary` queda claro después de observar la interfaz gráfica de la página anterior.

Dentro del elemento `ListPreference`, los atributos `android:entries` y `android:entry_values` especifican recursos correspondientes a las cadenas y códigos, respectivamente, de las distintas figuras iniciales. Estos recursos se especifican en el fichero `arrays.xml`:

```
/res/values/arrays.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="figures">
        <item>Completo</item>
        <item>Circulo</item>
        <item>Anillo</item>
    </string-array>
    <string-array name="figureCodes">
        <item>c0011000011001111111101111111100111000011100</item>
        <item>c0000000001100011110011011001111000111000000000</item>
        <item>c000000000110001101100100010011011000111000000000</item>
    </string-array>
</resources>
```

Estos valores se necesitan para construir la lista que aparece cuando se pulsa la preferencia titulada “Tipo de figura”. La cabecera del diálogo que surge contiene la cadena especificada en el atributo `android:DialogTitle` del elemento de tipo `ListPreference`:



En el código del proyecto, las cadenas del fichero `settings.xml`, como “Tipo de figura” se sustituyen por recursos (`@string/listPrefTitle`) que facilitarán más adelante la internacionalización:

/res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">CCC27</string>
    <string name="action_settings">Ajustes</string>
    <string name="gameOverTitle">Fin del juego</string>
    <string name="gameOverMessage">¿Quiere jugar otra partida?</string>
    <string name="initialImage">Imagen inicial del juego</string>
    <string name="aboutText">Acerca de</string>
    <string name="sendMessageText">Enviar mensaje</string>
    <string name="aboutMessage">Chacha es un juego solitario
        en el que hemos de comer una a una las fichas
        saltando por encima de ellas, solo en vertical u
        horizontal, hacia la casilla adyacente que se
        encuentre vacía. El objetivo es dejar
        la última ficha en el centro del tablero, el
        lugar que comenzó libre.</string>
    <string name="firstCategoryPrefTitle">Datos personales</string>
    <string name="editTextPrefTitle">Nombre del jugador</string>
    <string name="editTextPrefSummary">Pulse para teclear el nombre del jugador</string>
    <string name="secondCategoryPrefTitle">Ajustes del juego</string>
    <string name="listPrefTitle">Tipo de figura</string>
    <string name="listPrefSummary">Pulse para mostrar la lista</string>
    <string name="checkboxPrefTitle">Música de fondo</string>
    <string name="checkboxPrefSummary">Marque si desea escuchar música</string>
    <string name="listPrefDialogTitle">Elija la figura inicial</string>
    <string name="preferencesText">Ajustes</string>
</resources>
```

Finalmente, el atributo `android:defaultValue` indica cuál es el valor por omisión. Sin embargo no es suficiente con indicar este valor en el fichero `settings.xml`. También debemos incluir la siguiente llamada en el método `onCreate` de la primera actividad que se arranca en nuestro proyecto, en este caso en `Initial.java`:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_void);

    //Cargamos las preferencias por omisión
    PreferenceManager.setDefaultValues(this, R.xml.settings, false);
}
```

La llamada a `setDefaultValues` carga los valores por defecto de las preferencias. El parámetro `false` indica que esta carga solo se debe hacer la primera vez que se llama a la función.

## 27.3 La clase PreferenceFragment

Una vez especificada la interfaz en el fichero `settings.xml`, Android recomienda extender la clase `PreferenceFragment` para gestionar las preferencias. El fichero java del fragmento es muy sencillo pues simplemente invoca al método `addPreferencesFromResource()` desde su callback `onCreate()`:

/src/CCCPreferenceFragment.java

```

package es.uam.eps.android.CCC27;

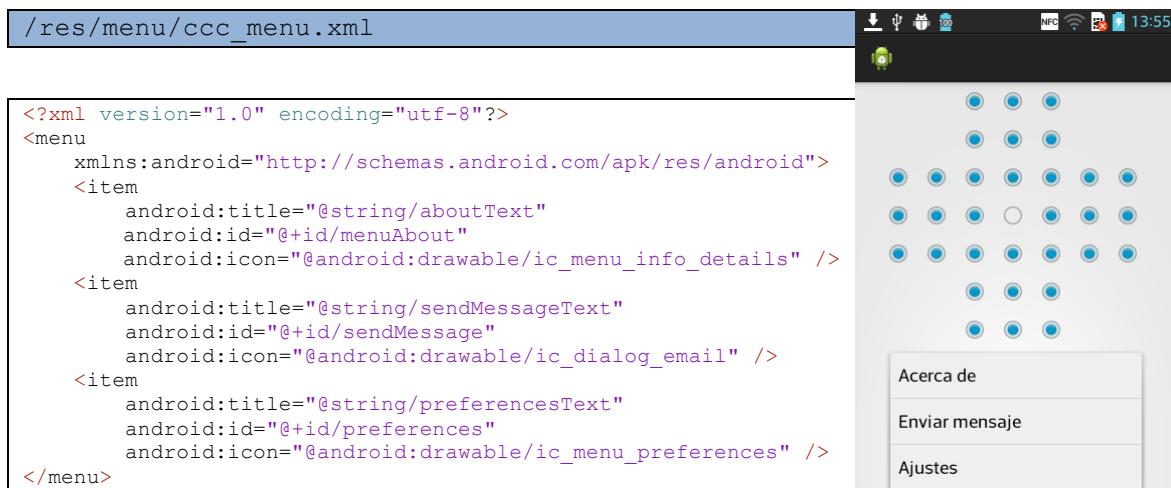
import android.os.Bundle;
import android.preference.PreferenceFragment;

public class CCCPreferenceFragment extends PreferenceFragment{

    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.settings);
    }
}

