

**Módulo:**

**Programación Multimedia y Dispositivos Móviles**

**UT: Programación En Dispositivos  
Móviles**

Profesor: Juan Carlos Pérez Rodríguez

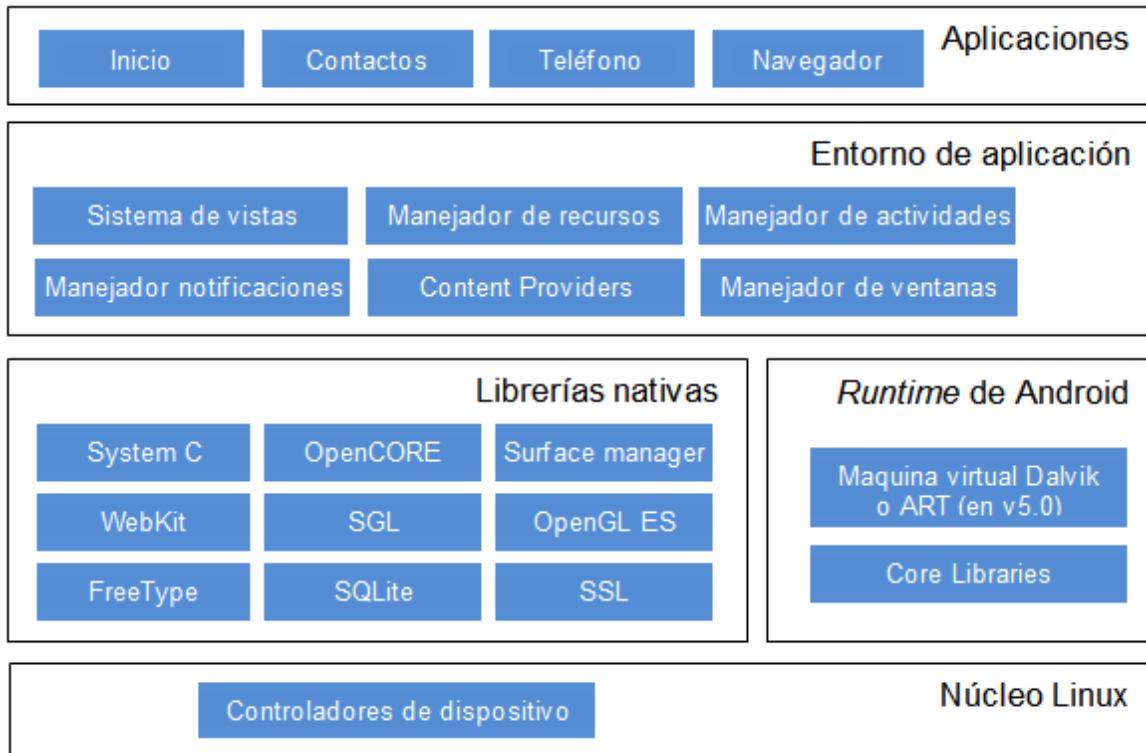
## **Sumario**

1. Arquitectura de Android.....	4
2. Instalación del entorno de desarrollo.....	6
2.1. Instalación de la máquina virtual Java.....	6
2.2 Instalación de Android Studio:.....	6
Estructura del proyecto.....	8
Interfaz de usuario.....	9
Ventanas de herramientas.....	10
Depuración integrada.....	13
2.3 Creación de un dispositivo virtual Android (AVD).....	14
3 Primer proyecto.....	18
4 Ficheros y carpetas de un proyecto Android.....	23
4.1 AndroidManifest.xml.....	24
5. Componentes de una aplicación.....	26
Vista (View).....	26
Layout.....	26
Actividad (Activity).....	26
Ciclo de vida de una actividad.....	27
Servicio (Service).....	28
Intención (Intent).....	28
Fragment.....	28
Receptor de anuncios (Broadcast receiver).....	28
Proveedores de Contenido (Content Provider).....	29
6. Creando Layouts.....	29
6.2 Atributos De Un Layout.....	31
6.3 Viewgroups.....	33
6.3.1 FrameLayout.....	33
6.3.2 LinearLayout.....	37
6.3.4 GridLayout.....	41
6.3.4 RelativeLayout.....	43
Dimensionando los elementos en el layout.....	44
Píxel (px).....	44
Resolución de Pantalla.....	44
Dots per Inch (DPI).....	45
Density-independent Pixel (dp).....	45
7. Creando el código.....	47
7.1 Activities.....	48
7.1.2 Escuchar y manejar eventos onClick.....	49
7.2 Intents.....	52
7.2.1 Introducción a los Intents.....	52
7.2.2 Intents disponibles en Android.....	53
7.2.3 Intent Explícito.....	55
7.2.4 Paso de información con el Intent.....	55
7.2.4.1 Primera opción.....	55
7.2.4.2 Segunda Opción.....	57
7.3 Fragment.....	61
7.3.1 Creando un fragmento dinámico.....	62

## Programación Multimedia y Dispositivos Móviles

7.4 Persistencia.....	65
7.4.1 Acceso a ficheros.....	65
7.4.1.1 Ficheros en memoria interna.....	65
7.4.1.2 Fichero en la memoria externa SD.....	66
7.4.2 Accediendo a SQLite.....	68
7.4.2.1 Insertar en la base de datos.....	69
7.4.2.2 Recuperando información de la base de datos.....	71
7.5 Servicios.....	75
7.5.1 Ciclo de vida de un servicio.....	76
7.5.2 Creando un servicio de ejemplo.....	78
7.5.3 IntentService.....	81
7.6 Comunicación mediante Socket.....	83

# 1. Arquitectura de Android



- **Aplicaciones:** las aplicaciones base incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.
- **Marco de trabajo de aplicaciones:** los desarrolladores tienen acceso completo a los mismos APIs del framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del framework). Este mismo mecanismo permite que los componentes sean reemplazados por el usuario.
- **Bibliotecas:** Android incluye un conjunto de bibliotecas de C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del marco de trabajo de aplicaciones de Android; algunas son: System C library (implementación biblioteca C estándar), bibliotecas de medios, bibliotecas de gráficos, 3D y SQLite, entre otras.

- **Runtime de Android:** Android incluye un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Dalvik ejecutaba hasta la versión 5.0 archivos en el formato Dalvik Executable (.dex), el cual está optimizado para memoria mínima. La Máquina Virtual está basada en registros y corre clases compiladas por el compilador de Java que han sido transformadas al formato.dex por la herramienta incluida "dx". Desde la versión 5.0 utiliza el ART, que compila totalmente al momento de instalación de la aplicación.
  - **Núcleo Linux:** Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. El núcleo también actúa como una capa de abstracción entre el hardware y el resto de la pila de software.
  - **Entorno de aplicación** Proporciona una plataforma de desarrollo libre para aplicaciones con gran riqueza e innovaciones (sensores, localización, servicios, barra de notificaciones, etc.).  
Esta capa ha sido diseñada para simplificar la reutilización de componentes. Las aplicaciones pueden publicar sus capacidades y otras pueden hacer uso de ellas (sujetas a las restricciones de seguridad). Este mismo mecanismo permite a los usuarios reemplazar componentes.  
Los servicios más importantes que incluye son:
    - **Views:** extenso conjunto de vistas, (parte visual de los componentes).
    - **Resource Manager:** proporciona acceso a recursos que no son en código.
    - **Activity Manager:** maneja el ciclo de vida de las aplicaciones y proporciona un sistema de navegación entre ellas.
    - **Notification Manager:** permite a las aplicaciones mostrar alertas personalizadas en la barra de estado.
    - **Content Providers:** mecanismo sencillo para acceder a datos de otras aplicaciones (como los contactos).
- Una de las mayores fortalezas del entorno de aplicación de Android es que se aprovecha el lenguaje de programación Java. El SDK de Android no acaba de ofrecer para su estándar todo lo disponible del entorno de ejecución Java (JRE), pero es compatible con una fracción muy significativa de este.

## 2. Instalación del entorno de desarrollo

### 2.1. Instalación de la máquina virtual Java

Las aplicaciones Android están escritas en Java, por lo que necesitas instalar un *software* para ejecutar código Java en tu equipo. Este *software* se conoce como máquina virtual Java, entorno de ejecución Java, Java Runtime Environment (JRE) o Java Virtual Machine (JVM).

Para instalar la máquina virtual Java: <http://www.java.com/es/download/>, descarga e instala el fichero correspondiente a tu sistema operativo.

### 2.2 Instalación de Android Studio:

#### Requisitos del sistema

##### Windows

- Microsoft® Windows® 7/8/10 (32 o 64 bits).
- 2 GB de memoria RAM como mínimo; se recomiendan 8.
- 2 GB de espacio mínimo disponible en el disco; se recomiendan 4 (500 MB para IDE + 1,5 GB para el Android SDK y la imagen de sistema del emulador).
- Resolución de pantalla mínima de 1280 x 800.
- Para el emulador acelerado: Sistema operativo de 64 bits y procesador Intel® compatible con Intel® VT-x, Intel® EM64T (Intel® 64) y la funcionalidad Execute Disable (XD) Bit.

##### Mac

- Mac® OS X® 10.8.5 o versiones posteriores hasta 10.11.4 (El Capitan).
- 2 GB de memoria RAM como mínimo; se recomiendan 8.
- 2 GB de espacio mínimo disponible en el disco; se recomiendan 4 (500 MB para IDE + 1,5 GB para el Android SDK y la imagen de sistema del emulador).
- Resolución de pantalla mínima de 1280 x 800.

##### Linux

- GNOME o KDE de escritorio.

## Programación Multimedia y Dispositivos Móviles

*Pruebas realizadas en Ubuntu® 12.04, Precise Pangolin (distribución de 64 bits capaz de ejecutar aplicaciones de 32 bits).*

- Distribución de 64 bits capaz de ejecutar aplicaciones de 32 bits.
- GNU C Library (glibc) 2.11 o versiones posteriores.
- 2 GB de memoria RAM como mínimo; se recomiendan 8.
- 2 GB de espacio mínimo disponible en el disco; se recomiendan 4 (500 MB para IDE + 1,5 GB para el Android SDK y la imagen de sistema del emulador).
- Resolución de pantalla mínima de 1280 x 800.
- Para el emulador acelerado: Procesador Intel® compatible con Intel® VT-x, Intel® EM64T (Intel® 64) y la funcionalidad Execute Disable (XD) Bit, o procesador AMD compatible con AMD Virtualization™ (AMD-V™).

## Programación Multimedia y Dispositivos Móviles

Información tomada de developer.android.com ( el lugar para la descarga de android studio )

### Estructura del proyecto

Cada proyecto en Android Studio contiene uno o más módulos con archivos de código fuente y archivos de recursos. Entre los tipos de módulos se incluyen los siguientes:

- Módulos de apps para Android
- Módulos de bibliotecas
- Módulos de Google App Engine

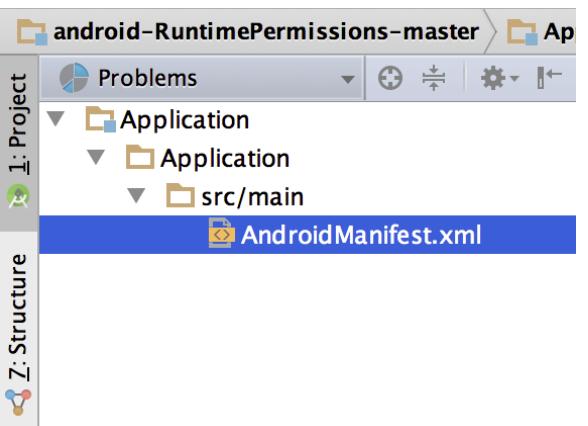
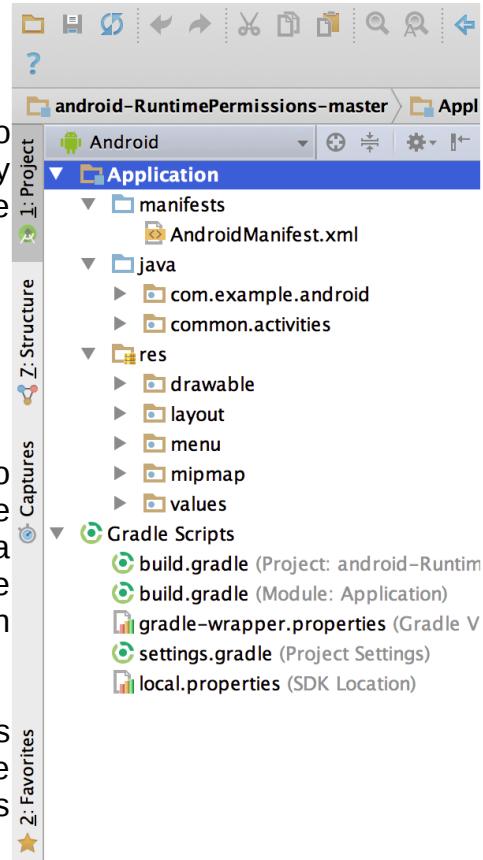
De forma predeterminada, en Android Studio se muestran los archivos de tu proyecto en la vista de proyectos de Android, como se muestra en la Figura 1. Esta vista está organizada en módulos para que puedas acceder rápidamente a los archivos de origen claves de tu proyecto.

Todos los archivos de compilación son visibles en el nivel superior de Secuencias de comando de Gradle y cada módulo de la aplicación contiene las siguientes carpetas:

- manifiestos: contiene el archivo `AndroidManifest.xml`.
- java: contiene los archivos de código fuente de Java, incluido el código de prueba JUnit.
- res: Contiene todos los recursos, como diseños XML, cadenas de IU e imágenes de mapa de bits.