```

Vamos a utilizar una transacción para construir y mostrar el fragmento dentro del menú de opciones de la actividad `MainActivity`. Primero actualicemos la interfaz del menú de opciones para incluir a las preferencias:



La transacción se ejecuta en el método `onCreate()` de una nueva actividad llamada `CCCPreference` que, además, contiene las constantes con las claves de cada una de las preferencias:

/src/CCCPreference.java

```

package es.uam.eps.android.CCC27;

import android.app.Activity;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.os.Bundle;

public class CCCPreference extends Activity {
    public final static String PLAY_MUSIC_KEY = "music";
    public final static boolean PLAY_MUSIC_DEFAULT = true;
    public final static String PLAYER_KEY = "playerName";
    public final static String PLAYER_DEFAULT = "unspecified";
    public final static String FIGURE_KEY = "figure";
    public final static String FIGURE_DEFAULT =
        "c0011000011001111111101111111100111000011100";

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}

```

```

        setContentView(R.layout.main_void);

        FragmentManager fragmentManager = getFragmentManager();
        FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
        CCCPreferenceFragment fragment = new CCCPreferenceFragment();
        fragmentTransaction.replace(android.R.id.content, fragment);
        fragmentTransaction.commit();
    }
}

```

Esta nueva actividad se arranca en el caso correspondiente del menú de opciones de la clase `MainActivity`:

/src/MainActivity.java

```

...
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menuAbout:
            startActivity(new Intent(this, About.class));
            return true;
        case R.id.sendMessage:
            Intent intent = new Intent(android.content.Intent.ACTION_SEND);
            intent.setType("text/plain");
            intent.putExtra(Intent.EXTRA_SUBJECT, "CHA CHA CHA");
            intent.putExtra(Intent.EXTRA_TEXT,
                "Hola ..., he llegado a ... puntos en cha cha cha ...");
            startActivity(intent);
            return true;
        case R.id.preferences:
            startActivity(new Intent(this, CCCPreference.class));
            return true;
    }
    return super.onOptionsItemSelected(item);
} ...

```

Ahora solo nos queda utilizar la información recogida por el menú de preferencias para alterar el curso del juego. En este proyecto, para no alargarnos demasiado, vamos a utilizar tan solo la preferencia ligada a la música. Haremos que la música se active solo si así lo solicita el jugador. Esto queda reflejado en el método `onResume()` de `MainActivity`:

/src/MainActivity.java

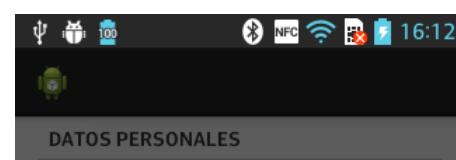
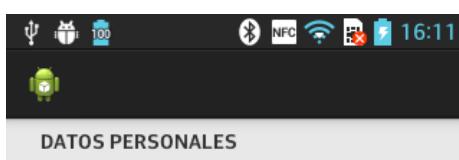
```

...
protected void onResume() {
    super.onResume();
    Boolean play = false;

    SharedPreferences sharedpreferences =
        PreferenceManager.getDefaultSharedPreferences(this);
    if (sharedpreferences.contains(CCCPreference.PLAY_MUSIC_KEY))
        play = sharedpreferences.getBoolean(CCCPreference.PLAY_MUSIC_KEY,
                                            CCCPreference.PLAY_MUSIC_DEFAULT);
    if (play == true)
        Music.play(this, R.raw.funkandblues);
} ...

```

Este es el resultado final:



# Jugando con Android

Aprende a programar tu primera App

Semana 6. Detalles finales

## 28. Animaciones

La plataforma Android proporciona cuatro tipos de animaciones:

- Imágenes GIF animadas. Los GIF animados son ficheros gráficos que contienen varios fotogramas (*frames*).
- Animaciones fotograma a fotograma. Mediante la clase `AnimationDrawable`, el programador suministra los fotogramas y las transiciones entre ellos.
- Animaciones de interpolación (*tweening*). Estas animaciones pueden resolverse con código XML y se aplican a cualquier vista.
- Animaciones con la biblioteca OPEN GL ES.