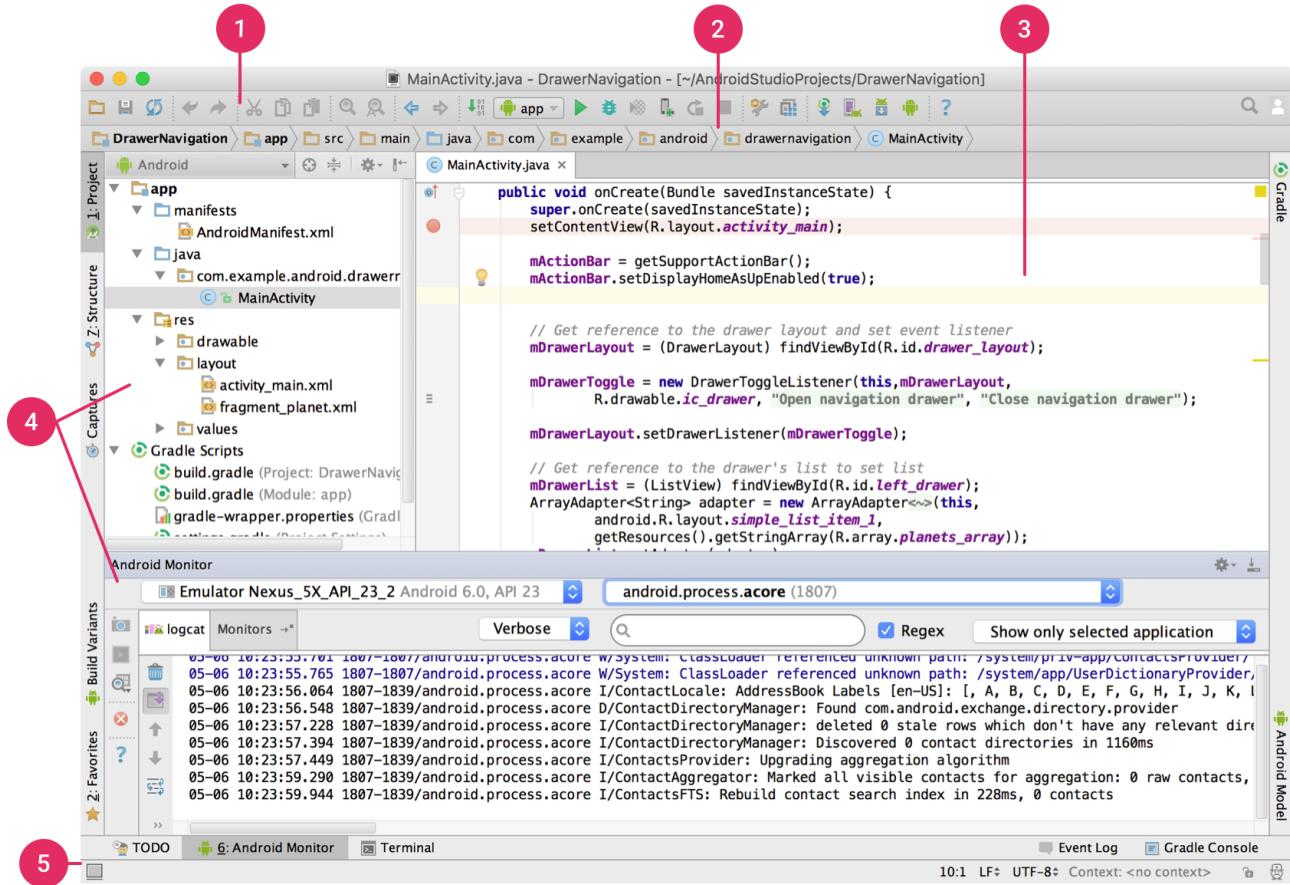
La estructura del proyecto para Android en el disco difiere de esta representación plana. Para ver la estructura de archivos real del proyecto, selecciona Project en la lista desplegable Project (en la figura 1 se muestra como `Android`).

También puedes personalizar la vista de los archivos del proyecto para concentrarte en aspectos específicos del desarrollo de tu app. Por ejemplo, al seleccionar la vista Problems de tu proyecto, aparecerán enlaces a los archivos de origen que contengan errores conocidos de codificación y sintaxis, como una etiqueta de cierre faltante para un elemento XML en un archivo de diseño.



## Interfaz de usuario

La ventana principal de Android Studio consta de varias áreas lógicas:



1. La barra de herramientas te permite realizar una gran variedad de acciones, como la ejecución de tu app y el inicio de herramientas de Android.
2. La barra de navegación te ayuda a explorar tu proyecto y abrir archivos para editar. Proporciona una vista más compacta de la estructura visible en la ventana Project.
3. La ventana del editor es el área en la que puedes crear y modificar código. Según el tipo de archivo actual, el editor puede cambiar. Al visualizar un archivo de diseño, por ejemplo, el editor muestra el Editor de diseño.
4. Las ventanas de herramientas te permiten acceder a tareas específicas, como la administración de proyectos, la búsqueda y los controles de versión, entre otras. Puedes expandirlas y contraerlas.
5. En la barra de estado se muestra el estado de tu proyecto y el IDE, además de advertencias o mensajes.

Puedes organizar la ventana principal para tener más espacio en pantalla ocultando o desplazando barras y ventanas de herramientas. También puedes usar combinaciones de teclas para acceder a la mayoría de las funciones del IDE.

En cualquier momento, puedes realizar búsquedas en tu código fuente, bases de datos, acciones, elementos de la interfaz de usuario, etc., presionando dos veces la tecla Shift o haciendo clic en la lupa que se encuentra en la esquina superior derecha de la ventana de Android Studio. Esto puede ser muy útil, por ejemplo, si intentas localizar una acción específica del IDE que olvidaste cómo activar.

### Ventanas de herramientas

En lugar de usar perspectivas preestablecidas, Android Studio sigue tu contexto y te ofrece automáticamente ventanas de herramientas relevantes mientras trabajas. De forma predeterminada, las ventanas de herramientas usadas con mayor frecuencia se fijan en la barra de ventanas de herramientas en los bordes de la ventana de la aplicación.

- Para expandir o contraer una ventana de herramientas, haz clic en el nombre de la herramienta en la barra de la ventana de herramientas. También puedes arrastrar, anclar, desanclar, adjuntar y ocultar ventanas de herramientas.
- Para volver al diseño predeterminado actual de ventanas de herramientas, haz clic en Window > Restore Default Layout o personaliza tu diseño predeterminado haciendo clic en Window > Store Current Layout as Default.
- 
- Para mostrar u ocultar la barra de ventanas de herramientas completa, haz clic en el ícono de ventana en la esquina inferior izquierda de la ventana de Android Studio.
- 
- Para localizar una ventana de herramientas específica, posiciona el puntero sobre el ícono de ventana y selecciona la ventana de herramientas en el menú.

También puedes usar combinaciones de teclas para abrir ventanas de herramientas. En la Tabla 1 se muestran las combinaciones de teclas para las ventanas más comunes.

Combinaciones de teclas para algunas ventanas de herramientas útiles.		
Ventana de herramientas	Windows y Linux	Mac
Proyecto	Alt+1	Comando+1
Control de versión	Alt+9	Comando+9
Ejecutar	Shift+F10	Control+R
Depurar	Shift+F9	Control+D
Android Monitor	Alt+6	Comando+6
Volver al editor	Esc	Esc

## Programación Multimedia y Dispositivos Móviles

### Completar código

Tipo	Descripción	Windows y Linux	Mac
Compleción básica	Muestra sugerencias básicas para variables, tipos, métodos y expresiones, entre otras. Si llamas a la compleción básica dos veces seguidas, verás más resultados. Entre otros, miembros privados y miembros estáticos sin importar.	Control+Espacio	Control+Espacio
Compleción inteligente	Muestra opciones relevantes en función del contexto. La compleción inteligente reconoce el tipo y los flujos de datos previstos. Si llamas a la compleción inteligente dos veces seguidas, verás más resultados. Por ejemplo, cadenas.	Control+Shift+Espacio	Control+Shift+Espacio
Compleción de enunciados	Completa la instrucción actual agregando elementos faltantes, como paréntesis, corchetes, llaves y formato, entre otros.	Control+Shift+Enter	Shift+Comando+Enter

También puedes realizar correcciones rápidas y mostrar acciones de intención presionando Alt+Enter.

### Sistema de compilación de Gradle

Android Studio usa Gradle como base del sistema de compilación, y proporciona más características específicas de Android a través del Complemento de Android para Gradle. Este sistema de compilación se ejecuta en una herramienta integrada desde el menú de Android Studio, y lo hace independientemente de la línea de comandos. Puedes usar las funciones del sistema de compilación para lo siguiente:

- personalizar, configurar y extender el proceso de compilación;
- crear varios APK para tu app con diferentes funciones usando el mismo proyecto y los mismos módulos;
- volver a usar códigos y recursos entre conjuntos de orígenes.

Recurriendo a la flexibilidad de Gradle, puedes lograr todo esto sin modificar los archivos de origen de tu app. Los archivos de compilación de Android Studio se denominan build.gradle. Son archivos de texto sin formato que usan sintaxis Groovy para configurar la compilación con elementos proporcionados por el complemento de Android para Gradle. Cada proyecto tiene un archivo de compilación de nivel superior para todo el proyecto y archivos de compilación de nivel de módulo independientes para cada módulo. Cuando importas un proyecto existente, Android Studio genera automáticamente los archivos de compilación necesarios.

### Depuración integrada

Usa la depuración integrada para mejorar las revisiones de código en la vista del depurador con verificación integrada de referencias, expresiones y valores de variables. La información de depuración integrada incluye:

- valores de variables integradas;
- objetos que hacen referencia a un objeto seleccionado;
- valores de retorno de métodos;
- expresiones Lambda y de operador;

## Programación Multimedia y Dispositivos Móviles

- valores de información sobre herramientas.

Para habilitar la depuración integrada, en la ventana Debug haz clic en Settings y selecciona la casilla de verificación Show Values Inline.

### Acceso a archivos de datos

Las herramientas del Android SDK, como Systrace, logcat y Traceview, , generan datos de rendimiento y depuración para un análisis detallado de la app.

Para ver los archivos de datos generados disponibles, abre la ventana de herramientas Captures. En la lista de los archivos generados, haz doble clic en uno para ver los datos. Haz clic con el botón secundario en cualquiera de los archivos .hprof para convertirlos al formato de archivo .hprof estándar.

### Mensajes de registro

Cuando compilas y ejecutas tu app con Android Studio, puedes ver mensajes adb de salida y mensajes de registro del dispositivo (logcat) haciendo clic en Android Monitor en la parte inferior de la ventana.

Si quieras depurar tu app con el Monitor de dispositivos Android, puedes iniciar el Monitor de dispositivos haciendo clic en Tools > Android > Android Device Monitor. En el Monitor de dispositivos encontrarás el conjunto completo de herramientas DDMS para perfilar tu app, controlar comportamientos del dispositivo y más. En este también se incluye la herramienta del Visor de jerarquía para ayudarte a optimizar tus diseños.

## 2.3 Creación de un dispositivo virtual Android (AVD)

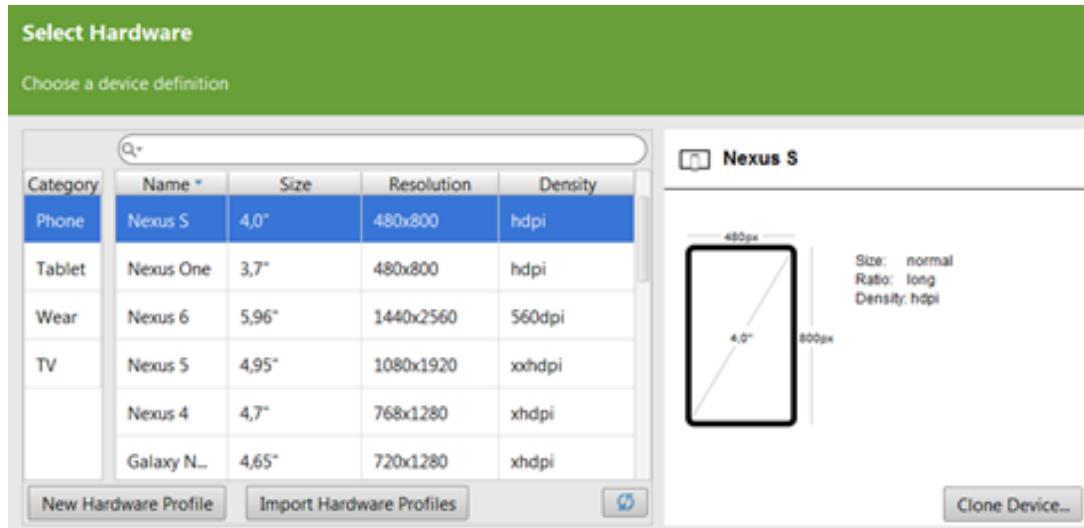
Un dispositivo virtual Android (AVD) te va a permitir emular en tu ordenador diferentes tipos de dispositivos basados en Android. De esta forma podrás probar tus aplicaciones en una gran variedad de teléfonos, tabletas, relojes o TV con cualquier versión de Android, tamaño de pantalla o tipo de entrada.

1. Pulsa el botón AVD Manager:



Te aparecerá la lista con los AVD creados. La primera vez estará vacía.

- 2.** Pulsa a continuación el botón *Create Virtual Device...* para crear un nuevo AVD. Aparecerá la siguiente ventana:



- 3.** En la primera columna podremos seleccionar el tipo de dispositivo a emular (móvil, tableta, dispositivo *wearable* o Google TV). A la derecha, se muestran distintos dispositivos que emulan dispositivos reales de la familia Nexus y también otros genéricos. Junto al nombre de cada dispositivo, se indica el tamaño de la pantalla en pulgadas, la resolución y el tipo de densidad gráfica. **NOTA:** Los tipos de pantalla se clasifican en Android según su densidad gráfica: ldpi, mdpi, hdpi, xhdpi, ... Véase sección 2.6 Recursos alternativos

- 4.** Este parámetro es de gran importancia para determinar el tamaño que ocuparán los gráficos en la pantalla del dispositivo.

Si quisieras añadir a esta lista crear un nuevo tipo de dispositivo, puedes seleccionar *New Hardware Profile*. Podrás indicar las principales características del dispositivo y ponerle un nombre. Usando *Clone Device* podrás crear un nuevo tipo de AVD a partir del actual. Pulsando con el botón derecho sobre un tipo de dispositivo podrás eliminarlos o exportarlos a un fichero.

- 5.** Pulsa *Next* para pasar a la siguiente ventana, donde podrás seleccionar la imagen del sistema que tendrá el dispositivo y el tipo de procesador:

## Programación Multimedia y Dispositivos Móviles

Release Name	API Level	ABI	Target
Lollipop	21	armeabi-v7a	Android 5.0.1
Lollipop	21	x86	Android 5.0.1
Lollipop	21	x86_64	Android 5.0.1
KitKat	19	armeabi-v7a	Android 4.4
KitKat	19	armeabi-v7a	Google APIs (Go)
Jelly Bean	18	armeabi-v7a	Android 4.3
Jelly Bean	18	armeabi-v7a	Google APIs (Go)
Jelly Bean	17	armeabi-v7a	Android 4.2.2

Observa como las distintas versiones de Android se pueden seleccionar, solo con el código abierto de Android o además añadiendo las API de Google. Si escogemos la segunda, podremos usar ciertos servicios de Google como Maps o Play.

6. Pulsa Next para pasar a la última ventana. Se nos mostrará un resumen con las opciones seleccionadas, además podremos seleccionar el tamaño inicial del AVD, si queremos usar el coprocesador gráfico (GPU) de nuestro ordenador o si queremos guardar una imagen congelada del emulador para que arranque más rápido las próximas veces.

7. Pulsa en el botón Show Avanced Settings para que se muestren algunas configuraciones adicionales:

Custom skin definition	C:\Program Files\Android\Android Studio\plugins\android\lib\device-art-resources\n	
<a href="#">How do I create a custom hardware skin?</a>		
Memory and Storage	RAM:	351428 KB
	VM heap:	32 MB
	Internal Storage:	200 MB
	SD card:	100 MB
<a href="#">Or use an existing data file...</a>		
Camera	Front:	None
	Back:	None
Keyboard	<input checked="" type="checkbox"/> Enable keyboard input	
Network	Speed:	Full
	Latency:	None

Podremos cambiar el aspecto estético de la carcasa del dispositivo. Podremos ajustar la memoria utilizada: RAM total del dispositivo, memoria dinámica usada por Java y memoria para almacenamiento, tanto interna como externa. Podremos hacer que el

## Programación Multimedia y Dispositivos Móviles

emulador utilice la cámara o teclado de nuestro ordenador. Finalmente, podremos limitar la velocidad y latencia en el acceso a la red.

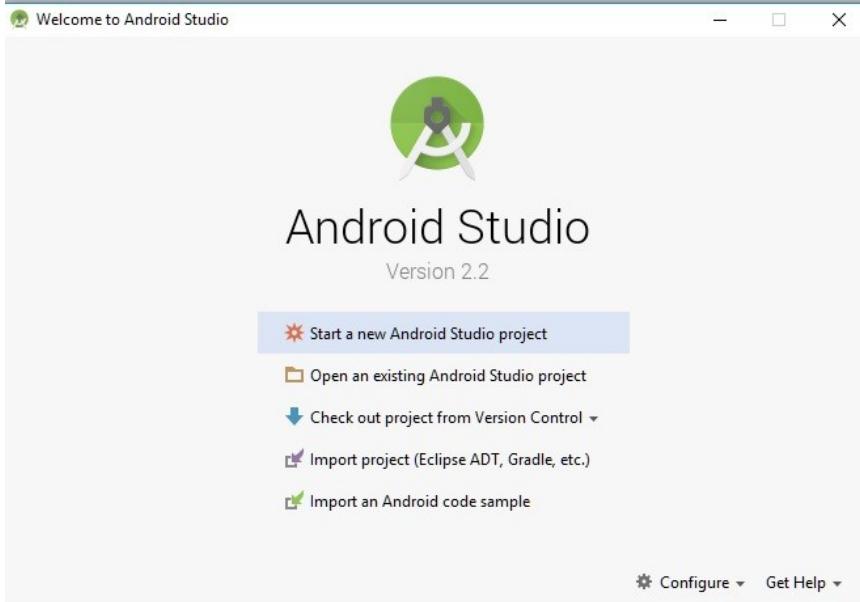
**8.** Una vez introducida la configuración deseada, pulsa el botón *Finish*. Aparecerá el dispositivo creado en la lista:



Type	Name	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
Tablet	Nexus 7	800 × 1280: t...	17	Android 4.2.2	arm	58 MB	  
Tablet	Nexus 5 API 21	1080 × 1920:...	21	Android 5.0.1	x86_...	1 GB	  
Tablet	Nexus 6 API 21	1440 × 2560:...	21	Android 5.0.1	x86_...	1 GB	  

**9.** Para arrancarlo, pulsa el botón con forma de triángulo verde que encontrarás en la columna de la derecha. Es posible que te pregunte por la entrada de vídeo para emular la cámara del AVD.

### 3 Primer proyecto

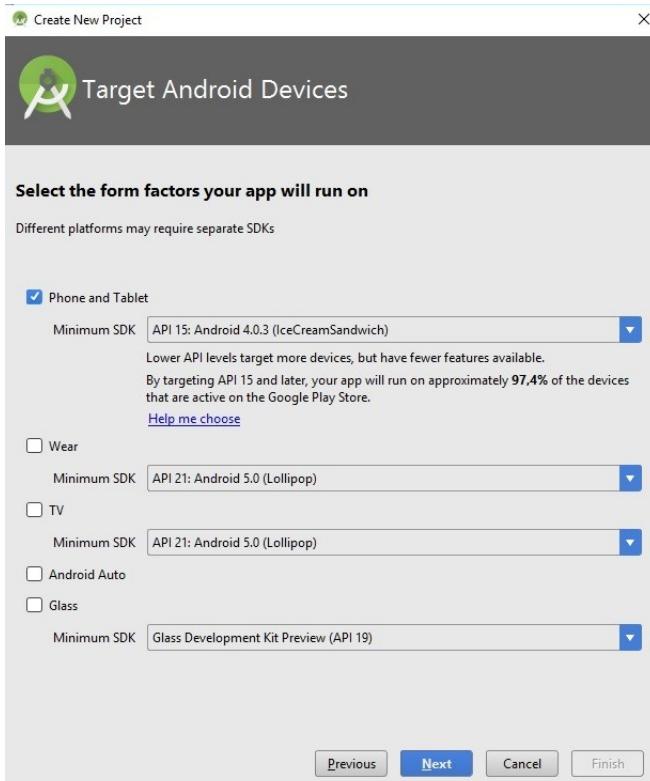


Elegir la opción "Start a New Android Studio project"

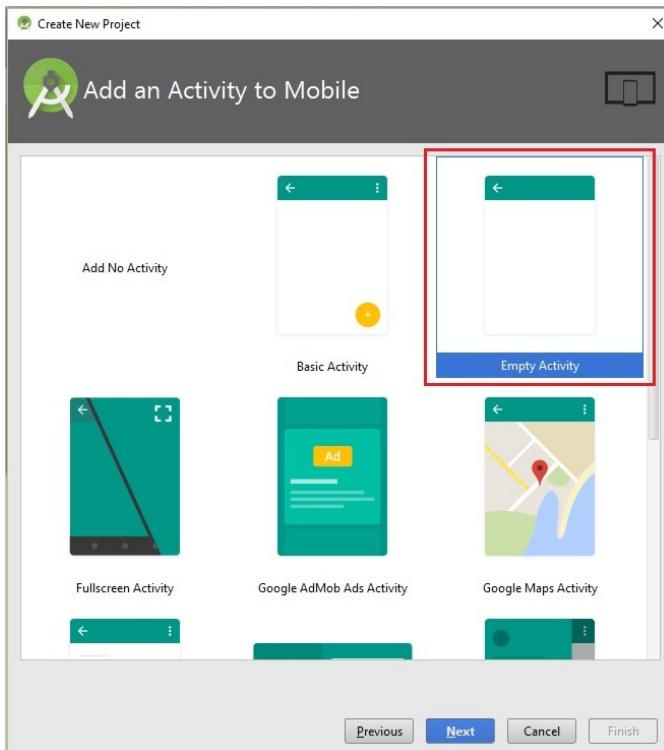
En el primer diálogo debemos especificar el Nombre de la aplicación, y la ubicación en el disco del proyecto

Segundo diálogo especificar la versión de Android mínima donde se ejecutará la aplicación que desarrollamos

## Programación Multimedia y Dispositivos Móviles

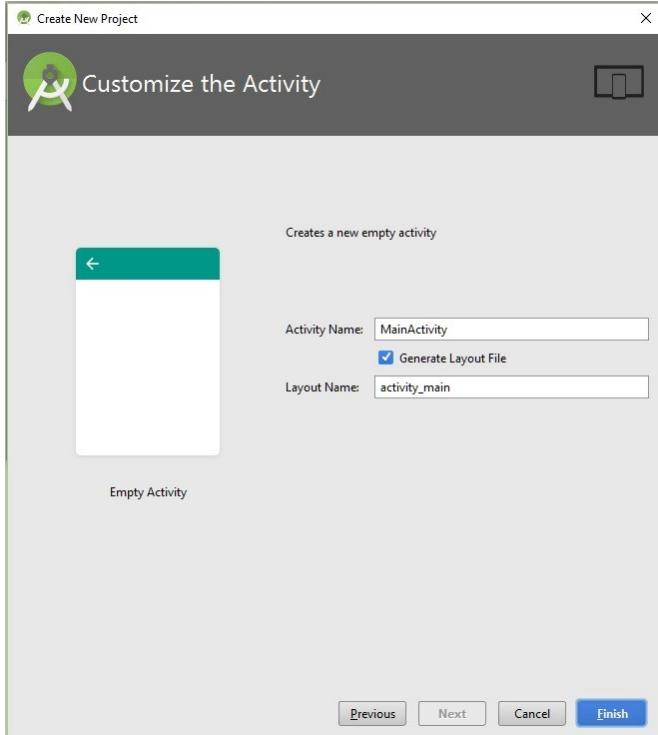


### Tercer diálogo: Elegir "Empty Activity"

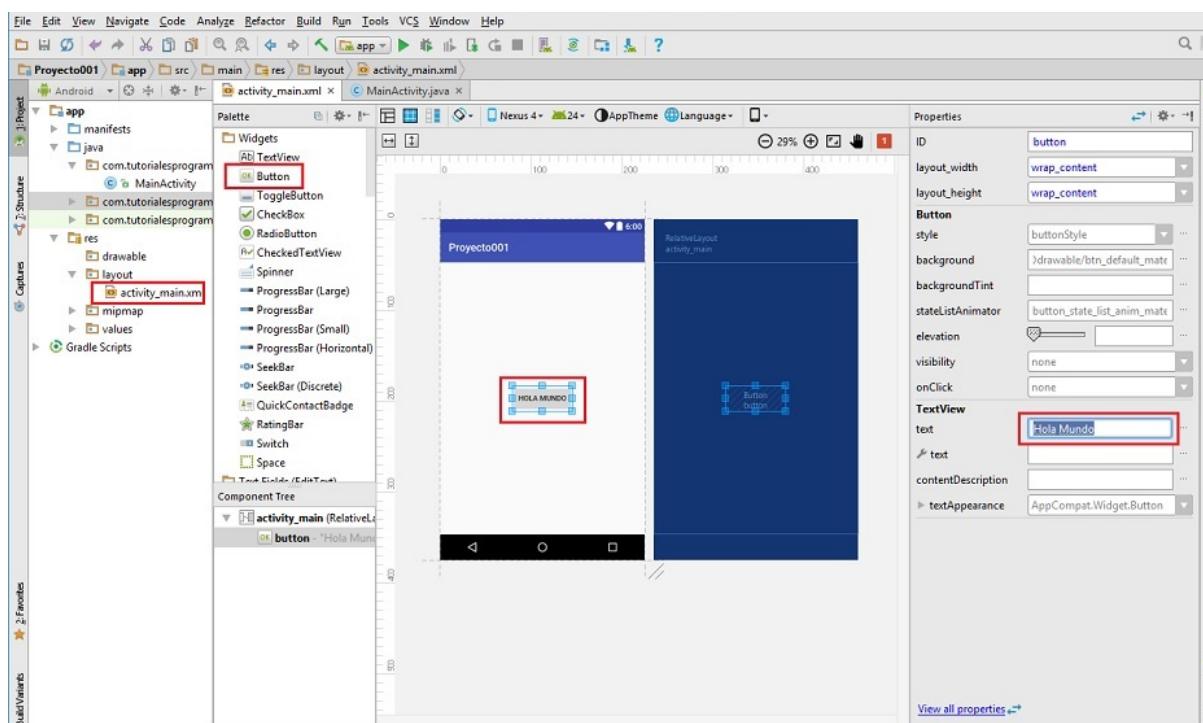


### Cuarto diálogo: indicar el nombre de la ventana principal de la aplicación (Activity Name)

## Programación Multimedia y Dispositivos Móviles

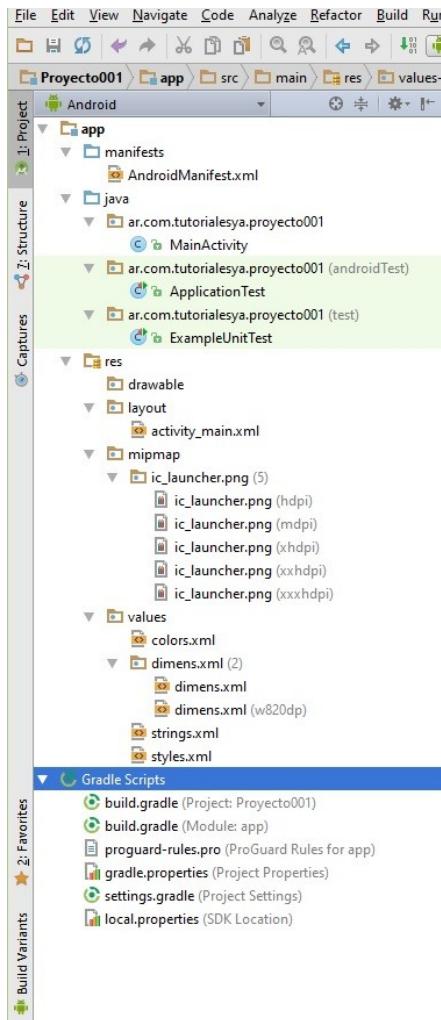


Android Studio nos genera todos los directorios y archivos básicos para iniciar nuestro proyecto



## Programación Multimedia y Dispositivos Móviles

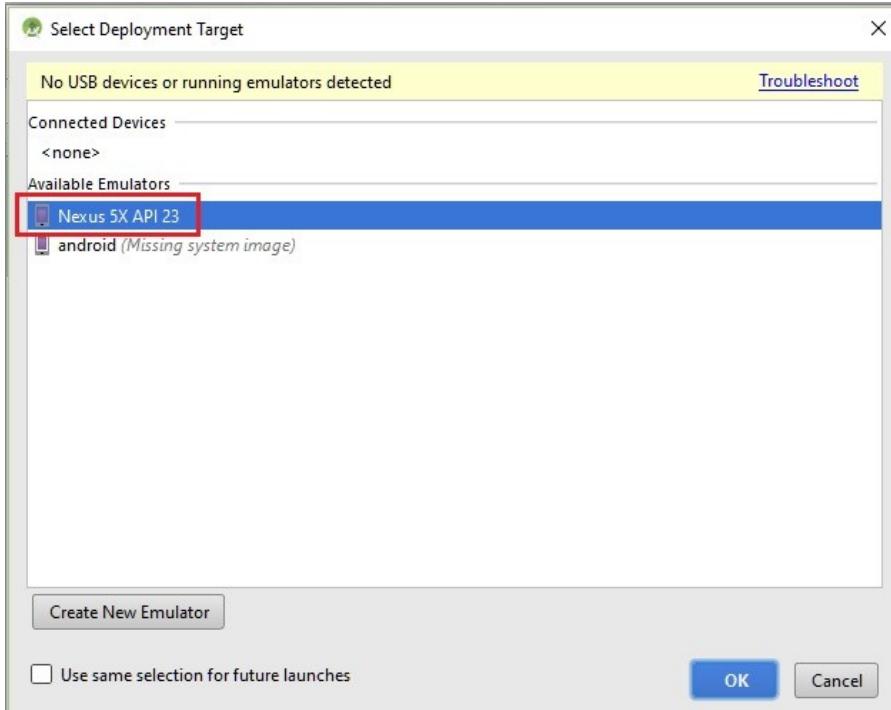
Veamos la estructura de ficheros generada:



- AndroidManifest.xml guardamos información de la aplicación importante para su ejecución y publicación
- En la carpeta java ponemos los fuentes de código
- En res→layout va la parte de modelado de pantallas gráficas
- En values→strings las cadenas de texto (esta parte es importante para hacer traducciones de nuestra aplicación en diferentes idiomas con facilidad)

Para ejecutar la aplicación el triángulo verde o "Run -> Run app" y presionamos el botón "OK" (si no hay emulador se puede crear en ese momento):

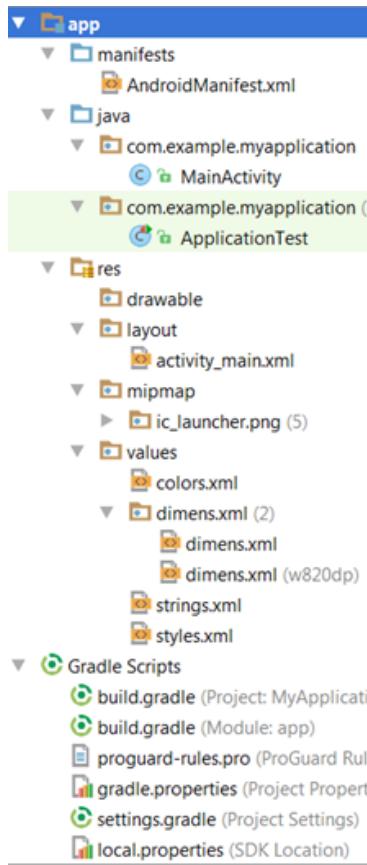
## Programación Multimedia y Dispositivos Móviles



Luego de un rato aparecerá el emulador de Android en pantalla. No cerrar este emulador es lento en la carga y si se deja cargado las aplicaciones van arrancando más rápido:

También se puede ejecutar en un terminal real pero hay que disponer de los driver. En el móvil se deberá activar la opción para desarrolladores: Ajustes->Información del teléfono y pulsar siete veces Luego activar Depuración de USB

## 4 Ficheros y carpetas de un proyecto Android



Cada módulo en Android está formado por un descriptor de la aplicación (*manifests*), el código fuente en Java (*java*), una serie de ficheros con recursos (*res*) y ficheros para construir el módulo (*Gradle Scripts*).

- *AndroidManifest.xml*: Describe la aplicación Android. Se define su nombre, paquete, etc. Se indican las *actividades*, las *intenciones*, los *servicios* y los *proveedores de contenido* de la aplicación. También se declaran los permisos que requerirá la aplicación. Se indica la versión mínima de Android para poder ejecutarla, el paquete Java, la versión de la aplicación, etc.
- *java*: Carpeta que contiene el código fuente de la aplicación. Como puedes observar los ficheros Java se almacenan en carpetas según el nombre de su paquete.
- *MainActivity*: Clase Java con el código de la actividad inicial.
- *ApplicationTest*: Clase Java pensada para insertar código de testeo de la aplicación utilizando el API Junit.
- *res*: Carpeta que contiene los recursos usados por la aplicación.
- *drawable*: En esta carpeta se almacenan los ficheros de imágenes (JPG o PNG) y descriptores de imágenes en XML.
- *mipmap*: Es una carpeta con la misma finalidad que *res/drawable*. La única diferencia es que si ponemos los gráficos en *mipmap*, estos no son rescalados para adaptarlos a la densidad gráfica del dispositivo donde se ejecuta la aplicación, sino que se buscará en las subcarpetas el gráfico con la densidad más parecida y se utilizará directamente. Es recomendable guardar los iconos de aplicación en esta carpeta
- *layout*: Contiene ficheros XML con vistas de la aplicación. Las vistas nos permitirán configurar las diferentes pantallas que compondrán la interfaz de usuario de la aplicación. Se utiliza un formato similar al HTML usado para diseñar páginas web.
- *values*: Ficheros XML para indicar valores usados en la aplicación, En el fichero *strings.xml*, tendrás definir todas las cadenas de caracteres resultará muy sencillo traducir una aplicación a otro idioma.

Un fichero especialmente importante es androidmanifest.xml:

### 4.1 AndroidManifest.xml

```

1<?xml version="1.0" encoding="utf-8"?>
2<manifest xmlns:android="http://schemas.android.com/apk/res/android"
3   package="com.androidapp.basicElements"
4   android:versionCode="1"
5   android:versionName="1.0">
6     <application android:icon="@drawable/icon" android:label="@string/app_name">
7       <activity android:name=".BasicElements"
8         android:label="@string/app_name">
9           <intent-filter>
10             <action android:name="android.intent.action.MAIN" />
11             <category android:name="android.intent.category.LAUNCHER" />
12           </intent-filter>
13         </activity>
14       </application>
15     <uses-sdk android:minSdkVersion="2" />
16   </manifest>

```

play

- installLocation: donde debe instalarse la aplicación ( ejemplo en la tarjeta sd )
- la etiqueta application:

    android:icon="@drawable/icon"  
    android:label="@string/app\_name"

    la @ significa que queremos referenciar un recurso que está en algún lado

    @drawable/icon dice que dentro de res/drawable está el archivo icon (observar que no lleva extensión )

    label="@string/app\_name" está en carpeta values ( res/values/strings.xml Aquí se almacenan todos los strings de la aplicación. Esto es todo texto que se muestra. Es cómodo así para cambiar de idioma )

    se puede poner directamente aquí la cadena de título de la aplicación en lugar de referenciarlo a string.xml ( no se recomienda por las traducciones

    )

- android:debuggable="true" ( evidente su utilidad)

#### **Etiqueta activity ( aparece dentro de application en manifest )**

atributos:

    android:name="nombre de la actividad"

    android:label="titulo" ( si no se especifica se usa label de application )

    android:screenOrientation="portrait" ( orientación de la pantalla al mostrar la actividad

En este caso únicamente podrá mostrarse en portrait. De no poder nada entonces se muestra en ambas formas. Cuidado con que al girar te destruye la activity y te la vuelve a crear)

- package: nombre del paquete raíz para las clases
- versionCode: número que se incrementa cada vez que cambiamos la versión para google play
- versionName: es el nombre que se les mostrará a los usuarios en google

### etiqueta intent-filter ( está dentro de activity )

Para relacionar las activities con los intent

```

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

cada vez que queramos una activity como la principal es la primera linea con MAIN

Si no se establecen los intent-filter únicamente accederas con la activity desde dentro de la aplicación Si se quiere nos dice de esta forma cuál es la activity que se lanzará al iniciar la aplicación

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.treslines.phonegapfirstapp"
4      android:versionCode="1"
5      android:versionName="1.0" 
6
7      <supports-screens
8          android:anyDensity="true"
9          android:largeScreens="true"
10         android:normalScreens="true"
11         android:resizeable="true"
12         android:smallScreens="true"
13         android:xlargeScreens="true" />
14
15     <uses-permission android:name="android.permission.CAMERA" />
16     <uses-permission android:name="android.permission.VIBRATE" />
17     <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
18     <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
19     <uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
20     <uses-permission android:name="android.permission.INTERNET" />
21     <uses-permission android:name="android.permission.RECEIVE_SMS" />
22     <uses-permission android:name="android.permission.RECORD_AUDIO" />
23     <uses-permission android:name="android.permission.RECORD_VIDEO" />
24     <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
25     <uses-permission android:name="android.permission.READ_CONTACTS" />
26     <uses-permission android:name="android.permission.WRITE_CONTACTS" />
27     <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
28     <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
29     <uses-permission android:name="android.permission.GET_ACCOUNTS" />
30     <uses-permission android:name="android.permission.BROADCAST_STICKY" />
31
32     <uses-sdk
33         android:minSdkVersion="7"
34         android:targetSdkVersion="17" />
35
36     <application
37         android:allowBackup="true"
38         android:icon="@drawable/ic_launcher"
39         android:label="@string/app_name"
40         android:theme="@style/AppTheme" >
41         <activity
42             android:name="com.treslines.phonegapfirstapp.MainActivity"
43             android:label="@string/app_name"
44
45             android:configChanges="orientation/keyboardHidden"
46         >
47     </application>

```

**Permisos( ya no estamos en application. Hijo directo de manifest):**

<uses-permission>  
a qué puede acceder ( internet, tarjeta sd,... )

Ej.

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

### Etiqueta: <uses-feature> ( hijo de manifest )

google play filtrará según las restricciones de hardware las aplicaciones que son apropiadas para el hardware dado según esa etiqueta

Por ej. para un juego precisarás una multitouch.distinct ( permite un control de mandos de juego más completo )

```

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

(uses-sdk hijo de manifest Versión número de android min para la mínima y target para la objetivo)

```
<uses-sdk android:minSdkVersion="3" android:targetSdkVersion="9" />
```

## 5. Componentes de una aplicación

### Vista (View)

Las vistas son los elementos que componen la interfaz de usuario de una aplicación: por ejemplo, un botón o una entrada de texto. Todas las vistas van a ser objetos descendientes de la clase View, y por tanto, pueden ser definidas utilizando código Java.

### Layout

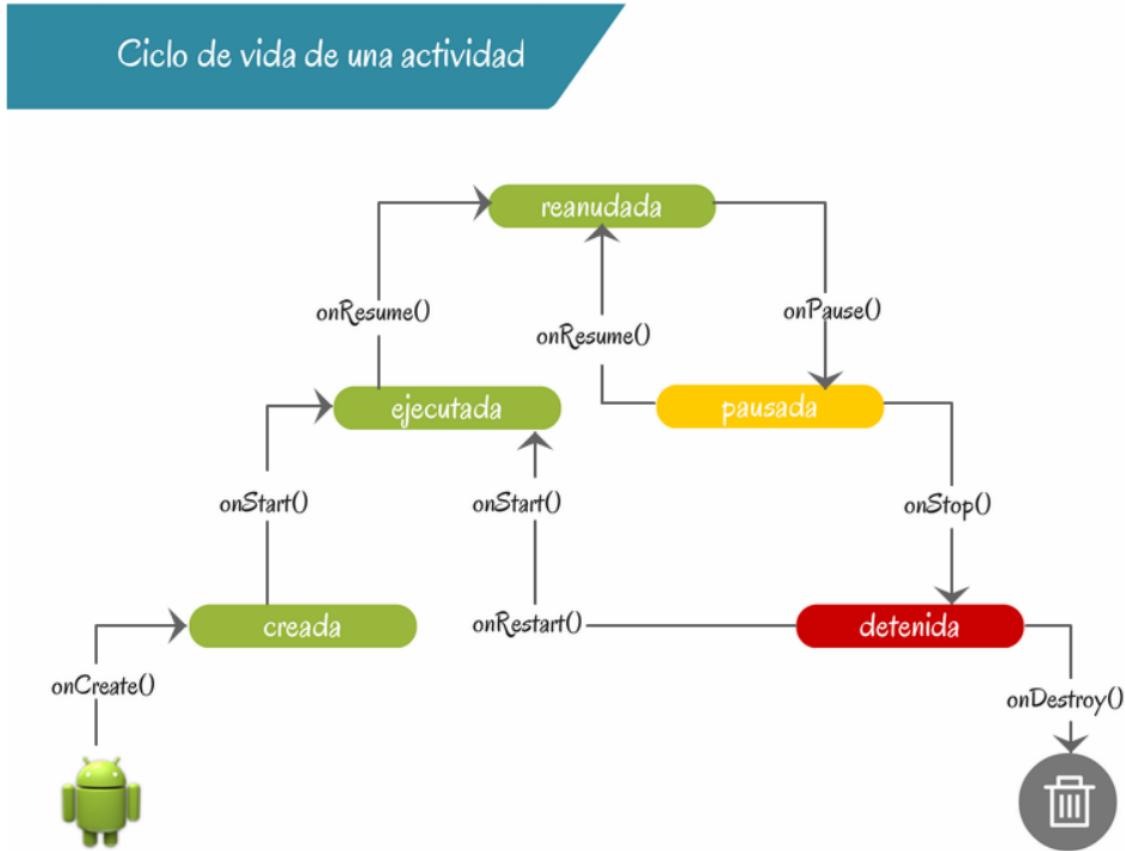
Un layout nos muestra como se van a ver nuestras diferentes pantallas. Es un conjunto de vistas agrupadas de una determinada forma. Vamos a disponer de diferentes tipos de layouts para organizar las vistas de forma lineal, en cuadrícula o indicando la posición absoluta de cada vista. Los layouts también son objetos descendientes de la clase View. Igual que las vistas, los layouts pueden ser definidos en código, aunque la forma habitual de definirlos es utilizando código XML ( hay un asistente gráfico para esta acción si se prefiere )

### Actividad (Activity)

Una aplicación en Android va a estar formada por un conjunto de elementos básicos de visualización, coloquialmente conocidos como pantallas de la aplicación. En Android cada uno de estos elementos, o pantallas, se conoce como actividad. Su función principal es la creación de la interfaz de usuario. Una aplicación suele necesitar varias actividades para crear la interfaz de usuario. Las diferentes actividades creadas serán independientes entre sí, aunque todas trabajarán para un objetivo común.

De cara a crear el código java no olvidar que: En el ciclo de vida de una activity Siempre existe el método onCreate() Es un lugar apropiado para empezar a trabajar nuestro código

## Ciclo de vida de una actividad



- **Creada:** Una actividad se ha creado cuando su estructura se encuentra en memoria, pero esta no es visible aun. Cuando el usuario presiona sobre el icono de la aplicación en su dispositivo, el método `onCreate()` es ejecutado inmediatamente para cargar el layout de la actividad principal en memoria.

- **Ejecutada:** Despues de haber sido cargada la actividad se ejecutan en secuencia a el método `onStart()` y `onResume()` . Aunque `onStart()` hace visible la actividad, es `onResume()` quien le transfiere el foco para que interactúe con el usuario.

- **Pausada:** Una actividad está en pausa cuando se encuentra en la pantalla parcialmente visible. Un ejemplo sería cuando se abren diálogos que toman el foco superponiéndose a la actividad. El método llamado para la transición hacia la pausa es `onPause()`

- **Detenida:** Una actividad está detenida cuando no es visible en la pantalla, pero aún se encuentra en memoria y en cualquier momento puede ser reanudada. Cuando una aplicación es enviada a segundo plano se ejecuta el método `onStop()`.

Al reanudar la actividad, se pasa por el método `onRestart()` hasta llegar a el estado de ejecución y luego al de reanudación.

- **Destruida:** Cuando la actividad ya no existe en memoria se encuentra en estado de destrucción. Antes de pasar a destruir la aplicación se ejecuta el método `onDestroy()`. Es común que la mayoría de actividades no implementen este método, a menos que deban destruir procesos (como servicios) en segundo

### Servicio (Service)

Un servicio es un proceso que se ejecuta “detrás”, sin la necesidad de una interacción con el usuario. Es algo parecido a un *demonio* en Unix o a un *servicio* en Windows. En Android disponemos de dos tipos de servicios: servicios locales, que son ejecutados en el mismo proceso y servicios remotos, que son ejecutados en procesos separados.

### Intención (Intent)

Una *intención* representa la voluntad de realizar alguna acción; como realizar una llamada de teléfono, visualizar una página web. Se utiliza cada vez que queramos:

- Lanzar una actividad
- Lanzar un servicio
- Enviar un anuncio de tipo broadcast
- Comunicarnos con un servicio

Los componentes lanzados pueden ser internos o externos a nuestra aplicación. También utilizaremos las *intenciones* para el intercambio de información entre estos componentes.

### Fragment

Un *fragment* está formado por la unión de varias vistas para crear un bloque funcional de la interfaz de usuario. Una vez creados los *fragments*, podemos combinar uno o varios *fragments* dentro de una actividad, según el tamaño de pantalla disponible. Esto es cómodo especialmente en tabletas.

### Receptor de anuncios (Broadcast receiver)

Un receptor de anuncios recibe anuncios *broadcast* y reacciona ante ellos. Los anuncios *broadcast* pueden ser originados por el sistema (por ejemplo: *Batería baja*, *Llamada entrante*) o por las aplicaciones. Las aplicaciones también pueden crear y lanzar nuevos tipos de anuncios *broadcast*.

## Proveedores de Contenido (Content Provider)

En muchas ocasiones las aplicaciones instaladas en un terminal Android necesitan compartir información. Android define un mecanismo estándar para que las aplicaciones puedan compartir datos sin necesidad de comprometer la seguridad del sistema de ficheros. Con este mecanismo podremos acceder a datos de otras aplicaciones, como la lista de contactos, o proporcionar datos a otras aplicaciones.

## 6. Creando Layouts

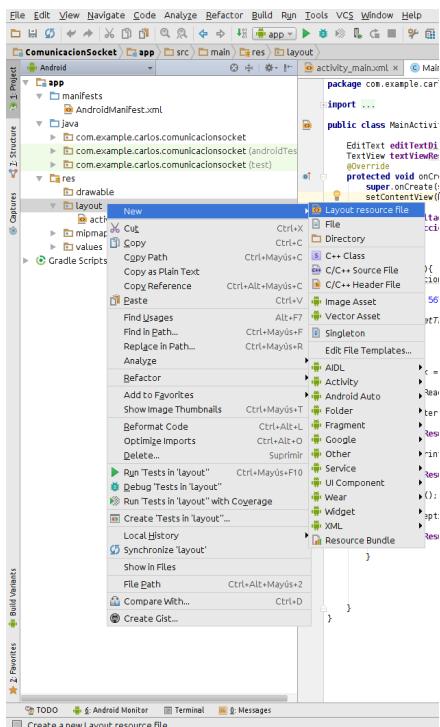
Un Layout es un elemento que representa el diseño de la interfaz de usuario de componentes gráficos como una actividad, fragmento o widget.

Ellos se encargan de actuar como contenedores de views para establecer un orden visual, que facilite la comunicación del usuario con la interfaz de la aplicación.

Los layouts pueden ser creados a través de archivos XML o con código Java de forma programática.

Dentro de la estructura de un proyecto en Android Studio existe el directorio **res** para el almacenamiento de recursos de la aplicación que se está desarrollando.

Las definiciones XML para los layouts se guardan dentro del subdirectorio **layout**.



Para crear un layout nuevo:, Sobre el directorio layout botón derecho --> New > Layout resource file.

Cada recurso del tipo layout debe ser un archivo XML, donde el elemento raíz solo puede ser un ViewGroup o un View. Dentro de este elemento puedes incluir hijos que definan la estructura del diseño.

Algunos de los view groups más populares son:  
**LinearLayout**  
**FrameLayout**  
**RelativeLayout**  
**TableLayout**  
**GridLayout**

## Programación Multimedia y Dispositivos Móviles

Observar el layout que se generó al crear el primer programa. El layout que generó se puede ver su código xml. Para ello abrimos el fichero correspondiente dentro de la carpeta layout: activity\_main.xml

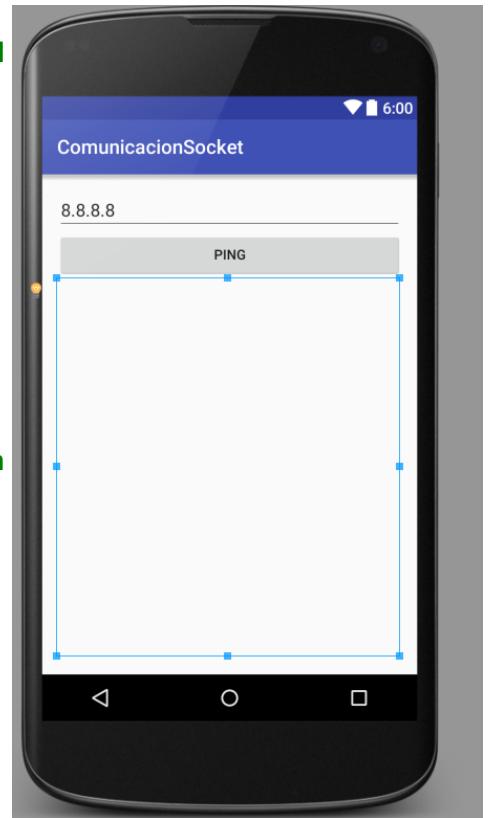
Aparecerá un renderizado de la posible presentación del layout en un dispositivo. Si miramos en la parte de abajo aparece “Design” y “Text” eligiendo este último podemos ver el código xml

Ejemplo de un layout:

```
<?xml version="1.0" encoding="utf-8"?>
<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: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="com.example.carlos.comunicationsocket.MainActivity"
    android:orientation="vertical">
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/editTextDireccionIP"
        android:text="8.8.8.8" />
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/buttonPing"
        android:id="@+id/buttonPing"
        android:onClick="ping" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:id="@+id/textViewResultado"
        android:layout_weight="1" />
</LinearLayout>
```



Agrupado en una estructura lineal por filas ( se van introduciendo los objetos uno encima de otro ) Nos encontramos Editext ( para introducir texto ), Button ( Un botón ), y un TextView ( salida de texto )

Cargar layout XML En Android— Al tener definido tu recurso, ya es posible inflar su contenido en la actividad. Esto podemos ver que nos lo ha generado ya el IDE cuando creamos nuestra primera aplicación. Mirar en MainActivity se puede ver este código:

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

R.layout.activity\_main es nuestro layout

## 6.2 Atributos De Un Layout

**Identificador de un view**— Existe un atributo que heredan todos los elementos llamado id. Este representa un identificador único para cada elemento. Lo que permite obtener una referencia de cada objeto de forma específica. La sintaxis para el id sería la siguiente:

android:id="@+id/nombre\_identificador"

Donde el símbolo '@' indica al parser interno de XML, que comienza un nuevo identificador para traducir. Y el símbolo '+' declara que dicho id no existe aún. Por ello se da la orden para crear el identificador dentro del archivo R.java a partir del string que proporcionamos.

**Este atributo es muy importante para acceder al elemento de diseño xml luego desde nuestro código java y usarlo.**

Sea por ejemplo:

```
<TextView  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:textAppearance="?android:attr/textAppearanceLarge"  
    android:id="@+id/textViewResultado"  
    android:layout_weight="1" />
```

Acceder a ese elemento desde código sería:

```
textViewResultado = (TextView) findViewById(R.id.textViewResultado);
```

Observar findViewById() lo usaremos mucho.

La estrategia es sencilla. Simplemente se declara un objeto del tipo buscado y luego asignar el resultado que produce `findViewById()`.

Como se puede ver se ha usado el **id** que está en el recurso **R**

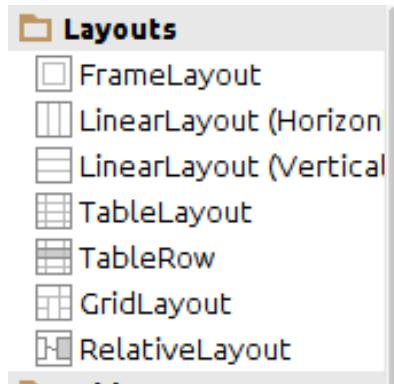
También vemos el cast (`TextView`) Haremos siempre el cast correspondiente al tipo que se precise.

Otros atributos que aparecen son los padding, las dimensiones de los objetos ( `width` y `height` ) etc

### 6.3 Viewgroups

Para todo diseño que hagamos nos apoyaremos en una estructura para ir ubicando nuestros elementos.

En la paleta se nos muestran las diferentes opciones:

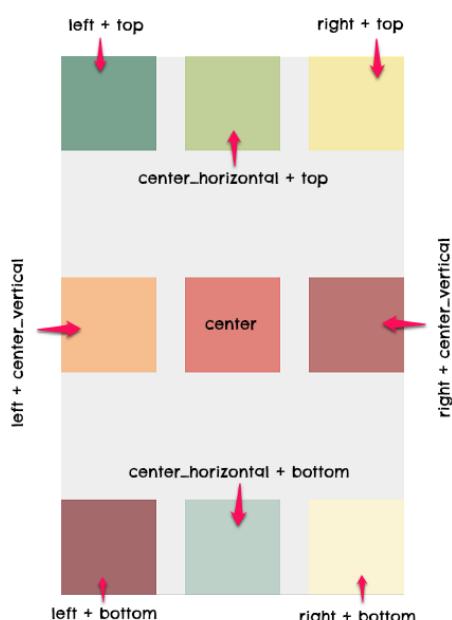
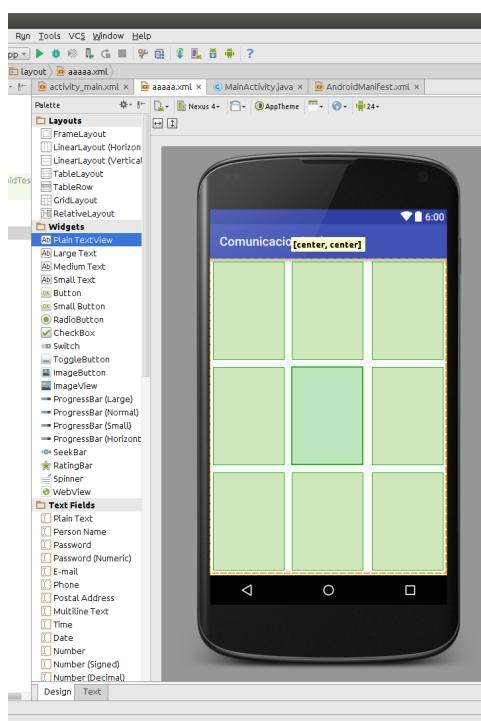


Vamos a trabajar con varios de ellos y practicar

#### 6.3.1 FrameLayout

Este no realiza ninguna distribución de las vistas, simplemente las coloca unas encima de otras. Esto le evita tener que relacionar los tamaños de unas vistas con los de las demás, por lo que se ahorra recorridos del árbol de vistas, tardando menos en mostrar su contenido.

Observemos las opciones que nos da cuando arrastramos un objeto a la pantalla que tiene un framelayout:



Disponemos de unas pocas posiciones básicas donde ubicar. Este viewgroup tiene la circunstancia especial que permite poner objetos encima uno de otro y eso pudiera ser interesante en alguna ocasión.

Ej. Se quiere poner un dibujo y que su comportamiento sea el mismo que un botón.

Bastará con poner la imagen debajo, un botón encima y hacer este último transparente. De cara al usuario tendremos que actúa la imagen como un botón

### Actividad 1

Vamos a hacer una operación sencilla en la que ponemos una imagen y un botón encima transparente

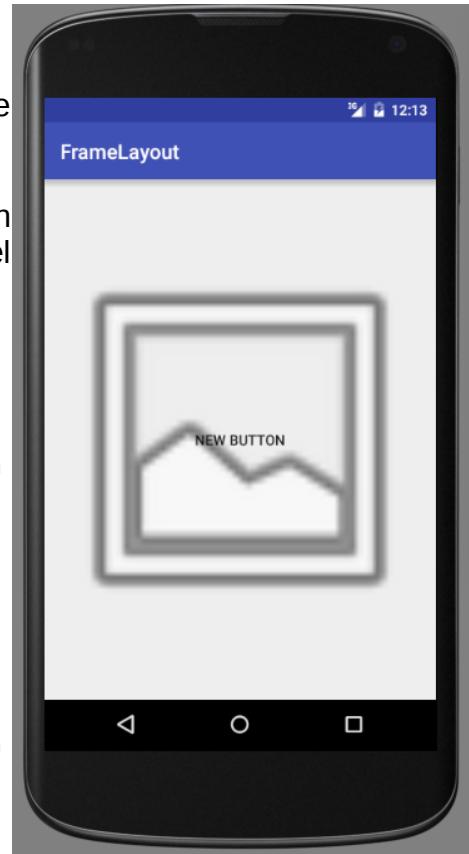
Crear un proyecto nuevo llamado FrameLayout Y en la vista diseño de texto del layout modificamos el relativelayout que pone por defecto por el framelayout

En el código xml nos aparece:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
```