En esta unidad nos vamos a centrar en las animaciones de interpolación, de las que hay cuatro tipos:

- Animación `alpha` para cambiar la transparencia de una vista.
- Animación `rotate` para rotar una vista un cierto ángulo alrededor de un eje o punto de pivot.
- Animación `scale` para agrandar o disminuir una vista según el eje X e Y.
- Animación `translate` para desplazar una vista a lo largo del eje X e Y.

Las animaciones de interpolación se pueden definir tanto en XML como en Java. Por ejemplo, el proyecto de esta unidad, CCC28, añade una animación `scale` en XML para la figura inicial de nuestro solitario.

Para crear una animación `scale` en XML, añadiremos una carpeta de nombre `anim` a la carpeta `res` de nuestro proyecto. Dentro de esta carpeta, añadiremos un archivo de nombre `initial.xml` como el siguiente:

/res/anim/initial.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<scale xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="4000"
    android:fromXScale="1.0"
    android:fromYScale="0.2"
    android:interpolator="@android:anim/bounce_interpolator"
    android:toXScale="1.0"
    android:toYScale="1.0" />
```

El atributo `android:fromXScale` especifica el valor inicial del escalado según el eje x. Por su lado, `android:toXScale` especifica el valor final. Como queremos conseguir un efecto de caída de telón, tanto el primero como el segundo valen 1.0, de tal forma que la vista no se modifique horizontalmente. Sin embargo, `android:fromYScale` se iguala a 0.2. El atributo `android:duration` especifica la duración de la animación. Añadiremos un interpolador para conseguir el efecto de rebote.

La actividad `Initial`, en su método `onCreate()`, se encarga de conseguir una referencia al `ImageView` de su pantalla y arrancar la animación:

```
/src/Initial.java
```

```
package es.uam.eps.android.ccc28;  
...  
public class Initial extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.initial);  
  
        Animation animation = AnimationUtils.loadAnimation(this, R.anim.initial);  
        ImageView imageView = (ImageView) findViewById(R.id.initial);  
        imageView.startAnimation(animation);  
    }  
    ...  
}
```

Para ello se utiliza la clase de ayuda `AnimationUtils`. Primero hemos de cargar la animación con el método `loadAnimation()`, que tiene dos argumentos: el contexto y el identificador del fichero XML donde se especifica la animación. Finalmente, la animación se arranca desde la vista que queremos animar llamando al método `startAnimation()`.

Las animaciones se pueden combinar en XML mediante un elemento `<set>`. Por ejemplo, en el siguiente fichero se especifica una animación de escala seguida por una de rotación. La segunda empieza cuando acaba la primera pues el atributo `android:startOffset` de la segunda animación se iguala a la duración de la primera (5000 milisegundos). La primera animación dobla el tamaño de la vista según el eje X y lo triplica según el eje Y. La segunda animación rota la vista alrededor de su punto medio (`android:pivotY="50%"`):

```
/res/anim/initial.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>  
<set xmlns:android="http://schemas.android.com/apk/res/android">  
    <scale  
        android:fromXScale="1.0"  
        android:toXScale="2.0"  
        android:fromYScale="1.0"  
        android:toYScale="3.0"  
        android:pivotX="50%"  
        android:pivotY="50%"  
        android:duration="5000"/>  
    <rotate  
        android:fromDegrees="0"  
        android:toDegrees="360"  
        android:pivotX="50%"  
        android:pivotY="50%"  
        android:duration="4000"  
        android:startOffset="5000"/>  
</set>
```

El atributo `android:fillAfter` permite especificar si se quiere que la vista vuelva o no a su estado inicial. Si le asignamos el valor `true`, la vista no volverá a su estado inicial.

## 29. Internacionalización

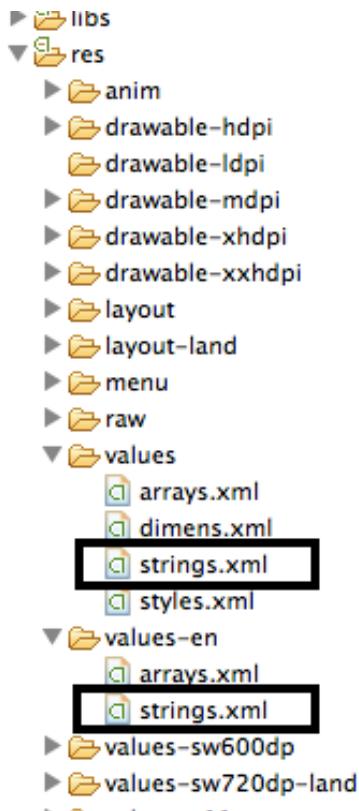
Hemos visto que Android permite adaptar la interfaz de las actividades a la posición del dispositivo (landscape o portrait) mediante ficheros de diseño alternativos. El mismo mecanismo se utiliza para adaptar la aplicación a otras características concretas del dispositivo como la versión del sistema operativo, el idioma, el tamaño de la pantalla, la densidad, etc. En esta unidad vamos a ver cómo crear un recurso alternativo para el idioma inglés en nuestro juego.

Igual que en la unidad 21 añadimos un fichero de diseño `activity_main.xml` alternativo a la carpeta `/res/layout-land/`, ahora añadiremos versiones en inglés de `strings.xml` y `arrays.xml` dentro de una nueva carpeta de recursos alternativa de nombre `/res/values-en`. Android se encargará de seleccionar el fichero de recursos más adecuado dada la configuración del dispositivo:

```
/res/values-en/strings.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">CCC29</string>
    <string name="action_settings">Settings</string>
    <string name="gameOverTitle">Game over</string>
    <string name="gameOverMessage">Do you want to play again?</string>
    <string name="initialImage">Imagen inicial del juego</string>
    <string name="aboutText">About</string>
    <string name="sendMessageText">Send message</string>
    <string name="aboutMessage">Cha cha cha is a solitaire game
        in which we have to get rid of the pieces
        jumping over them, vertically or
        horizontally, towards the adjacent position which
        must be empty. The aim is to leave
        the last piece in the center of the board, the position
        that is initially empty.</string>
    <string name="firstCategoryPrefTitle">Personal data</string>
    <string name="editTextPrefTitle">Player name</string>
    <string name="editTextPrefSummary">Click to type the player name</string>
    <string name="secondCategoryPrefTitle">Game properties</string>
    <string name="listPrefTitle">Type of figure</string>
    <string name="listPrefSummary">Click to pop up a list to choose from</string>
    <string name="checBoxPrefTitle">Background music</string>
    <string name="checBoxPrefSummary">Mark if you want to listen to music</string>
    <string name="listPrefDialogTitle">Choose the initial figure</string>
    <string name="preferencesText">Preferences</string>
</resources>
```

Como puedes ver, el nombre del archivo de recursos es el mismo (`strings.xml`) así como el de las cadenas en su interior (`app_name`, `aboutText`, ...). Lo que cambia es su valor y el nombre de la carpeta en que se encuentra el archivo (`values-en` en lugar de `values`):



De igual manera también proporcionamos una versión en inglés de arrays.xml:

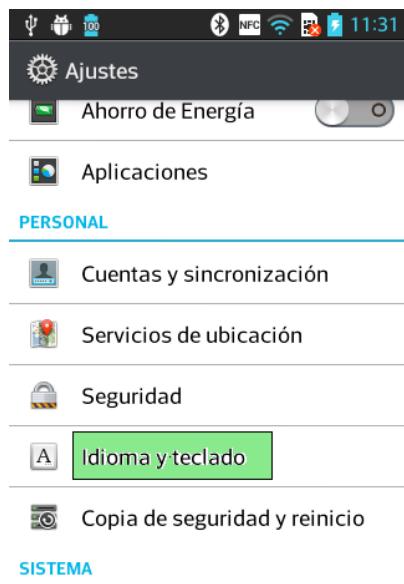
/res/values-en/arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="figures">
        <item>Complete</item>
        <item>Circle</item>
        <item>Ring</item>
    </string-array>
    <string-array name="figureCodes">
        <item>0011000011001111111101111111100111000011100</item>
        <item>00000000110011110011011001111000111000000000</item>
        <item>0000000011000110110010011011000111000000000</item>
    </string-array>
</resources>
```

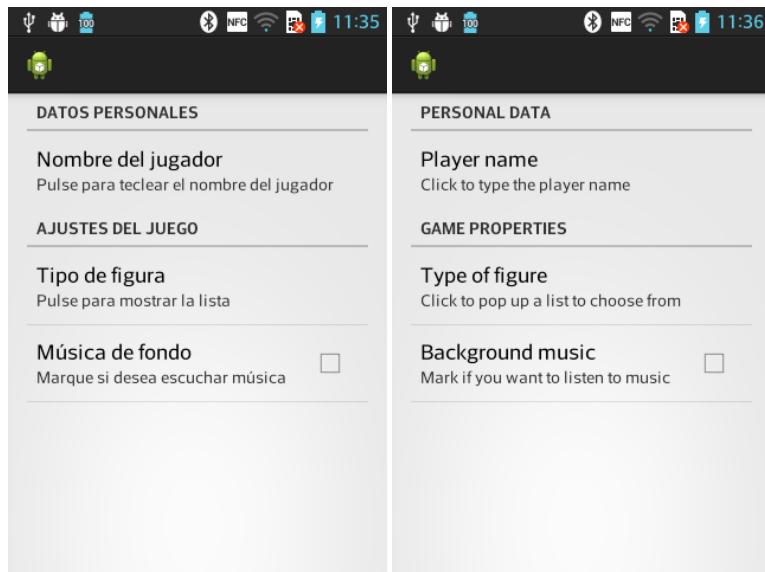
Para dar soporte al idioma francés tendríamos que añadir una carpeta de nombre values-fr, así como un nuevos archivo strings.xml con versiones en francés de los recursos correspondientes.

Podemos localizar aún más los recursos: por ejemplo, podríamos añadir una carpeta de nombre values-en-rUS para proporcionar una alternativa para el inglés de EEUU. Android resuelve la elección de recursos yendo de lo particular a lo general.

Puedes comprobar el efecto cambiando el idioma de tu dispositivo en Ajustes -> Idioma y teclado:



Lo que sigue es el menú de preferencias en castellano y en inglés:



Es muy conveniente dotar a la app de recursos por omisión, es decir, recursos en la carpeta `/res/values`, sin calificadores. Si no existen recursos por omisión, la app terminará bruscamente si Android necesita un recurso y no encuentra el que encaja con la configuración del dispositivo. Nuestro juego, si se fija el italiano como lenguaje del dispositivo, utilizará los recursos en castellano de la carpeta `/res/values`.

Debido al gran número de calificadores de configuración que existen en Android, puede que la configuración del dispositivo encaje con mas de un recurso alternativo. Es por ello que los calificadores tienen asignado un orden o precedencia. Así, por ejemplo, el calificador de lenguaje va antes que el de la orientación del dispositivo. Este orden se ha de respetar al crear recursos alternativos: `/res/values-en-land` en lugar de `/res/values-land-en`.

Finalmente, no olvides que los recursos deben situarse en subdirectorios de `res`. No es posible colocar recursos en la raíz del proyecto, por ejemplo. Los nombres de los subdirectorios están determinados por Android. Si creas uno nuevo de nombre `principal`, por poner un ejemplo, no será tenido en cuenta. Además, no se puede anidar estructura en los directorios estándar, es decir, no podemos crear subdirectorios como, por ejemplo, `fragments` dentro de `/res/layout`. Dada esta limitación, Android recomienda utilizar prefijos y barras bajas, como en `activity_main.xml`, para mantener ordenados los archivos.

## 30. Publicación

Una vez desarrollada tu aplicación, es probable que te plantees compartirla con otros usuarios. Esto se puede hacer publicándola en alguno de los mercados disponibles, como [Google Play](#), enviándola directamente por correo o permitiendo su descarga desde tu propia web. En cualquiera de estos casos, lo primero que debes hacer es empaquetar la aplicación, es decir, generar el fichero APK (*Application Package File*) para su publicación. Antes de empaquetar la aplicación conviene recordar algunas cuestiones relevantes.

### 30.1 Preparación

Escoge un nombre de aplicación y crea un ícono amigable para cada una de las densidades. Ambos se utilizan con frecuencia en el mercado donde publiques la aplicación. Esta [herramienta](#) puede resultarte útil para generar los iconos.

Debes escoger valores para los siguientes dos atributos del fichero de manifiesto: `android:versionCode` y `android:versionName`. El código de versión es un entero que debe incrementarse con cada actualización de la aplicación. Típicamente, la primera entrega de la aplicación lleva código 1. Este valor se va incrementando en una unidad con cada actualización posterior, sea ésta mayor o menor. Esto es así porque este valor es el utilizado internamente por Android y no tiene porqué mostrarse al usuario. Es el valor del otro atributo, el nombre de versión, el que se muestra a los usuarios. Por ejemplo, una app podría tener como código 2 y como nombre 1.1, indicando que se trata de la tercera entrega de la app y que es una actualización menor de la primera entrega.

Asegúrate de que la versión target del sdk es la adecuada para el público al que te diriges.

Ciertos elementos del fichero de manifiesto como, por ejemplo, `<uses-configuration>`, `<uses-permission>`, son utilizados por los mercados para filtrar aplicaciones. Si los utilizas indiscriminadamente, puede que tu aplicación desaparezca para muchos usuarios.

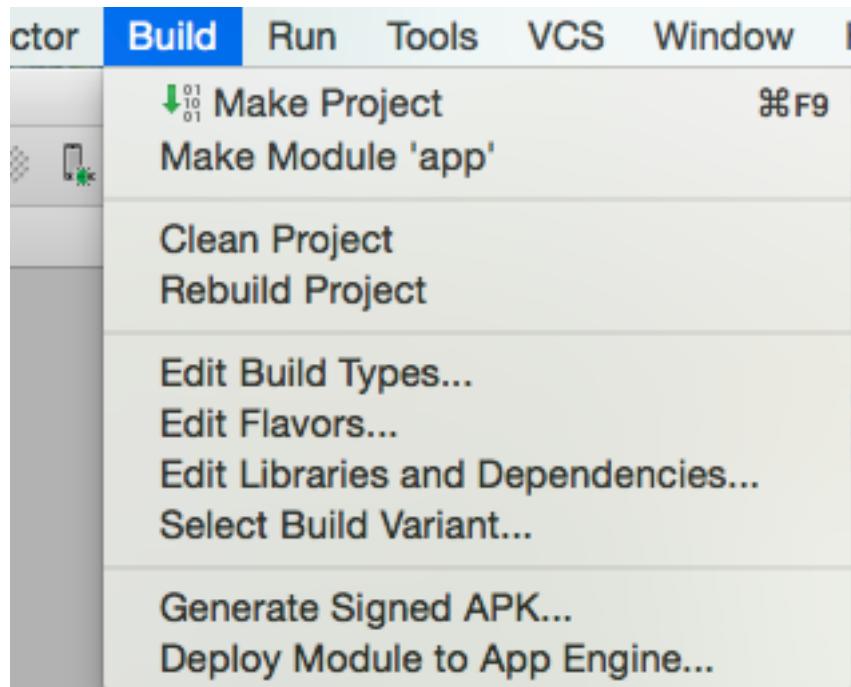
Debes eliminar los mensajes de depuración y de log. En este sentido, debes poner a `false` el elemento `android:debuggable` del fichero de manifiesto.