Y nosotros pondremos:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
```



El Ide nos cambiará el texto que está en el cierre también de relativelayout a framelayout

Una vez regresamos al modo diseño Agregamos un ImageView en la propiedad "src" pulsamos sobre el ícono de los tres puntos y nos desplegará una ventana

Elegimos la pestaña Drawable y elegimos ic\_menu\_gallery

De esta forma estamos poniendo ya una imagen

Observar que no está ocupando toda la pantalla. Vamos a introducir el concepto de dos valores que se usan mucho en nuestros diseños:

**match\_parent**

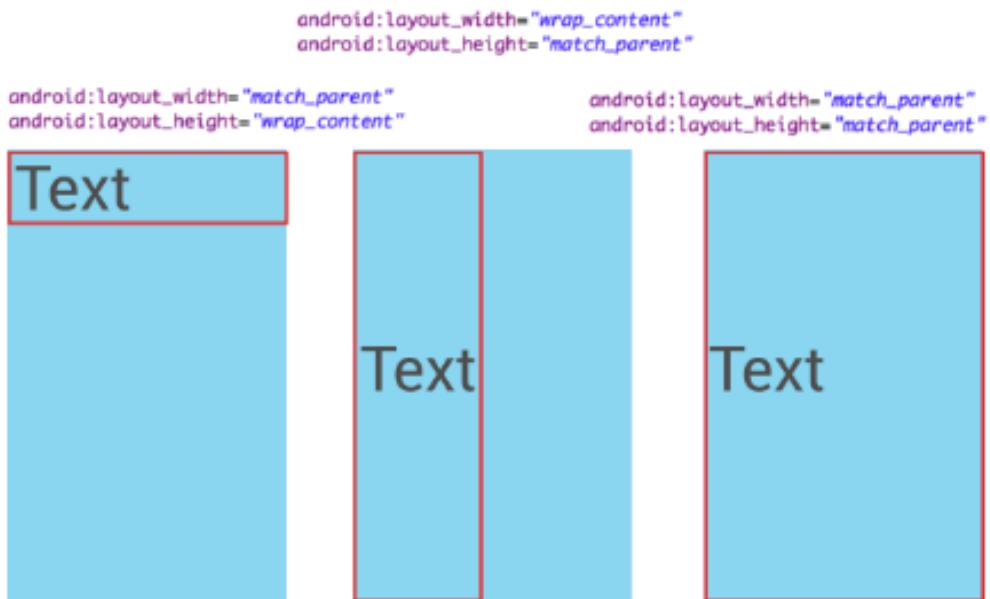
**wrap\_content**

Estas propiedades se usan en width y height habitualmente

wrap\_content hace que se ajuste al contenido, match\_parent que ocupe el espacio que le permita el padre

Una ayuda visual:

## match\_parent



Ahora, establecemos:

`layout:width match_parent  
layout:height match_parent`

y podemos observar que nos ocupa toda la pantalla la imagen

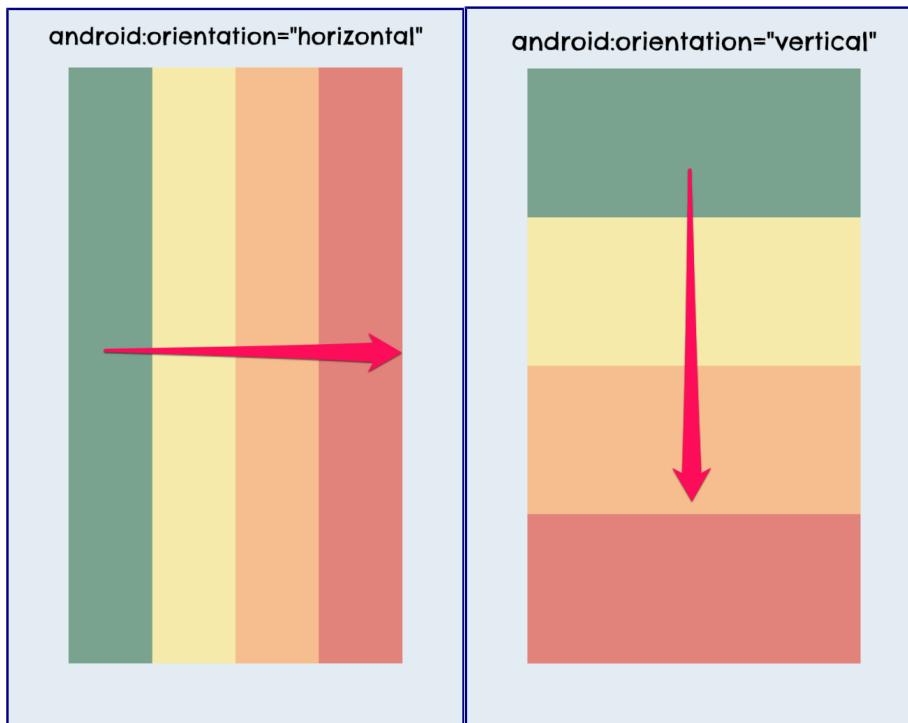
Ahora arrastramos un Button y lo ubicamos en la misma posición que hicimos con el ImageView hacemos lo mismo de antes con width y height

## Programación Multimedia y Dispositivos Móviles

Recordar que hay multitud de dispositivos con pantallas diferentes. Usar tamaños fijos mediante **dp** nos puede dar renderizados inapropiados. **match\_parent** y **wrap\_content** evitan esa problemática.

### 6.3.2 LinearLayout

Este es un viewgroup sencillo que permite ir colocando los objetos uno detrás de otro según la dirección escogida ( vertical, horizontal )



#### Actividad 2

vamos a usar el LinearLayout vertical para mostrar como ocupar los espacios.

Recrear el layout de la imagen para conseguir que el textView ocupe el mismo espacio que el button vamos a hacer uso de una propiedad muy útil:

`layout:weight`

En esa propiedad establecemos el peso que tiene el elemento respecto a los demás dentro de nuestro viewgroup

Si establecemos ambos elementos a 1 le estamos diciendo que queremos que cojan el mismo espacio ambos



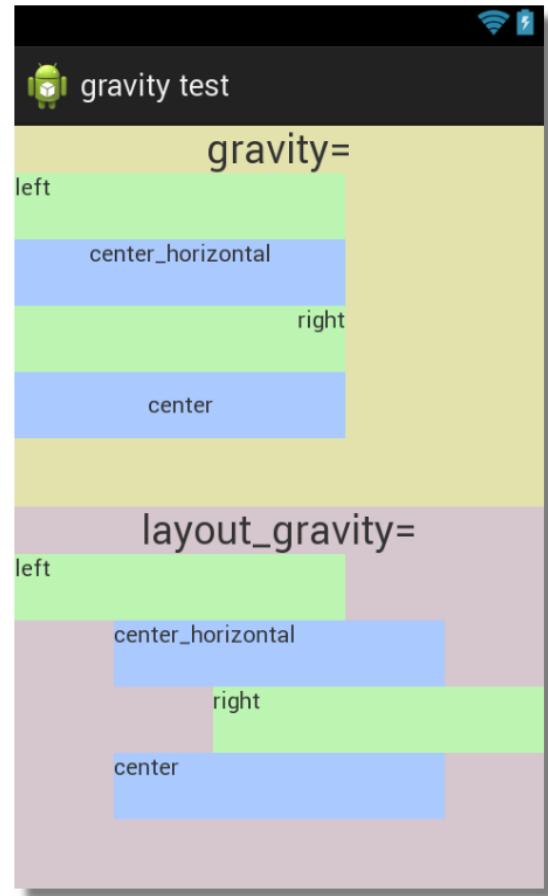
## Programación Multimedia y Dispositivos Móviles

para que esto funcione apropiadamente en layout:height le ponemos: wrap\_content

Otra propiedad importante es `gravity` y `layout_gravity`. Ambas nos ayudan a ubicar los elementos. Para entenderlo mejor ver un ejemplo:

Como se puede observar `gravity` afecta a la posición interna. Así pues el texto nos queda ubicado según lo establecido en esa propiedad.

`Layout_gravity` viene a determinar la posición respecto al padre. Observar que los `textview` se alinean ellos mismos no su contenido ( el texto )

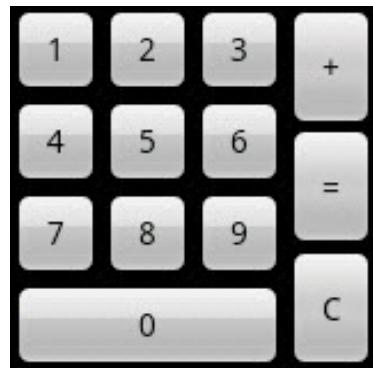


Los viewgroup ( frameLayout, LinearLayout,... ) se pueden anidar unos dentro de otros para conseguir los diseños deseados.

Es conveniente usar el mínimo número de layouts posibles porque supone un importante gasto de recursos en el renderizado

Aún con ello para practicar su uso vamos a realizar el siguiente layout utilizando únicamente LinearLayout ( habrá varios anidados )

### Actividad 3



### 6.3.4 GridLayout

Hemos ido directamente al GridLayout sin nombrar el tablelayout al ser muy parecidos uno a otro pero más potente el GridLayout. Lo único que se debe tener en cuenta para el tablelayout es que, al igual que con html, vamos creando filas de cada tabla mediante TableRow y dentro vamos ubicando los diferentes elementos

Así pues, a diferencia del TableLayout indicaremos el número de filas y columnas como propiedades del layout, mediante **android:rowCount** y **android:columnCount**.

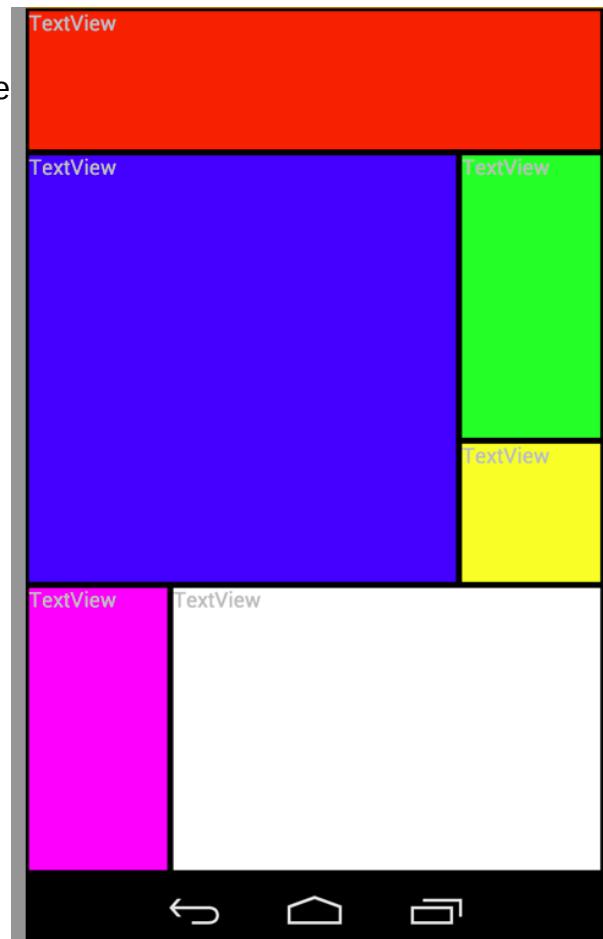
Con estos datos ya no es necesario ningún tipo de elemento para indicar las filas, (no hace falta TableRow) sino que los diferentes elementos hijos se irán colocando ordenadamente por filas o columnas (dependiendo de la propiedad android:orientation) hasta completar el número de filas o columnas indicadas en los atributos anteriores.

Adicionalmente, también tendremos disponibles las propiedades **android:layout\_rowSpan** y **android:layout\_columnSpan** para conseguir que una celda ocupe el lugar de varias filas o columnas.

Existe también una forma de indicar de forma explícita la fila y columna que debe ocupar un determinado elemento hijo contenido en el GridLayout, y se consigue utilizando los atributos **android:layout\_row** y **android:layout\_column**.

#### Actividad 4

Realizar el layout de la figura mediante gridlayout



Para ir preparando el camino de una de las actividades que vamos a realizar. Hacer el layout para la siguiente imagen:

**Actividad 5**



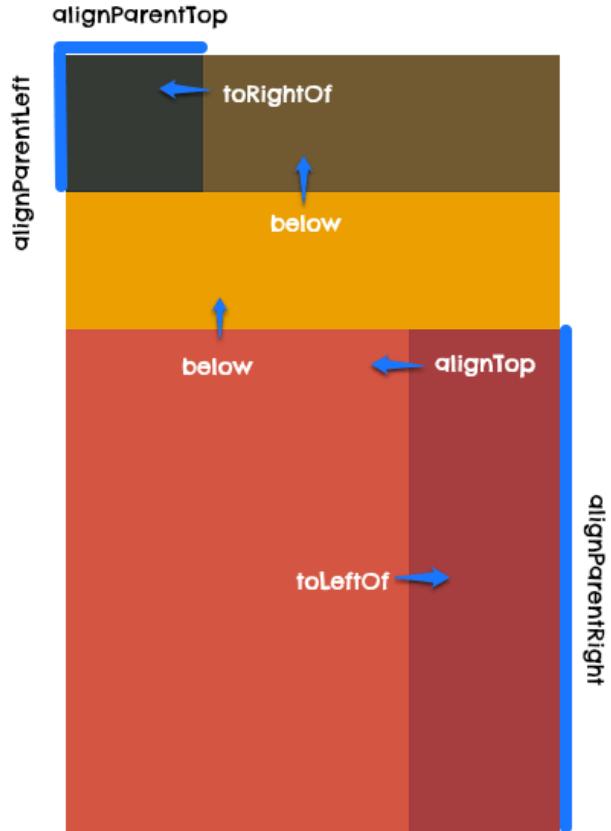
#### 6.3.4 RelativeLayout

Es el más flexible de todos los viewgroups. El RelativeLayout permite alinear cada hijo con referencias subjetivas de cada hermano.

Con el RelativeLayout expresaremos cosas como: "el botón estará por debajo del texto" o "la imagen se encuentra a la derecha de la descripción".

En ninguno de los casos se habla de posición absoluta o un espacio determinado. Simplemente describimos la ubicación.

Ejemplo De Un Relative Layout:



### Dimensionando los elementos en el layout

Para los diferentes tamaños de pantalla ( principalmente distinguir tablet de teléfono ) usamos distinto layout ( ficheros que guardan el layout con nombre identificativo de para que tamaño de pantalla debe aplicarse): layout-w600dp, layout-w720dp ( con esta especificación se vería afectado también por el cambio de giro del dispositivo )

Para que no se apliquen con giro de pantalla del dispositivo usamos:

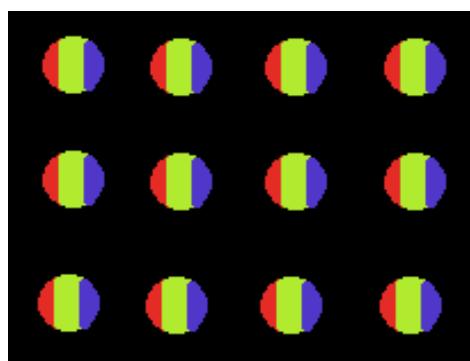
layout-sw600dp, layout-sw720dp

Ahora bien, nuestros cálculos tendrán que establecerse respecto a esos “dp” ¿ y de qué estamos hablando ? Empecemos por el principio:

### Píxel (px)

Un píxel es un concepto. Hace referencia a la unidad mínima que puede ser renderizada en una pantalla.

Por mucho que se quieran juntar los píxeles siempre va a haber espacio entre ellosUna imagen vale más que mil palabras:



Cuanto menos vacío exista entre ellos, más píxeles entrarán para un mismo área, luego más definición para la imagen mostrada

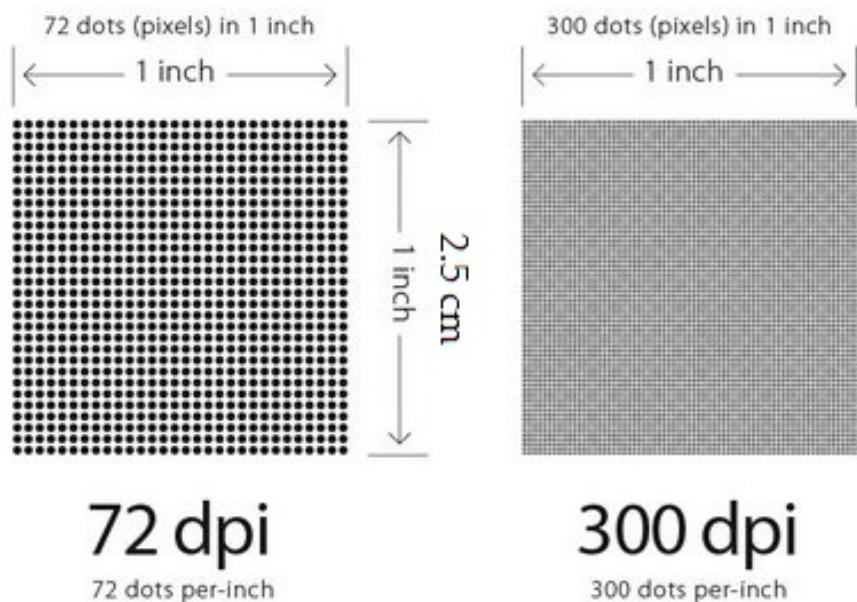
### Resolución de Pantalla

La resolución de una pantalla establece el número de píxeles que tiene una pantalla. Se suele expresar en función de su número horizontal y vertical Ej. “1024x768”

Evidentemente no será lo mismo tener esa resolución en una pantalla de tres pulgadas que en una de cincuenta pulgadas. Notaremos una gran diferencia de definición entre ambas. Para establecer una medida que permita comparar definición usamos DPI

### Dots per Inch (DPI)

Simboliza el número de píxeles existentes en un área equivalente a una pulgada(2.54cm)



En la imagen anterior se puede observar fácilmente que a mayor cantidad de dpi mejor definición de la imagen

Hay cuatro grupos de densidades para trabajar en android: low, medium, high, extrahigh

### Density-independent Pixel (dp)

Para ayudar a que los escalados de un layout sean apropiados y que cuando se diseñe un layout no haya que estar tan pendiente de la enorme variabilidad que se puede encontrar de densidades y tamaños de pantalla. Google introdujo los “**dp**”.

Se consideró que según lo que había en el mercado había una densidad media de 160dpi en los dispositivos.

Un **dp** es 1px en una pantalla de 160dpi de densidades

## Programación Multimedia y Dispositivos Móviles

Para definir los layouts, usaremos los dps. El sistema ya se encargará de traducirlos a píxeles físicos mediante la fórmula:

$$px = dp * (dpi/160)$$

Ejemplos:

- Diseñamos una línea de 3dps, Si el dispositivo real que vaya a renderizar la imagen es de 160dpi entonces esos 3dps se traducen en 3píxeles
- Supongamos ahora un nexus 5 que tiene 445dpi (mucho mayor que la densidad media de los dispositivos). Esos mismos 3 dps se transforman en 8 píxeles físicos para esa pantalla.

Ahora bien, en muchas ocasiones esa ayuda que son los dps no son suficientes para resolver la problemática de la representación en los dispositivos. Para generar un layout para un dispositivo específico y conocer los dps que tiene tendremos que hacer algunos cálculos:

1. Observemos la fórmula de google:  $px = dp * (dpi / 160)$
2. Para conocer los dp primero tenemos que conocer los dpi del dispositivo.
3. Una información que sí que obtendremos con facilidad de un dispositivo ( se suele publicar en las especificaciones del dispositivo ) son las pulgadas de su diagonal y su resolución. Supongamos que el dispositivo tiene una resolución de: 2560x1600 Px y 10 pulgadas de diagonal
4. Usando el teorema de pitágoras tenemos que los píxeles de la diagonal son:  
 $Px \text{ diagonal} = \sqrt{1600^2 + 2560^2} \Rightarrow 3018px$  en la diagonal
5. Como esa misma diagonal sabemos que es de 10pulgadas:  
 $3018px / 10 \text{ pulgadas} = 302dpi$
6. Ahora ya podemos aplicar la fórmula de google:  
 $2560 = dp * (302 / 160) \Rightarrow dp = 2560 * 160 / 302$

Tenemos pues que  $dp = 1356$  Y ese es el ancho de pantalla con el que tenemos que trabajar para nuestro diseño.

Por ejemplo, sabremos que tres objetos de 400dps nos caben en ese ancho

## 7. Creando el código

Una buena forma de aprender es mirando códigos de ejemplo. Un lugar que concentra mucha información es:

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

Hay una sección que nos informa de los diferentes controles etc

También nos referencia ejemplos. Aquí el enlace en github:

<https://github.com/googlesamples/>

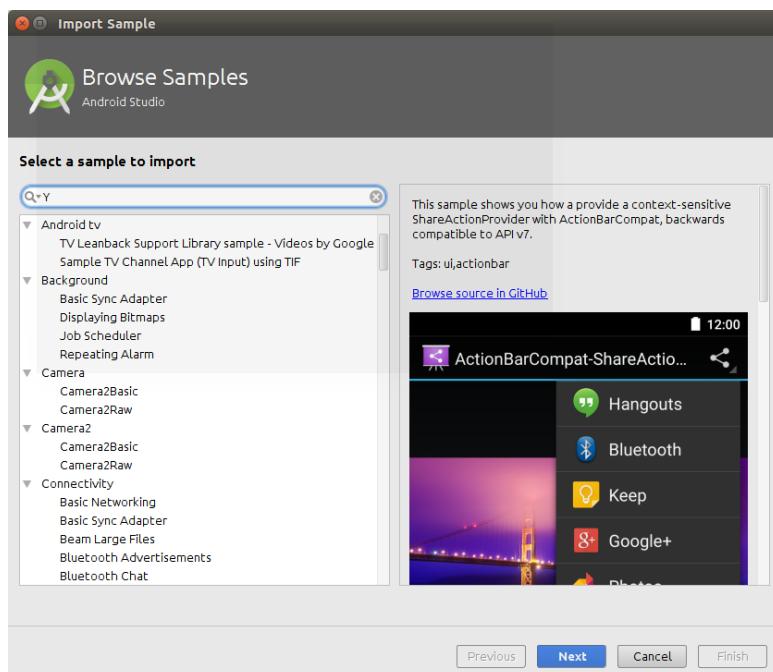
¿ Cómo importar proyectos de GitHub?

File>New>Project from Version Control>Git

Por ejemplo: [https://github.com/codepath/intro\\_android\\_demo.git](https://github.com/codepath/intro_android_demo.git)

En android studio ya tenemos ejemplos que podemos cargar:

File- → New- → Import sample



## 7.1 Activities

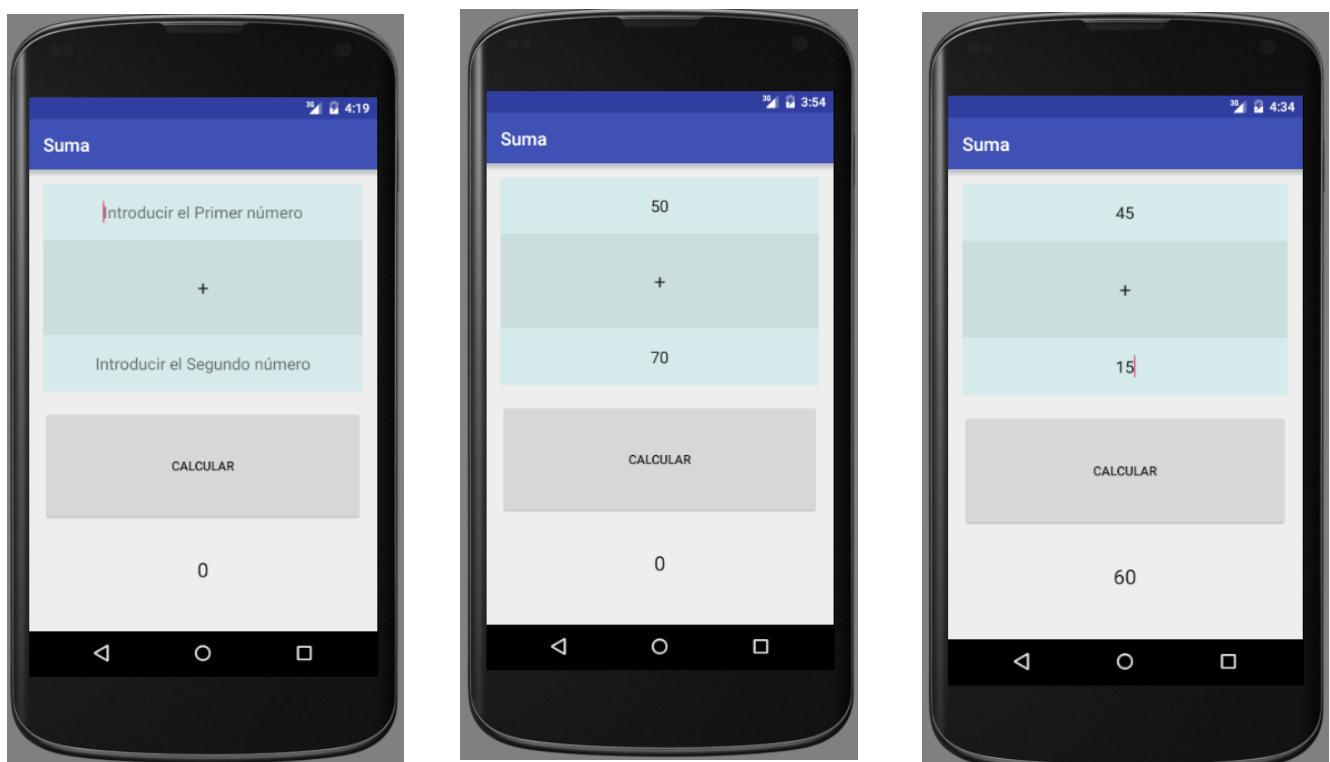
Como una primera aproximación una definición de Activity sería cada una de las pantallas ( y su código asociado ) que se muestra en una aplicación

Así pues habrá un fichero .java para el código y un .xml para el layout de la Activity

Quizás una forma que nos introduzca de una forma apropiada es ir haciendo ejercicios

### Actividad 6

Realizar la siguiente aplicación



La parte de layout ya sabemos como hacerla. En cuanto a ejecutar una acción de botón, habaremos de los listener

### 7.1.2 Escuchar y manejar eventos onClick

Varias formas de hacerlo:

#### 1. Declarando el método en el fichero xml

```
<Button  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="@string/calcular"  
    android:id="@+id/buttonCalcular"  
    android:longClickable="false"  
    android:onClick="calcular" />    <!-- observar el onClick -->
```

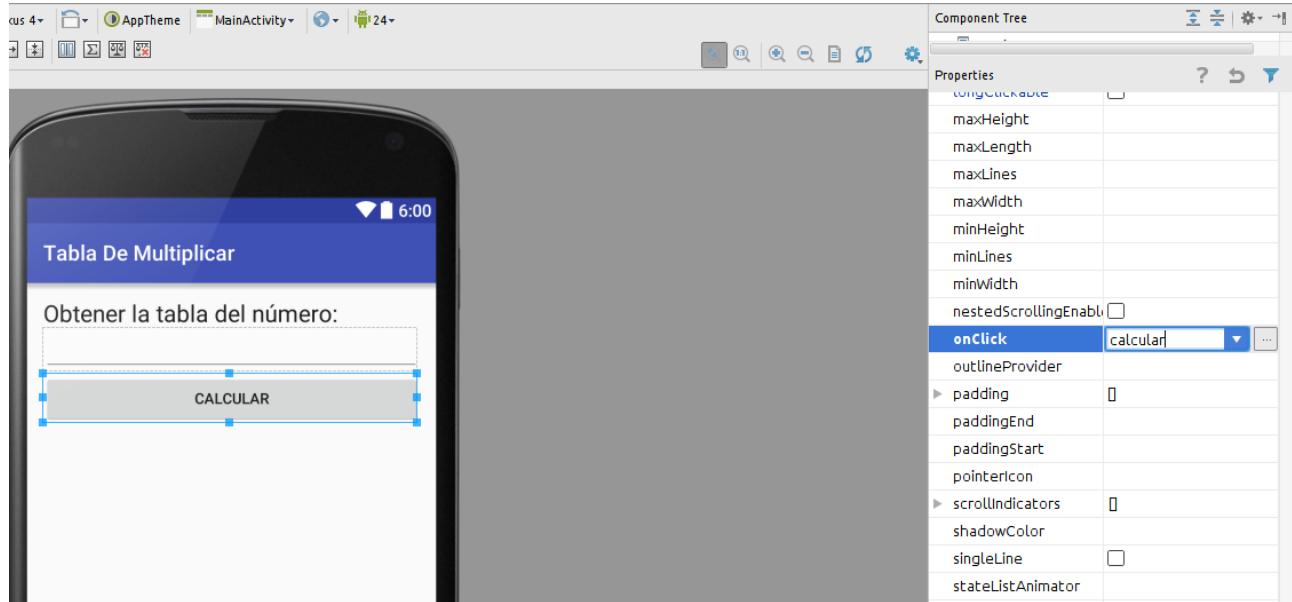
De esta forma en el código podemos llamar al método llamado onClick(). Lo que está ocurriendo es que internamente android nos está creando el listener

El código de MainActivity nos queda:

```
public class MainActivity extends AppCompatActivity {  
    TextView resultado;  
    Button boton;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        resultado = ((TextView) findViewById(R.id.textViewResultado));  
        boton = (Button) findViewById(R.id.button);  
    }  
  
    void calcular(View view){  
        int num1 =  
Integer.parseInt(((EditText) findViewById(R.id.editTextNumero1)).getText().toString());  
        int num2 =  
Integer.parseInt(((EditText) findViewById(R.id.editTextNumero2)).getText().toString());  
        int res = num1 + num2;  
        resultado.setText(num1 + " + " + num2 + " = " + res);  
    }  
}
```

## Programación Multimedia y Dispositivos Móviles

Esta acción del xml lo podemos realizar en las propiedades del diseño del xml desde el IDE:



2. Agregando un listener en el código Java:

```
Button b = (Button) findViewById(R.id.button1);
b.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onclick(View v) {
        //aquí el código
    }
});
```

3. Agregando un interfaz a la activity:

```
public class SurvivingClickActivity extends Activity implements
View.OnClickListener {

    .
    .

    Button b = (Button) findViewById(R.id.button1);
    b.setOnClickListener(this);

    @Override
    public void onClick(View v) {
        int id = v.getId();
        switch(id) {

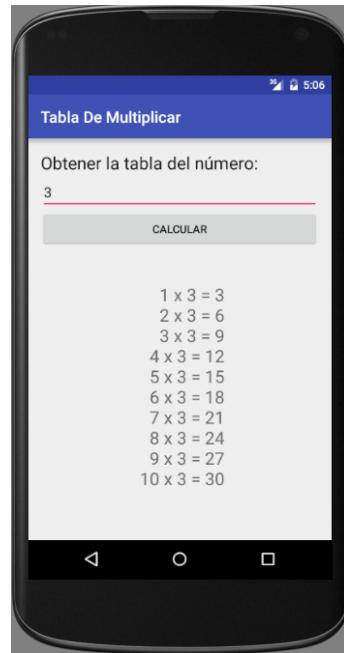
            case R.id.button1: // aquí el código del botón
                .
                .
        }
    }
}
```

## Programación Multimedia y Dispositivos Móviles

Anteriormente no se disponía de la opción desde xml y hacerlo desde código permite mucho más potencial y control adicionalmente a tener una mayor separación del diseño y el código. Pero en muchas ocasiones es suficiente desde xml

### Actividad 7

Realizar la siguiente Activity usando las tres formas que hemos visto de implementar el listener



Ahora ya toca hacer una aplicación sencilla, útil y plenamente funcional

### Actividad 8

Realizar una calculadora sencilla con la interfaz que se muestra en la imagen



## 7.2 Intents

### 7.2.1 Introducción a los Intents

Un intent sirve para invocar componentes: Activities, Services, AsyncTask ( hilos en background ), broadcast receivers, Content Providers, ...

Los Intents nos permiten llamar a aplicaciones externas a la nuestra, lanzar eventos a los que otras aplicaciones puedan responder, lanzar alarmas etc.

Ej. Supongamos la siguiente activity:

```
public class MiActivity extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.MiActivity);  
    }  
}
```

Hay que registrar el Intent en AndroidManifest.xml:

```
<activity android:name=".MiActivity" android:label="Mi Activity">  
    <intent>  
        <action android:name="nuestra.accion.nombreAccion">  
            <category android:name="android.intent.category.DEFAULT">  
            </category>  
        </action>  
    </intent>  
</activity>
```

Ahora se lanza la activity con el Intent:

```
public static void invokeMiActivity(Activity activity){  
    String actionName= "nuestra.accion.nombreAccion";  
    Intent intent = new Intent(actionName);  
    activity.startActivity(intent);  
}
```

## Programación Multimedia y Dispositivos Móviles

Se puede tomar desde la Activity lanzada por el Intent la información de ese propio intent. Esto se hace así:

```
//Para que funcione poner dentro de onCreate() de la Activity
Intent intent = this.getIntent();
if (intent == null){
    Log.d("Tag", "No hubo llamada desde Intent.")
}
```

### 7.2.2 Intents disponibles en Android

En developer.android.com/guide/appendix/g-app-intents.html se puede encontrar una lista con las aplicaciones disponibles en Android junto con los intents que las invocan.

Por ejemplo, para el navegador web, tenemos dos acciones, ACTION\_VIEW \*\* y ACTION\_WEB\_SEARCH, que abren el navegador en una url específica o realizan una búsqueda.

En el caso del dialer (marcador), tenemos las acciones ACTION\_CALL y ACTION\_DIAL, que vienen dadas por la URI tel:numero\_de\_telefono, la diferencia entre estas dos acciones, es que ACTION\_CALL realiza la llamada al número de la URI, y ACTION\_DIAL solo lo marca, pero no realiza la llamada.

Ejemplos:

```
public static void invokeWebBrowser(Activity activity){
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse("http://www.google.com"));
    activity.startActivity(intent);
}
```

```
public static void invokeWebSearch(Activity activity){
    Intent intent = new Intent(Intent.ACTION_WEB_SEARCH);
    intent.setData(Uri.parse("http://www.google.com"));
    activity.startActivity(intent);
}
```

```
public static void dial(Activity activity){
    Intent intent = new Intent(Intent.ACTION_DIAL);
    activity.startActivity(intent);
}
```

```
public static void call(Activity activity){
    Intent intent = new Intent(Intent.ACTION_CALL);
    intent.setData(Uri.parse("tel:555-555-555"));
    activity.startActivity(intent);
}
```

## Programación Multimedia y Dispositivos Móviles

```
public static void showMapAtLatLong(Activity activity){  
    Intent intent = new Intent(Intent.ACTION_VIEW);  
    intent.setData(Uri.parse("geo:0,0?z=4&q;=restaurantes"));  
    activity.startActivity(intent);  
}
```

### Actividad 9

Realizar una aplicación que al pulsar el botón nos lleve a la página escrita por el usuarios



En el xml de AndroidManifest deberá aparecer:

```
<activity>  
    <intent>  
        <action android:name="android.intent.action.VIEW">  
            <data android:scheme="http">  
            </data>  
            <data android:scheme="https">  
            </data></action>  
    </intent>  
</activity>
```

Observar que es el sistema quién tiene que decidir a quién llamar. ¿ Cómo lo hace ? Observa la Uri que se le envía, ve http y en consecuencia llama al navegador:

```
intent.setData(Uri.parse("http://www.google.com"));
```

**Este tipo de Intent los llamamos implícitos ya que no se ha nombrado ningún componente al que se va a llamar. Como contrapartida se llaman explícitos a los que lo hacen**

### 7.2.3 Intent Explícito

Primero una nota importante a tener en cuenta: Cuando creamos una Activity mediante el IDE este nos modifica convenientemente AndroidManifest.xml para que nos aparezca la nueva activity. De no hacer esto NO podrá ser llamada con un intent explícito. Es pues algo que tenemos que tener en cuenta si lo hacemos nosotros a mano

Imaginemos que queremos llamar a una Activity llamada: Activity2

La forma de llamarla sería:

```
Intent intent = new Intent(activity, Activity2.class);
activity.start(intent);
```

En el caso de los Intent explícitos no hay que registrarlos en AndroidManifest. Esto es porque estamos llamando a la actividad en el intent directamente con el nombre de la clase

### 7.2.4 Paso de información con el Intent

#### 7.2.4.1 Primera opción

La forma de hacer esto es mediante clave/valor:

```
intent.putExtra("clave", valor);
```

Una forma útil y elegante es hacer uso de los Bundle. Donde agrupamos lo que queremos enviar:

```
Bundle bundle = new Bundle();
bundle.putString("nombre", "Alfonso Hernández");
intent.putExtras(bundle)
```

## Programación Multimedia y Dispositivos Móviles

Bundle permite pasar los siguientes tipos de datos:

```
putBoolean (String key, boolean value)
putInt (String key, int value)
putString (String key, String value)
putAll (Bundle map)
putBinder (String key, IBinder value)
putBooleanArray (String key, boolean[] value)
putBundle (String key, Bundle value)
putByte (String key, byte value)
putByteArray (String key, byte[] value)
putChar (String key, char value)
putCharArray (String key, char[] value)
putCharSequence (String key, CharSequence value)
putCharSequenceArray (String key, CharSequence[] value)
putCharSequenceArrayList (String key, ArrayList<CharSequence> value)
putDouble (String key, double value)
putDoubleArray (String key, double[] value)
putFloat (String key, float value)
putFloatArray (String key, float[] value)
putIntArray (String key, int[] value)
putIntegerArrayList (String key, ArrayList<Integer> value)
putLong (String key, long value)
putLongArray (String key, long[] value)
putParcelable (String key, Parcelable value)
putParcelableArray (String key, Parcelable[] value)
putParcelableArrayList (String key, ArrayList<? extends Parcelable> value)
putSerializable (String key, Serializable value)
putShort (String key, short value)
putShortArray (String key, short[] value)
putSparseParcelableArray (String key, SparseArray<? extends Parcelable> value)
putStringArray (String key, String[] value)
putStringArrayList (String key, ArrayList<String> value)
```

Ahora bien, ¿ cómo se recibe la información desde el otro lado ? ( desde la actividad que ha sido lanzada por el Intent ):

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

Cuando se cierre la Activity que se ha llamado hay un método específico que nos permite recoger si nos quiere pasar algún resultado:

```
protected void onActivityResult(int requestCode, int resultCode, Intent
intent) {
    String resultado = intent.getStringExtra("confirmation");
}
```

Tener en cuenta que una forma para cerrar un Activity es llamando a: finish()

### 7.2.4.2 Segunda Opción

Lo anterior tiene varios problemas:

- las etiquetas que ponemos para enviar un dato pueden ser problemáticos si no ponemos exactamente lo mismo en ambas activities

Para resolverlo podemos crear una clase pública accesible para las dos activities que nos garantice que las etiquetas serán las mismas siempre:

```
public class ActivityHelper {
    public static final String KEY_USER_NAME = "key_user_name";
    public static final String KEY_USER_EMAIL = "key_user_email";
    public static final String KEY_USER = "key_user";
}
```

- Segundo problema: ¿ qué ocurre si queremos pasar objetos personalizados ?

Para eso existe una opción ya incluida en android llamada: Parcelable

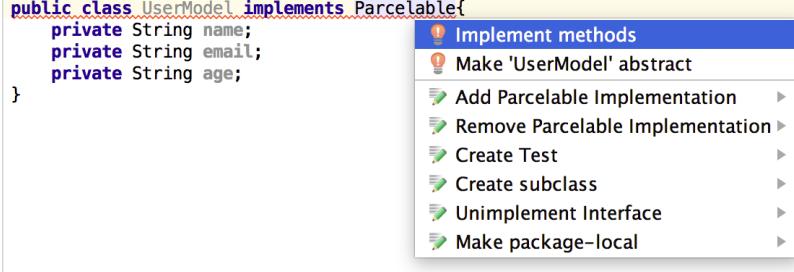
Es una interfaz que nos permite determinar como vamos a serializar nuestros objetos y como los reconstruimos ( otra opción es mediante Serializable de java pero se demuestra que es mucho mejor en gasto computacional Parcelable )

## Programación Multimedia y Dispositivos Móviles

Ej. Tenemos una clase de la que queremos enviar objetos:

```
public class UserModel {  
    private String name;  
    private String email;  
    private String age;  
}
```

Usamos implements y nos apoyamos en las opciones del ide para que nos genere los override:



La clase nos quedará ahora:

```
public class UserModel implements Parcelable {  
    private String name;  
    private String email;  
    private String age;  
    public UserModel(Parcel in){  
        // El constructor de UserModel aquí  
    }  
  
    protected UserModel(Parcel in) {  
        name = in.readString();  
        email = in.readString();  
        age = in.readString();  
    }  
  
    public static final Creator<UserModel> CREATOR = new  
Creator<UserModel>() {  
    @Override  
    public UserModel createFromParcel(Parcel in) {  
        return new UserModel(in);  
    }  
    @Override  
    public UserModel[] newArray(int size) {  
        return new UserModel[size];  
    }  
};  
    @Override  
    public int describeContents() {  
        return 0;  
    }  
    @Override  
    public void writeToParcel(Parcel dest, int flags) {  
        dest.writeString(name);  
        dest.writeString(email);  
        dest.writeString(age);  
    }  
}
```

## Programación Multimedia y Dispositivos Móviles

Observar que con Creator decimos como queremos que nos construya el objeto en la activity que recibe el objeto. WritetoParcel() escribe los atributos de nuestro objeto para enviarlo en el intent serializado

Ahora nos queda el proceso de enviar nuestro objeto:

```
Intent i = new Intent(CONTEXT, CLASS_NAME.class);
i.putExtra(ActivityHelper.KEY_USER, userObject);
```

Y en la segunda activity

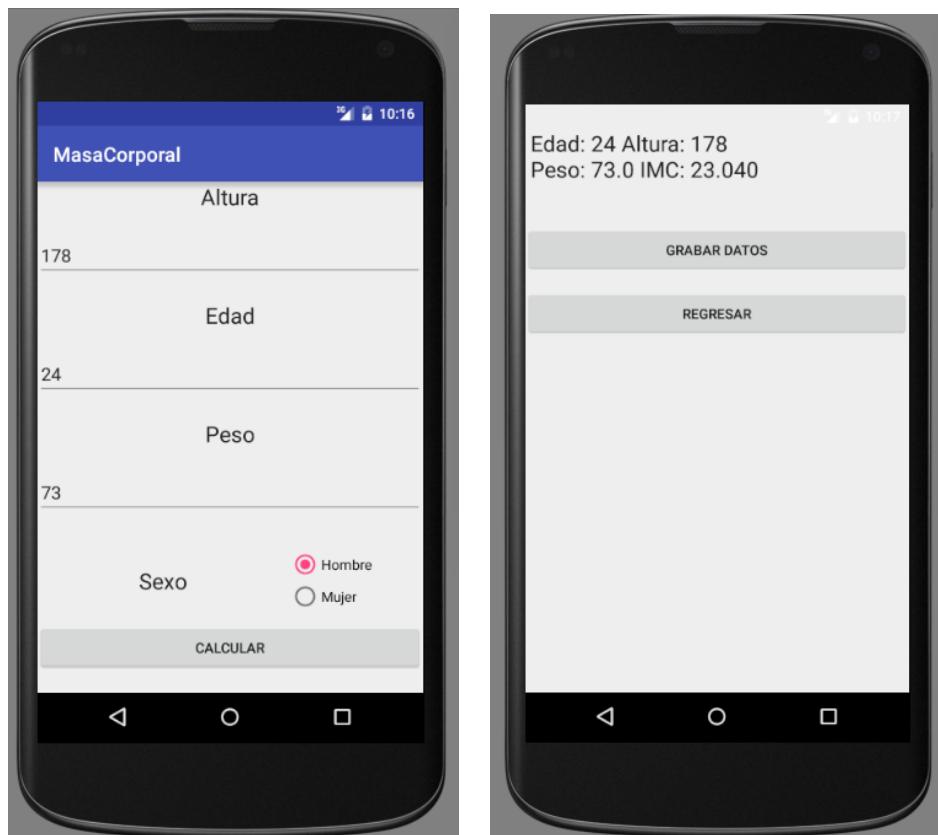
```
UserModel user = (UserModel)
getIntent().getParcelableExtra(ActivityHelper.KEY_USER);
```

Éste procedimiento puede parecer que “ensucia” nuestras clase original porque establece todo lo de la transmisión de información incorporado cuando es algo que únicamente sería preciso para transmitir entre activities

Hay librerías que podemos incorporar que nos dan otras opciones. Para ampliar, echar un vistazo a: <http://parceler.org/>

### Actividad 10

Realizar una aplicación que calcule los datos de índice de masa corporal. En la primera Activity se introducen los datos y al pulsar el botón calcular nos muestra la segunda Activity con los resultados. En esta segunda Activity el botón regresar ejecuta el finish(). El botón grabar datos quedará en el layout pero no lo vamos a implementar en el código de momento.



## Programación Multimedia y Dispositivos Móviles