Finalmente, no olvides poner a prueba a fondo tu app al menos en un teléfono y una tableta.

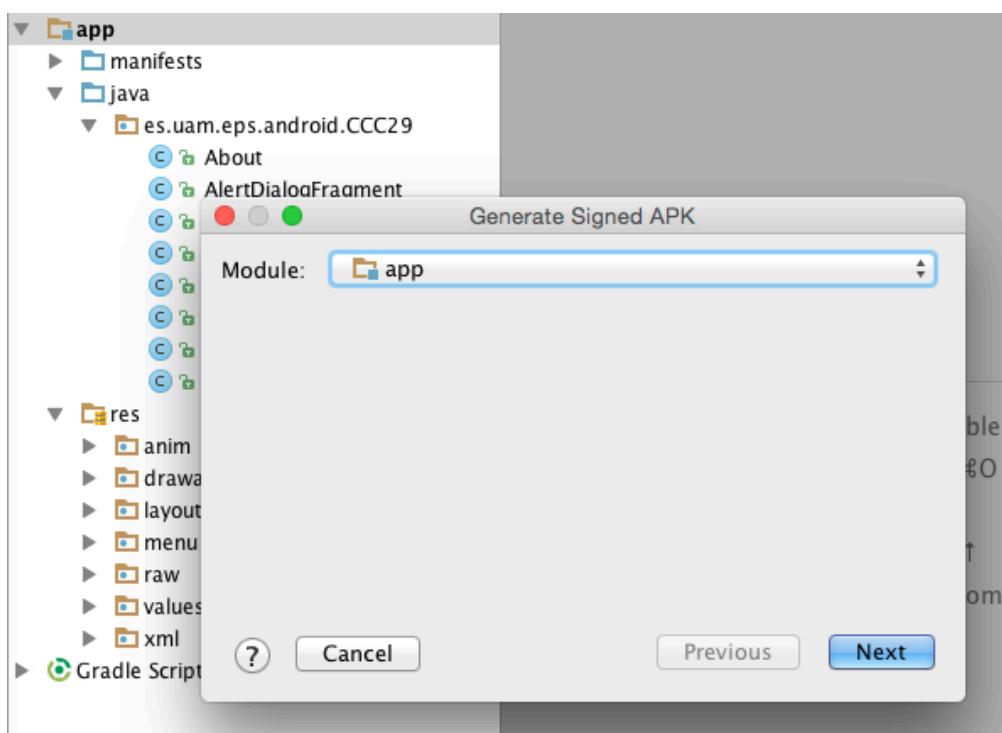
## 30.2 Firma

El gestor de paquetes (*package manager*) de un dispositivo Android no instalará una aplicación que no haya sido firmada. Durante el desarrollo se utiliza una llave de depuración pero para la publicación es necesario utilizar una llave privada que nos identifique. Para firmar la aplicación debes seguir las siguientes instrucciones:

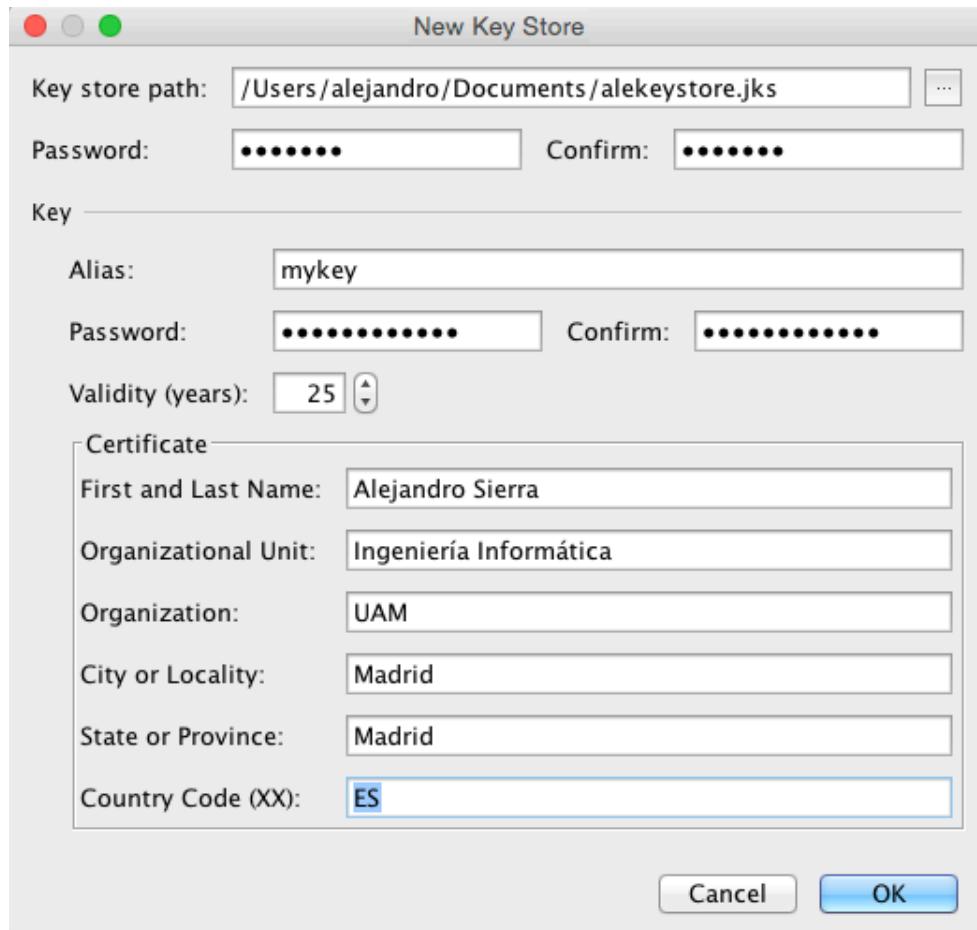
- Pulsa en Build -> Generate Signed APK



- Una vez elegido el módulo, pulsa el botón Next:



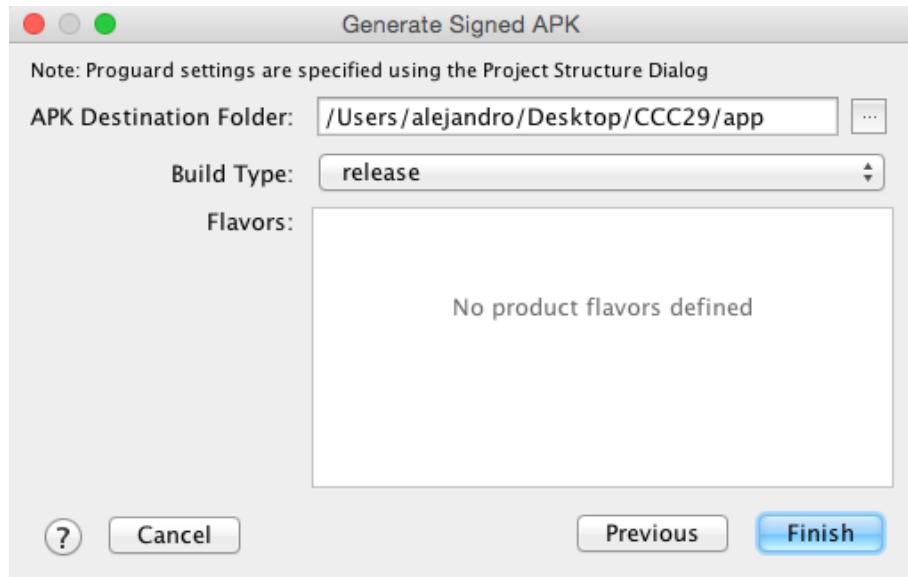
- En la ventana que aparece a continuación, pulsa el botón `Create new ...` para crear un nuevo almacén de claves (*Key Store*) así como una nueva clave (*Key*). Deberás llenar una serie de campos como en la siguiente figura:



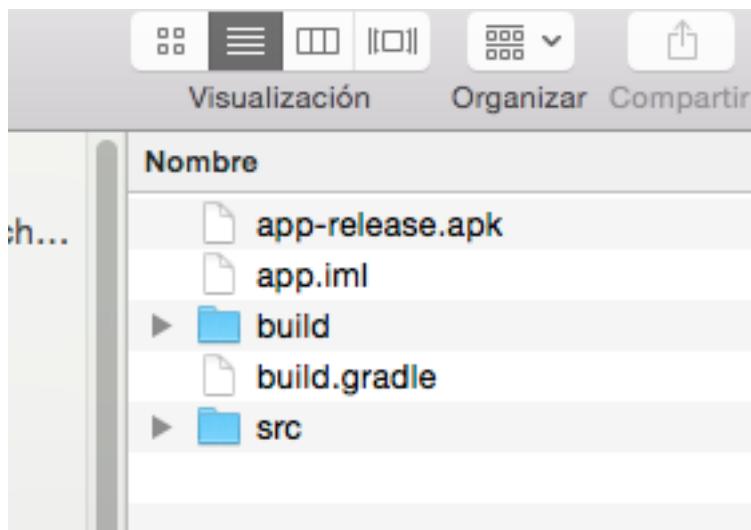
- Al pulsar el botón `OK` verás reaparecer la ventana de partida con sus campos rellenos:



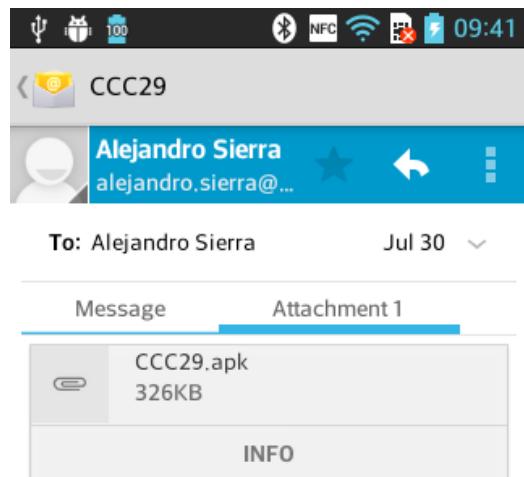
- Pulsamos el botón `Next` y, a continuación, el botón `Finish` de la ventana resultante, donde se muestra la carpeta donde se colocará el fichero APK firmado:



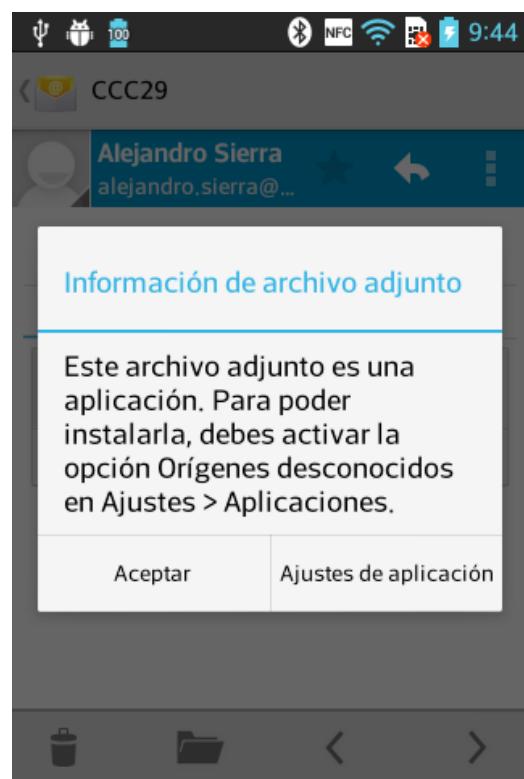
- El fichero resultante recibe el nombre `app-release.apk`, al que renombraremos a `CCC29.apk`:



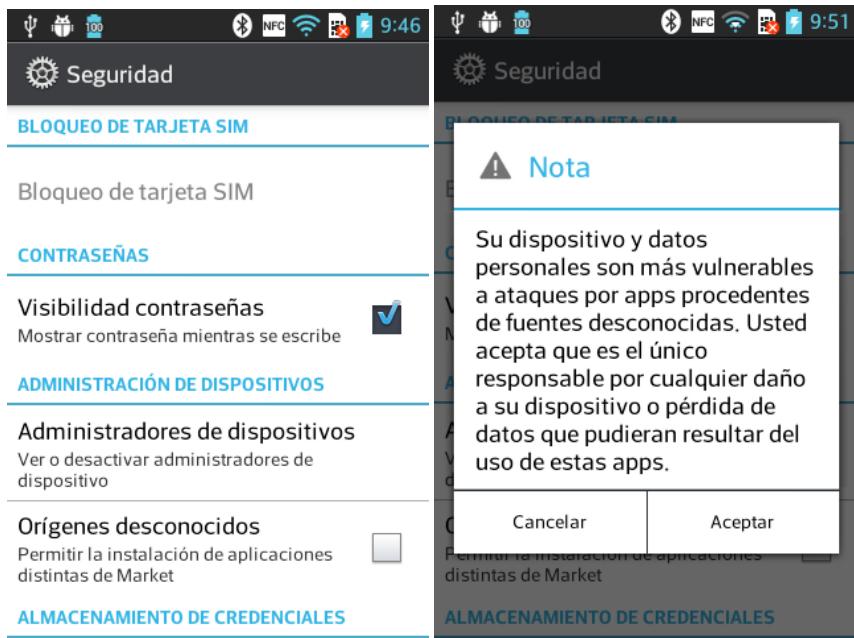
Existen distintos mercados donde puedes publicar tu fichero APK, de entre los cuales el más popular es Google Play. Pero quizás la forma más sencilla de compartir tu app con tus compañeros es enviarla por correo como hacemos a continuación con nuestra aplicación. Algunos gestores de correo permiten ejecutarla directamente:



Al pulsar en INFO obtenemos el siguiente mensaje:



En los ajustes de aplicación podemos solicitar la posibilidad de instalar apps de fuentes desconocidas. Al marcar esta opción, una nota nos advierte del riesgo que corremos:



Después de pulsar Aceptar, ya podemos descargar nuestra aplicación enviada por correo:

