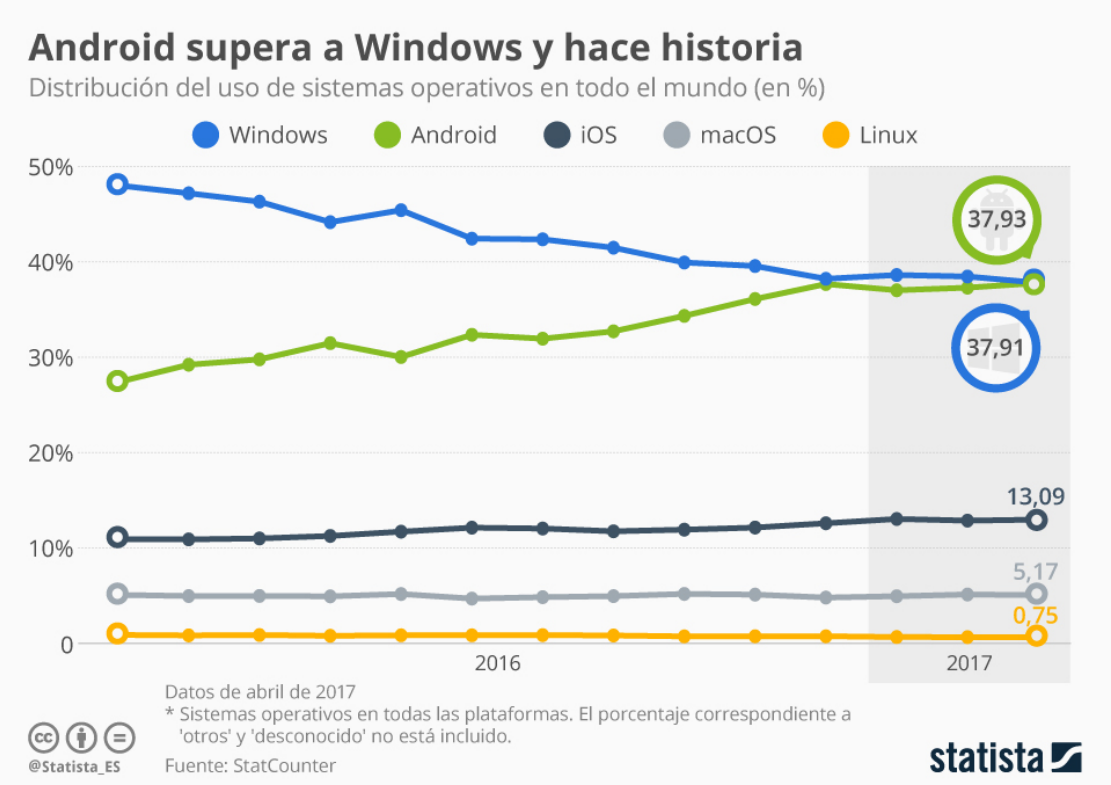
Tema 2: Introducción al Desarrollo en Android

En este tema veremos …

- ¿Por qué Android?
- Orígenes de Android
- Arquitectura de Android
- Versiones
- Herramientas de Desarrollo: Android Studio
- Actividades y pila de actividades

¿Por qué Android?

- Plataforma de software libre de Google
- Líder del Mercado de los SO



http://infographic.statista.com/normal/chartoftheday_8787_android_es_ya_el_sistema_operativo_mas_usado_del_mundo_n.jpg

… pero existen otras

Android combina …

- Código Abierto
  - Software Libre (Linux, SQLite, Webkit, …)
  - Plataforma de Desarrollo Libre
  - Código abierto

- Portable (Aplicaciones en Java)
- Componentes reutilizables
- Servicios (SQL, sensores, localización, …)
- Alto nivel de seguridad (permisos de las aplicaciones)
- Optimización para dispositivos móviles (Máquina Virtual Dalvik), con bajo consumo de recursos (memoria, batería, …)
- Alta calidad multimedia (gráficos, vídeo, sonido).

Orígenes de Android

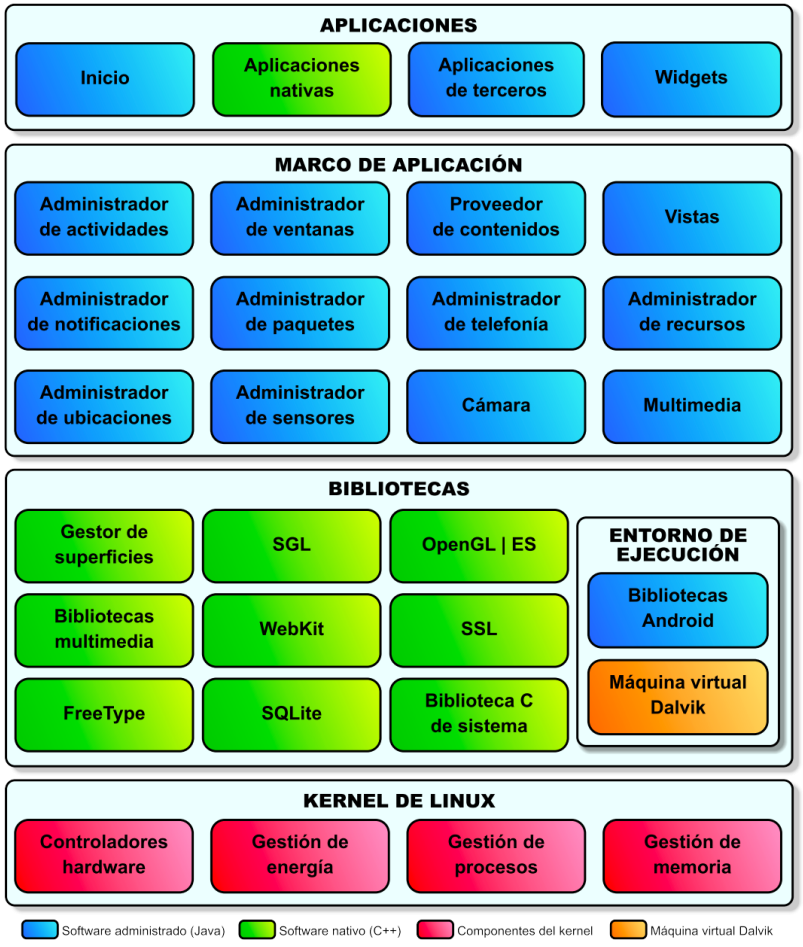
2005	• Google compra Android Inc. • Se crea la Dalvik Virtual Machine
2007	• Nace la corporación OHA (Open Handset Alliance) para promover estándares abiertos para móviles. • 34 compañías (actualmente 84): Google, Texas Instruments, HTC, Dell, Intel, Motorola, Texas Instruments, Samsung, LG, T-Mobile, Nvidia, Ericson, Vodafone, Toshiba, … • Lanzan la primera versión del Android SDK
2008	• Primer móvil Android: HTC Dream (T-Mobile G1) • Android Market • Google lidera el código fuente Android
2009	• Versiones 1.5 y 2.0
2010	• Android supera a iPhone en EEUU
2011	• Versión 3.0 del SDK pata Tablets y 4.0
2012 - 2017	• Versiones de la 4.1 a la actual

Arquitectura de Android

- Es importante conocer el funcionamiento interno de Android para entender sus capacidades y limitaciones

- La **Aquitectura de Android** está formada por 4 capas, todas basadas en software libre.

Arquitectura de Android

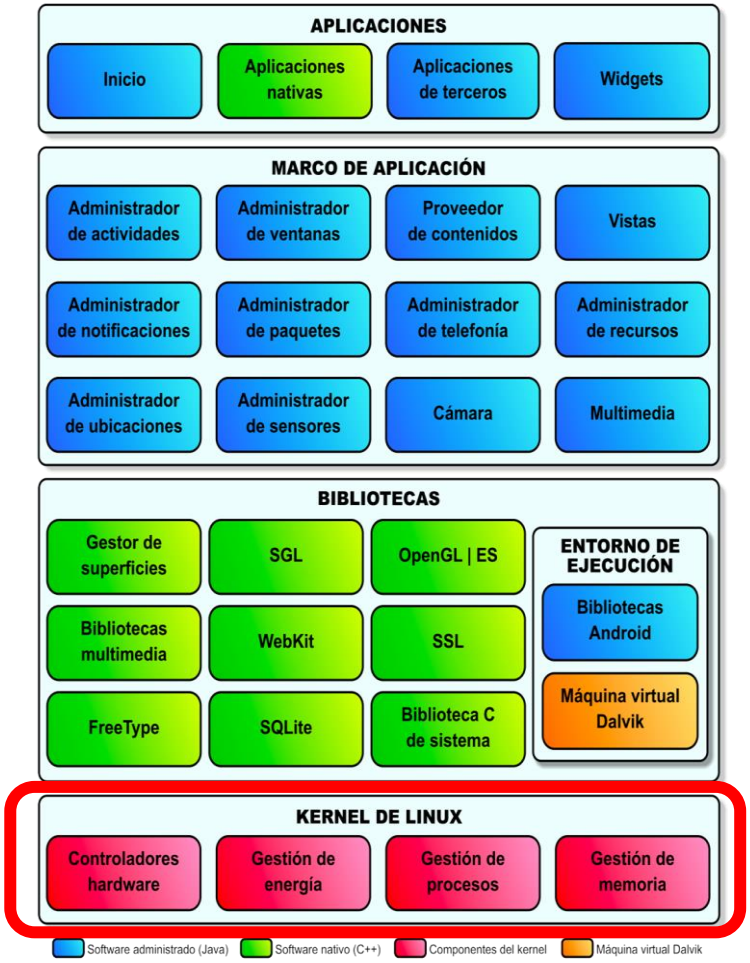


Arquitectura: Kernel de Linux

Android utiliza el núcleo de Linux como una **capa de abstracción para el hardware** disponible en los dispositivos móviles.

Esta capa contiene los drivers necesarios para que cualquier componente hardware pueda ser utilizado mediante las llamadas correspondientes.

Es la única capa que depende del fabricante. Siempre que un fabricante incluye un nuevo elemento de hardware, debe crear las librerias de control o drivers necesarios dentro de este kernel de Linux.



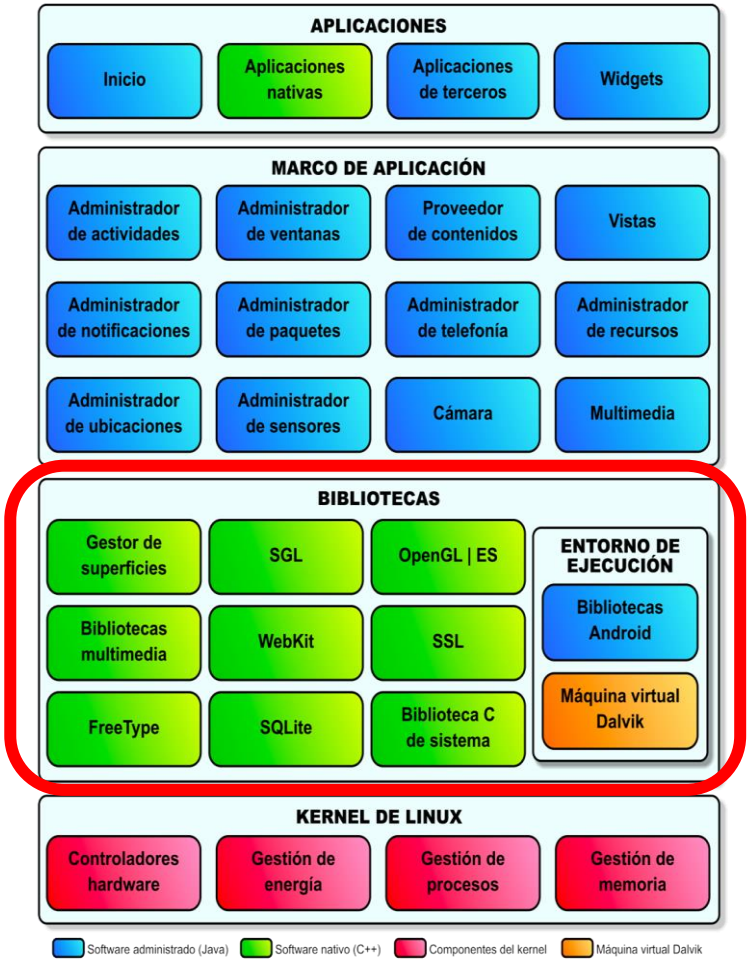
Arquitectura: Runtime

Lo constituyen:

- **Core Libraries**, que son librerias con mulititud de clases Java
- **Dalvik VM**, máquina virtual optimizada para dispositivos móviles (con poca capacidad de proceso).

Se encarga de ejecutar las aplicaciones (APK, Android Package), interpretando el código Java (ficheros .dex, Dalvik Executable) apoyándose en el Kernel del Linux y delegando en éste algunas funciones (manejo de memoria, threading, …)

Cada aplicación se ejecuta en su propio proceso, por lo que se instancia una Dalvik VM por cada aplicación.

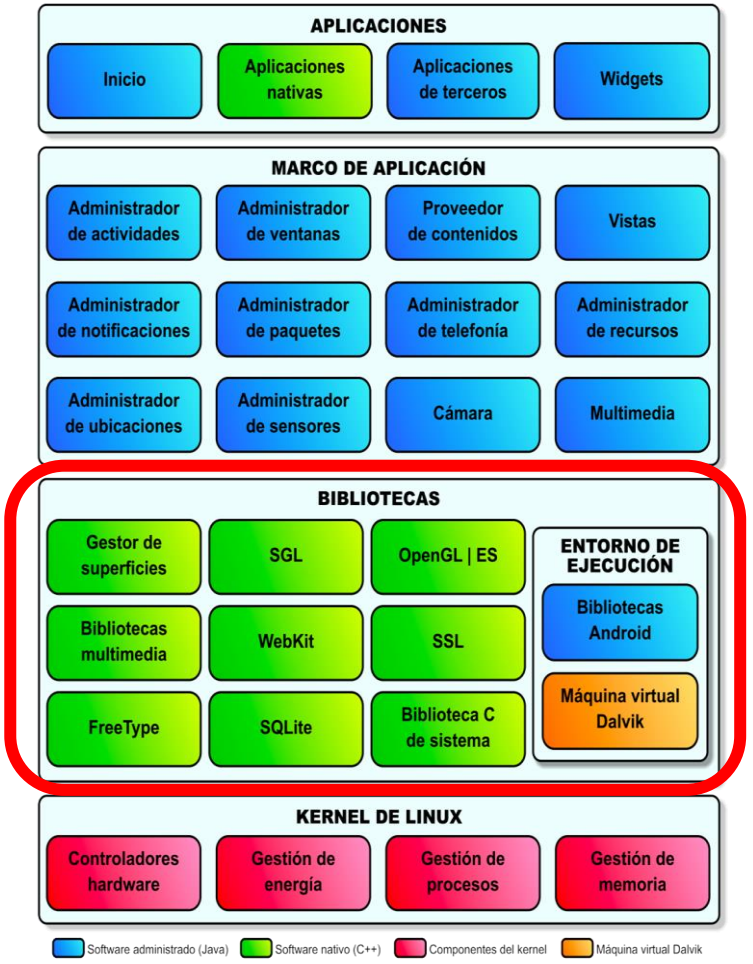


Arquitectura: Librerías

Librerías C/C++ nativas para el Kernel de Linux. La mayoría de las aplicaciones utilizan la Dalvik VM para acceder a ellas.

libc: Incluye todas las cabeceras y funciones según el estándar del lenguaje C. Todas las demás librerías se definen en este lenguaje.

Surface Manager: Es la encargada de componer los diferentes elementos de navegación de pantalla. Gestiona también las ventanas pertenecientes a las distintas aplicaciones activas en cada momento.



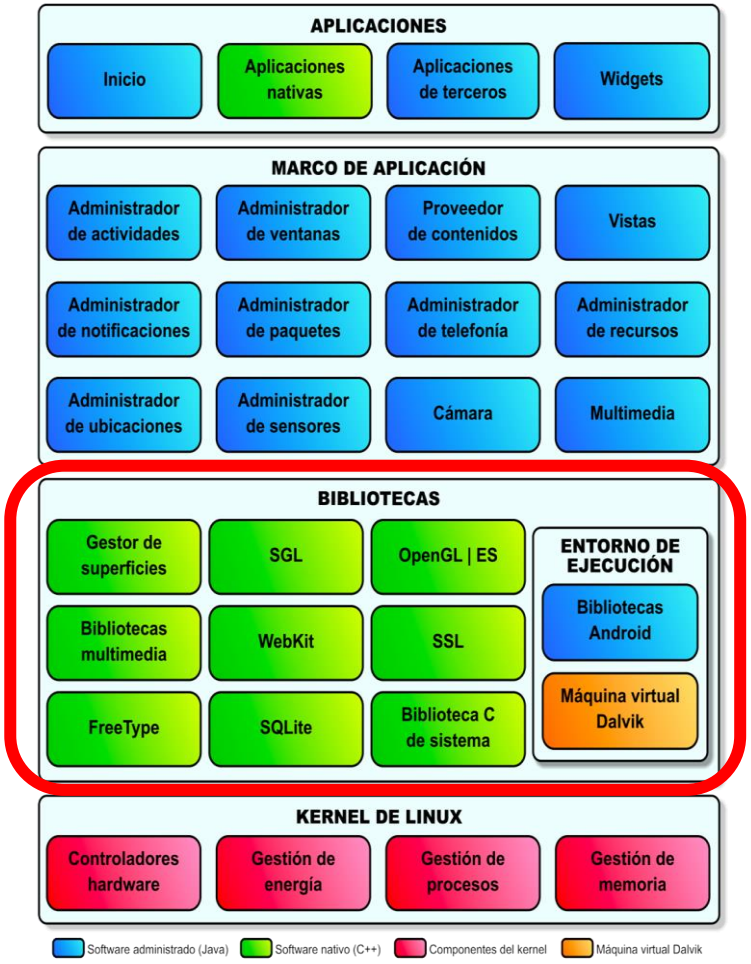
Arquitectura: Librerías

OpenGL/SL y SGL: Representan las librerías gráficas y, por tanto, sustentan la capacidad gráfica de Android.

- OpenGL/SL maneja gráficos en 3D y permite utilizar, en caso de que esté disponible en el propio dispositivo móvil, el hardware encargado de proporcionar gráficos 3D.

- SGL proporciona gráficos en 2D, por lo que será la librería más habitualmente utilizada por la mayoría de las aplicaciones.

Media Libraries: Proporciona todos los códecs necesarios para el contenido multimedia soportado en Android (vídeo, audio, imágenes estáticas y animadas, etc.)



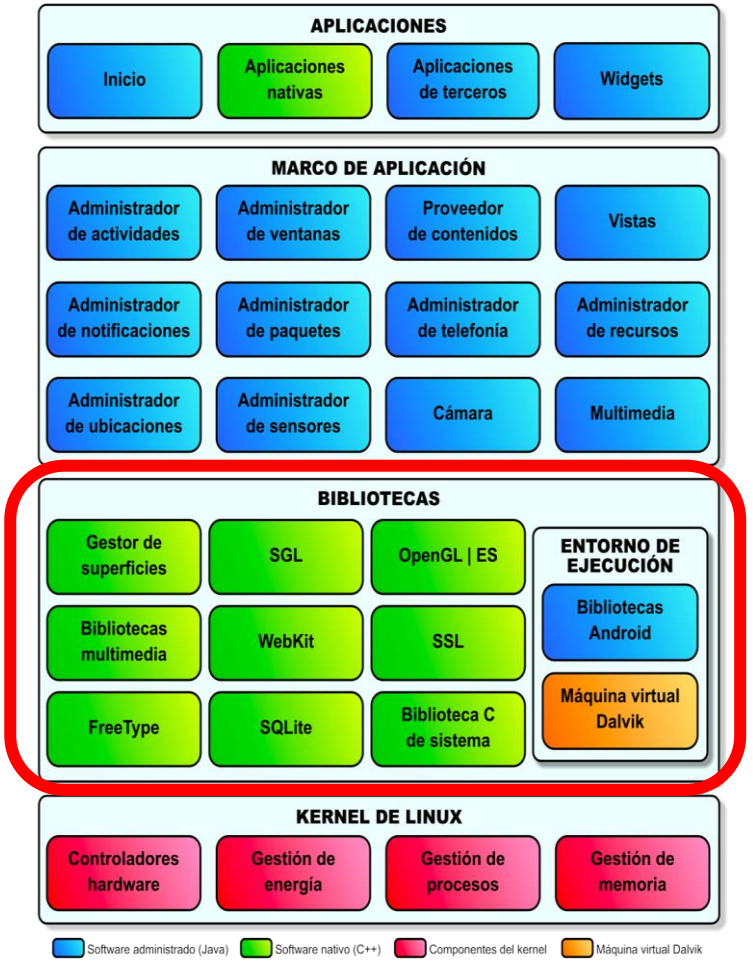
Arquitectura: Librerías

FreeType: Permite trabajar de forma rápida y sencilla con distintos tipos de fuentes.

SSL: Posibilita la utilización de dicho protocolo para establecer comunicaciones seguras.

SQLite: Creación y gestión de bases de datos relacionales.

WebKit: Proporciona un motor para las aplicaciones de tipo navegador y forma el núcleo del actual navegador incluido por defecto en la plataforma Android.



Arquitectura: Framework de Aplicaciones

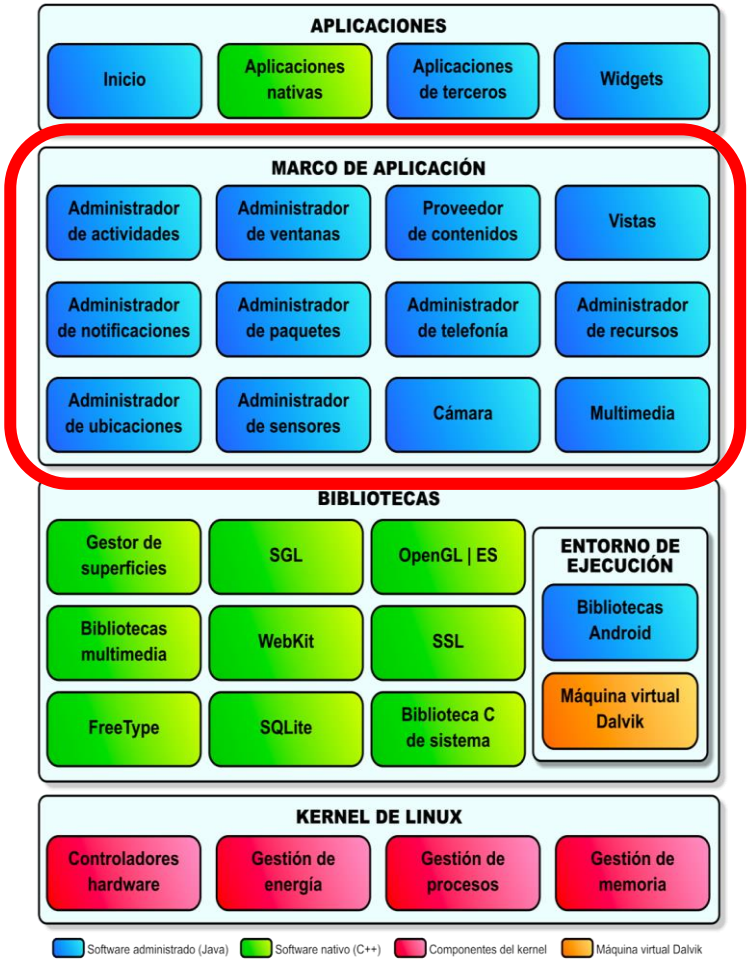
Conjunto de herramientas de desarrollo de cualquier aplicación (API).

Activity Manager: Gestiona el ciclo de vida de las aplicaciones.

Window Manager: Gestiona las ventanas de las aplicaciones y utiliza la librería Surface Manager.

Telephone Manager: Funcionalidades propias del teléfono (llamadas, mensajes, etc.).

Content Provider: Permite compartir datos entre aplicaciones (por ejemplo, información de contactos, agenda, mensajes, ...)



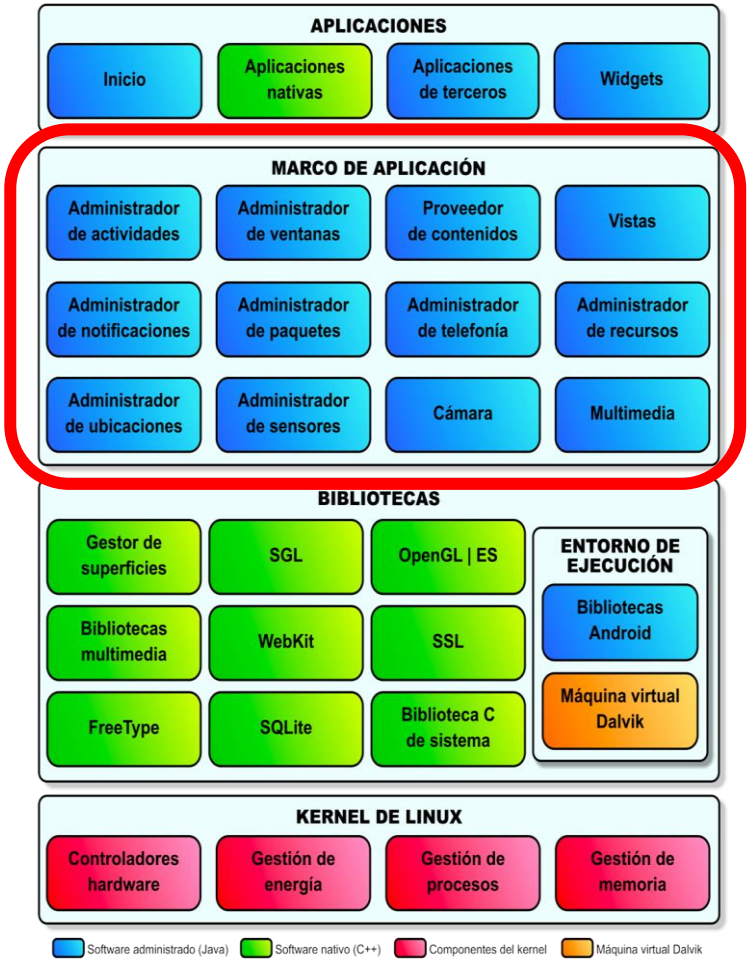
Arquitectura: Framework de Aplicaciones

View System: Construcción de interfaces de usuario (GUI), como listas, mosaicos, botones, …

Location Manager: Posibilita la obtención de información de localización y posicionamiento.

Notification Manager: Comunicación al usuario eventos que ocurran durante su ejecución: una llamada entrante, un mensaje recibido, …

XMPP Service: Colección de API para utilizar este protocolo de intercambio de mensajes basado en XML.



Arquitectura: Aplicaciones

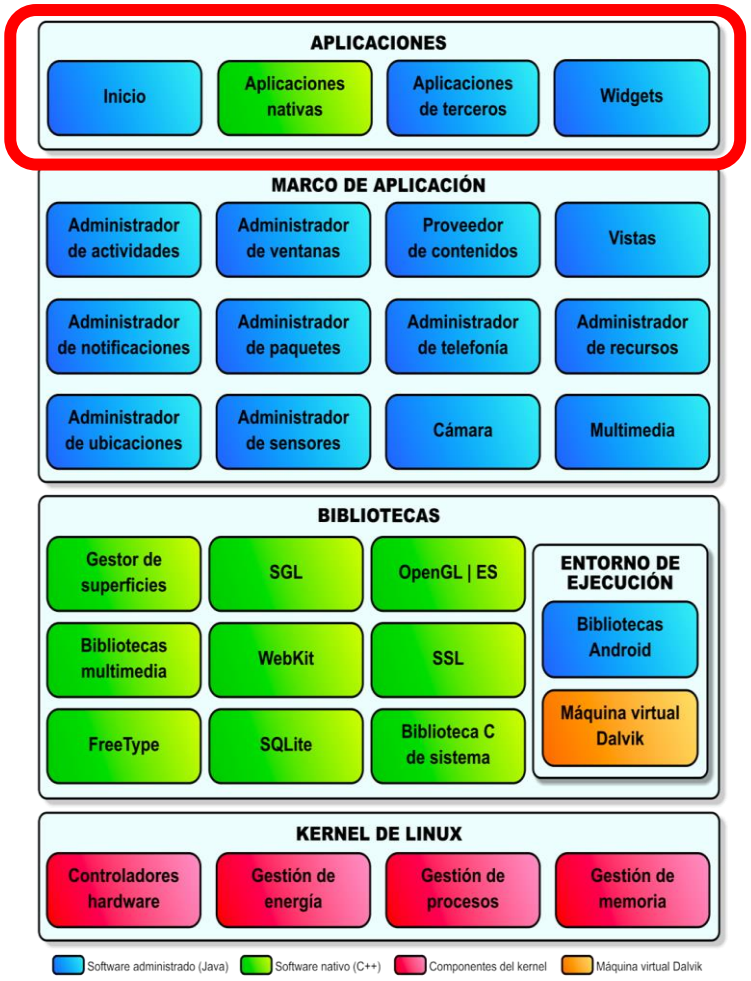
Este nivel contiene las aplicaciones de usuario, tanto las incluidas por defecto de Android como aquellas que el usuario vaya añadiendo posteriormente, ya sean de terceras empresas o de su propio desarrollo.

Todas utilizan los servicios, las API y librerías de los niveles anteriores.

Podemos tener dos tipos:

- Aplicaciones que se ejecutan en la Dalvik VM: escritas en Java usando SDK (Software Development Kit).

- Aplicaciones en código nativo: escritas en C/C++ usando Android NDK (Native Development Kit)



Versiones de Android

- Hasta la fecha (3 septiembre 2019) se han distribuido **29 ”versiones“** de Android, la mayoría aún en uso.
- Todas las versiones son compatibles hacia atrás.
- Existe gran nivel de **Fragmentación** a nivel de plataforma: Existen muchos usuarios con diferentes sistemas.
- Ello tiene gran repercusión en el desarrollo de aplicaciones.
- Condicionarán los criterios a tener en cuenta al elegir la versión mínima.

Identificación de las versiones

- **Nombre del código**: Nombres de dulces en orden alfabético, cada uno tiene un icono de dicho dulce
- **Número Versión**: Identifica cada versión lanzada dentro de cada código
- **Fecha de lanzamiento**: del código
- **Nivel de API**: Número entero correlativo que se incrementa sólo cuando se añaden nuevas funciones al API de desarrollo.

Nombre código	Número de versión	Fecha de lanzamiento	Nivel de API
Apple Pie[1]	1.0	23 de septiembre de 2008	1
Banana Bread[1]	1.1	9 de febrero de 2009	2
Cupcake	1.5	25 de abril de 2009	3
Donut	1.6	15 de septiembre de 2009	4
Eclair	2.0 – 2.1	26 de octubre de 2009	5 – 7
Froyo	2.2 – 2.2.3	20 de mayo de 2010	8
Gingerbread	2.3 – 2.3.7	6 de diciembre de 2010	9 – 10
Honeycomb[2]	3.0 – 3.2.6	22 de febrero de 2011	11 – 13
Ice Cream Sandwich	4.0 – 4.0.5	18 de octubre de 2011	14 – 15
Jelly Bean	4.1 – 4.3.1	9 de julio de 2012	16 – 18
KitKat	4.4 – 4.4.4	31 de octubre de 2013	19 – 20
Lollipop	5.0 – 5.1.1	12 de noviembre de 2014	21 – 22
Marshmallow	6.0 – 6.0.1	5 de octubre de 2015	23
Nougat	7.0 – 7.1.2	15 de junio de 2016	24 – 25
Oreo	8.0 – 8.1	21 de agosto de 2017	26 – 27
Pie	9.0	6 de agosto de 2018	28
Android 10 [3]	10.0	3 de septiembre del 2019	29

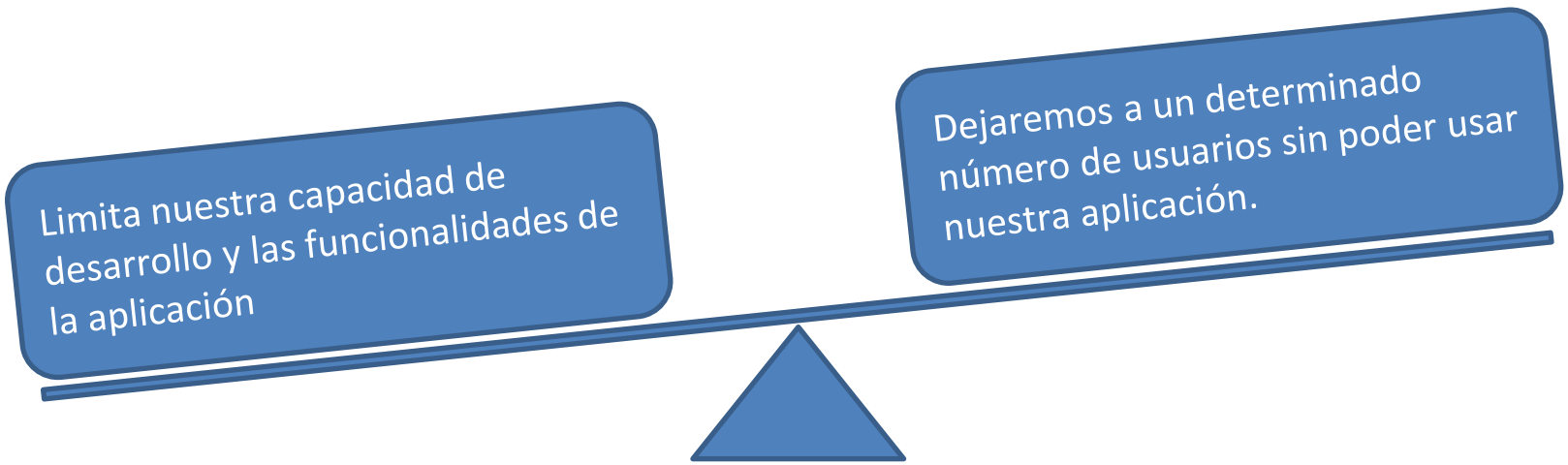
¿Qué versión mínima elijo para mi aplicación?

- Cuando creemos un proyecto Android tendremos que elegir la versión mínima de la aplicación (**mínimo nivel de API**).

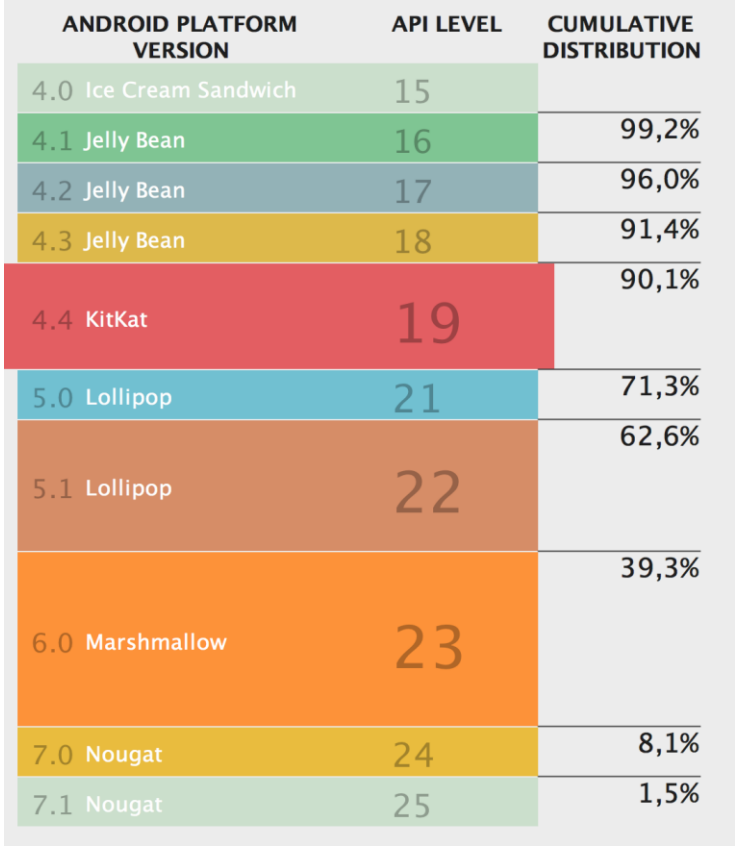
- Este aspecto establece restricciones muy importantes:

No se podrán utilizar funcionalidades introducidas en versiones posteriores de la API.

No se podrá instalar la aplicación en dispositivos con una versión anterior.



¿Qué versión mínima elijo para mi aplicación?



ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.0 Ice Cream Sandwich	15	
4.1 Jelly Bean	16	99,2%
4.2 Jelly Bean	17	96,0%
4.3 Jelly Bean	18	91,4%
4.4 KitKat	19	90,1%
5.0 Lollipop	21	71,3%
5.1 Lollipop	22	62,6%
6.0 Marshmallow	23	39,3%
7.0 Nougat	24	8,1%
7.1 Nougat	25	1,5%

Ejemplo: Si seleccionamos
API 16 - Android 4.1 (Jelly Bean):

- 0,8% no podrán usar nuestra app.
- 99,2% podrán usar nuestra app.
- No podremos usar funcionalidades de la API 17 en adelante.
- A mayor API, mayores funcionalidades pero menos dispositivos compatibles

Más sobre versiones:
http://developer.android.com/intl/es/about/dashboards/

¿Qué versión mínima elijo para mi aplicación?

Estrategia recomendada

- Partir de la **versión más baja posible** y ampliarla sólo si necesitamos alguna funcionalidad o capacidad realmente imprescindible introducida en una versión posterior de la API.

- Si la versión mínima de la API deja fuera demasiados usuarios (clientes), podemos **compilar varias versiones** de nuestra aplicación para diferentes versiones de la plataforma y publicarlas por separado (Google Play gestiona este aspecto de manera auntomática)

Descripción de las Versiones

https://es.wikipedia.org/wiki/Anexo:Historial_de_versiones_de_Android

HERRAMIENTAS DE DESARROLLO

En este apartado hablaremos principalmente de:

- JDK
- SDK
- Android Studio
- AVD

Y veremos las partes principales de un programa en Android Studio

¿Qué herramientas necesitamos para desarrollar en Android?

- Java (JDK)
- Android SDK
- Emulador AVD (Android Virtual Device)
- IDE
  - Android Studio (official)
  - Antiguos/no soportados
    - Eclipse + Plug-in ADT (Android Developer Tools)
    - Otro (NetBeans, ...)

Android Studio

Incluye todas las herramientas necesarias para el desarrollo y depuración de proyectos Android (IDE + Android SDK + AVD)

Instalación:

- Instalar JDK (mínima versión JDJ7)
- Descargar e instalar el paquete de instalación de Android Studio: http://developer.android.com/intl/es/develop/index.html
- En algunas versiones de Windows es necesario especificar dónde se encuentra el JDK, haciendo que la variable de entorno JAVA_HOME apunte a la carpeta correcta (por ejemplo: *C:\Program Files\Java\jdk1.7.0_2*)
- Se recomienda un equipo con al menos 4 GB de memoria RAM
- También debemos comprobar que nuestro equipo cumple con los requisitos mínimos
- Opcionalmente podremos activar el Intel Visualization Technology (para el uso del emulador AVD)

Android Studio

http://developer.android.com/intl/es/develop/index.html

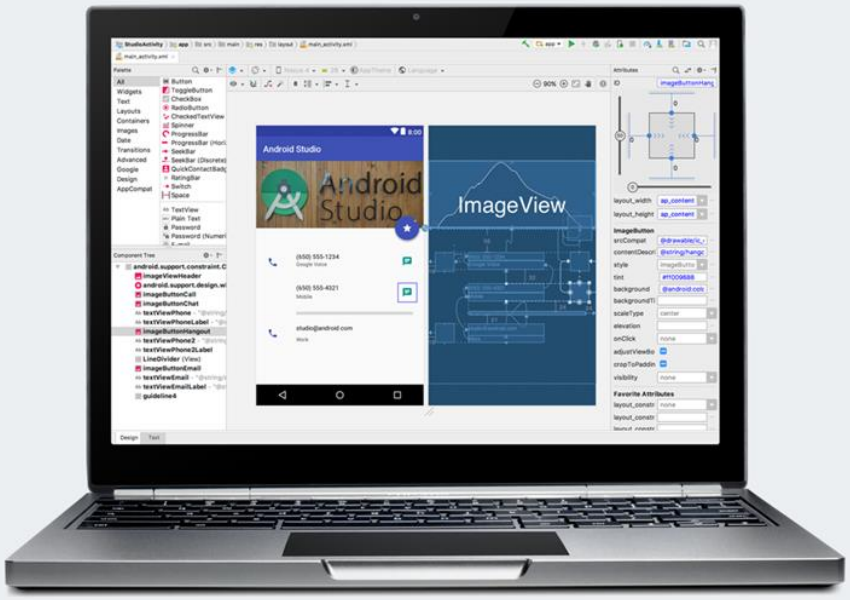
Android Studio

IDE oficial para Android

Android Studio proporciona las herramientas más rápidas para crear apps en todas las clases de dispositivos Android.

La edición de códigos de primer nivel, la depuración, las herramientas de rendimiento, un sistema de compilación flexible y un sistema instantáneo de compilación e implementación te permiten concentrarte en la creación de aplicaciones únicas y de alta calidad.

DESCARGAR ANDROID STUDIO
3.0.1 FOR MAC (738 MB)



Android Studio: Configuración Básica del SDK

- Configuración Básica del SDK
  - Acceder al SDK Manager (en opción Configuración)
  - Instalar por completo SDK Tools
  - En extras se debe tener instalado el Intel x86 Emulator Acelerator (HAXM installer) *Ver documentos extra en Moodle*
  - Instalar los paquetes convenientes para las diferentes versiones

Android Studio: Configuración de la AVD

- Instalar el Intel x86 Emulator Acelerator (HAXM installer)
- Activar la opción de **Visualization Technology** en el Setup del equipo (normalmente en las opciones avanzadas).
- Crear un emulador Emulador (**AVD Manager**)

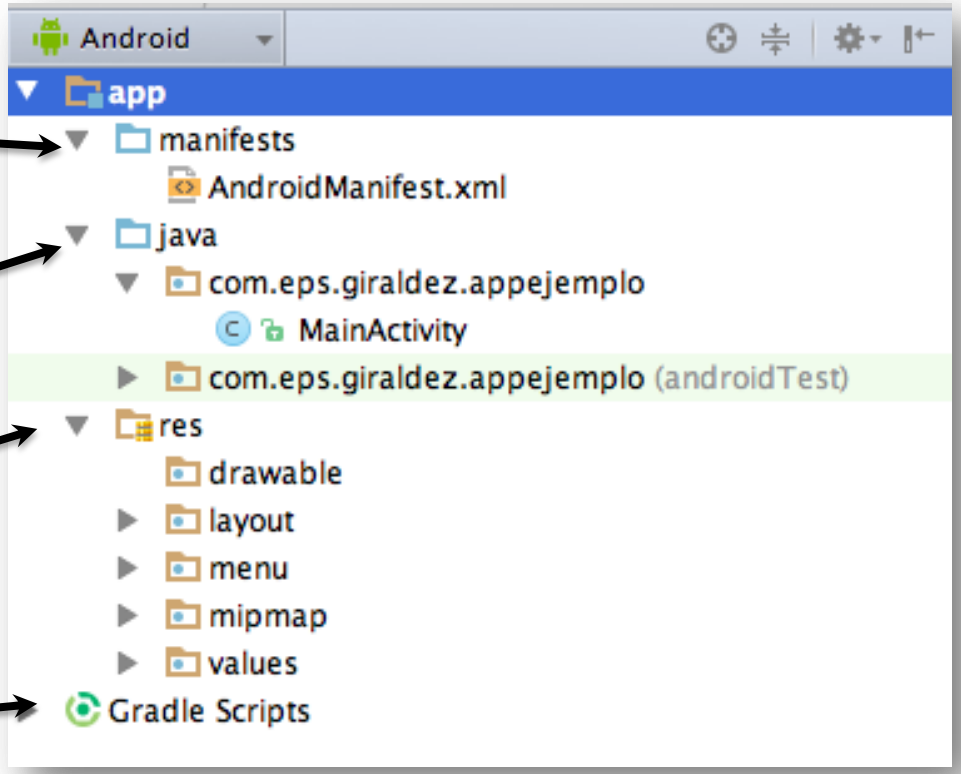
Elementos de una Proyecto Android

Descriptor de la aplicación

Código Fuente (ficheros .java)

Recursos

Archivos de construcción



AndroidManifest.xml

- **Archivo XML** que contiene nodos descriptivos sobre las características de la aplicación Android (por ejemplo, los *building blocks* existentes, la versión de SDK usada, los permisos necesarios para ejecutar algunos servicios, etc.)

- Todas las aplicaciones deben contener este archivo por convención.

- El nombre debe permanecer intacto.

- El nodo raíz de este documento se representa con la etiqueta ***<manifest>*** y por obligación debe contener un hijo de tipo ***<application>***.

AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:dist="http://schemas.android.com/apk/distribution"
    package="com.dam.chari.helloactivity">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="HelloActivity"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <dist:module dist:instant="true" />

</manifest>
```

AndroidManifest.xml

Manifest posee dos atributos:

- **xmlsn:android**: no debemos cambiarlo nunca, ya que es el *namespace* del archivo.
- **package**: indica el nombre del paquete Java que soporta a nuestra aplicación. El nombre del paquete debe ser único y un diferenciador a largo plazo.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:dist="http://schemas.android.com/apk/distribution"
    package="com.dam.chari.helloactivity">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="HelloActivity"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <dist:module dist:instant="true" />

</manifest>
```

La etiqueta **<application>** indica como estará construida nuestra aplicación. Dentro de ella definiremos nodos referentes a las **actividades** que contiene, las **librerías incluidas**, los **Intents**, **Providers**, y demás componentes.

AndroidManifest.xml

Algunas etiquetas:

allowBackup: Este atributo puede tomar los valores de true o false. Indica si la aplicación será persistente al cerrar nuestro AVD.

icon: Indica donde está ubicado el icono que se usará en la aplicación.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:dist="http://schemas.android.com/apk/distribution"
    package="com.dam.chari.helloactivity">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="HelloActivity"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <dist:module dist:instant="true" />

</manifest>
```

label: Es el nombre de la aplicación que verá el usuario en su teléfono. Normalmente apunta a la cadena “app_name” que se encuentra en el recurso *strings.xml*.

theme: Este atributo apunta al archivo de recursos *styles.xml*, donde se define la personalización del estilo visual de nuestra aplicación.

AndroidManifest.xml

Dentro de **<application>** encontraremos descrita la actividad principal.

Usaremos **<activity>** para representar un nodo tipo actividad. Tiene dos atributos:

- **name**: el cual se refiere a la clase Java que hace referencia a esta actividad (comienza con un punto “.”).

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:dist="http://schemas.android.com/apk/distribution"
    package="com.dam.chari.helloactivity">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="HelloActivity"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <dist:module dist:instant="true" />

</manifest>
```

- **label**: hace referencia al texto que se mostrará en la cabecera de la actividad. Por defecto no aparece hasta que no lo definimos

En próximos temas veremos el propósito de los nodos **<intent-filter>**, **<action>**, **<category>**, entre otros.

Recursos: carpeta *res*

- La carpeta ***res*** contiene los recursos usados por la aplicación y por el interfaz de usuario.

- El sistema reconoce el significado de cada fichero de recursos, que se debe almacenar en una subcarpeta específica, y puede cambiar su contenido.

- Cada recurso será identificado a través de un identificador.

Recursos: carpeta *res*

- **drawable**: ficheros de imágenes y descriptores de imágenes (jpg, png, xml …)

- **mipmap**: ficheros de iconos de la aplicación, independientes de los otros ficheros de imágenes.

Para diferentes densidades y resoluciones de pantalla, podemos tener carpetas ***drawable*** e ***mipmap*** independientes con sufijos: *-mdpi, -hdpi, -xhdpi, -xxhdpi*. Por ejemplo: *mipmap-xhdpi.*

Recursos: carpeta *res*

- **layout**: ficheros XML con los diseños de las actividades. Por ejemplo, para la actividad principal tenemos:
  - ***activity_main.xml***: Define el estilo de la pantalla, o actividad.
  - ***content_main.xml***: Determina los contenidos dentro de activity_main.xml, por ejemplo, texto, botones, etc. Android Studio crea por defecto este layout de tipo ***RelativeLayout***, que permite crear un grupo de componentes con ubicaciones relativas.
- **menu**: ficheros XML con los menús de la aplicación

Recursos: carpeta *res*

- **values**: contiene los ficheros que definen las cadenas, colores y estilos de la aplicación. La carpeta *values* puede tener variantes (por ejemplo, values-w820dp) según la adaptación de los ficheros.
  - ***string.xml***: Este fichero almacena todas las cadenas que se muestran en los *widgets* (controles, formas, botones, vistas, etc.) de nuestras actividades. Una de sus grandes utilidades es facilitar el uso de múltiples idiomas, ya que podemos externalizar las cadenas del código java y seleccionar la versión del archivo strings.xml con el lenguaje necesitado.

```
<resources>
    <string name="app_name">AppEjemplo</string>
    <string name="action_settings">Settings</string>
    <string name="mi_cadena">MI APLICACIÓN</string>
</resources>
```

  - ***colors.xml***: Determina los colores de la aplicación.
  - **styles.xml**: Determina los estilos de los diferentes componentes de la aplicación.
  - **dimens.xml**: Determina las dimensiones (márgenes) por defecto de nuestra aplicación.

Recursos: carpeta *res*

- Otras carpetas (puede cambiar según la versión de Android Studio):
  - **anim**: ficheros XML con descriptores de animación
  - **xml**: otros ficheros XML no incluidos en carpetas anteriores.
  - **raw**: ficheros que o se encuentran en formato XML y que podrán ser accedidos por su ID de recurso.
  - **doc**: ficheros de documentación asiciada a proyecto.

Carpeta java

- La carpeta **java** aloja todos los archivos relacionados con nuestras **actividades** y otros archivos fuente auxiliares, los cuales implementan la lógica de nuestra aplicación.

- Por ejemplo, al abrir nuestro archivo **MainActivity.java** veremos toda la lógica necesaria para que la actividad interactúe de manera correcta con el usuario.

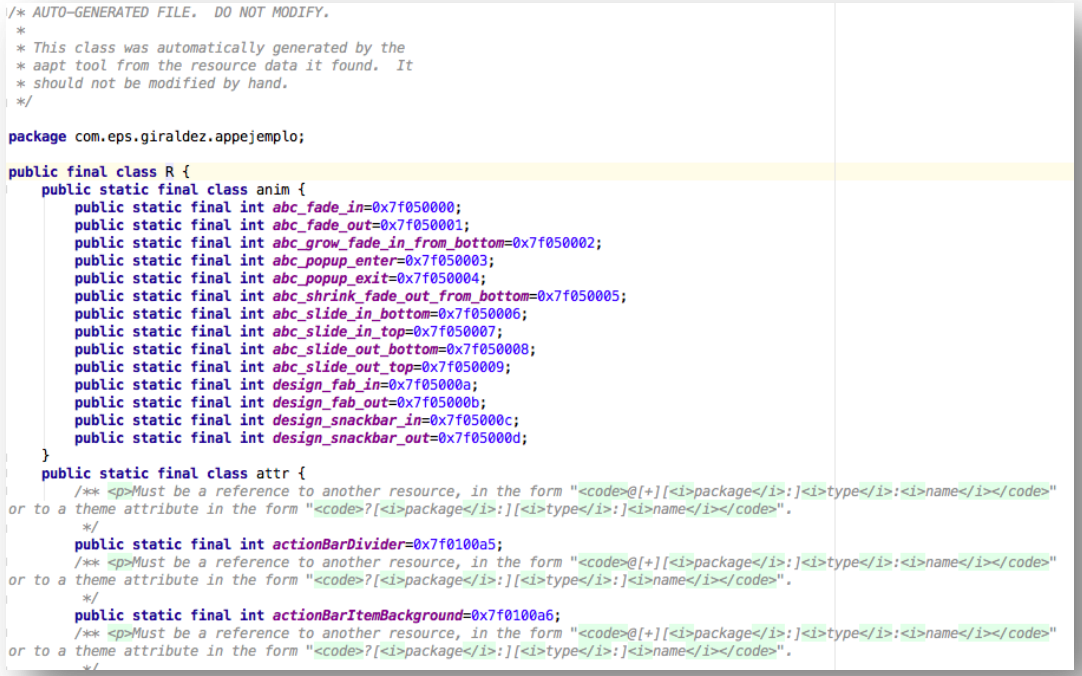
- Más adelante veremos que una actividad tiene **un ciclo de vida** y lo único que esta bajo nuestro control es la manipulación de cada estado.

```java
package com.eps.giraldez.appejemplo;

import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener((view) -> {
                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
        });
    }

    @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);
    }
}
```

Clase R.java

- El archivo **R.java** es un archivo que se genera automáticamente dentro de la carpeta build, para enlazar todos los recursos que tenemos en nuestro proyecto al código Java.

- Contiene clases anidadas que representan todos los recursos de nuestro proyecto.
- Cada atributo tiene una dirección de memoria asociada referenciada a un recurso en específico.
- **NUNCA** se debe modificar manualmente el fichero R.java.



```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package com.eps.giraldez.appejemplo;

public final class R {
    public static final class anim {
        public static final int abc_fade_in=0x7f050000;
        public static final int abc_fade_out=0x7f050001;
        public static final int abc_grow_fade_in_from_bottom=0x7f050002;
        public static final int abc_popup_enter=0x7f050003;
        public static final int abc_popup_exit=0x7f050004;
        public static final int abc_shrink_fade_out_from_bottom=0x7f050005;
        public static final int abc_slide_in_bottom=0x7f050006;
        public static final int abc_slide_in_top=0x7f050007;
        public static final int abc_slide_out_bottom=0x7f050008;
        public static final int abc_slide_out_top=0x7f050009;
        public static final int design_fab_in=0x7f05000a;
        public static final int design_fab_out=0x7f05000b;
        public static final int design_snackbar_in=0x7f05000c;
        public static final int design_snackbar_out=0x7f05000d;
    }
    public static final class attr {
        /** <p>Must be a reference to another resource, in the form "<code>@[+][<i>package</i>:]<i>type</i>:<i>name</i></code>"
or to a theme attribute in the form "<code>?[<i>package</i>:][<i>type</i>:]<i>name</i></code>".
         */
        public static final int actionBarDivider=0x7f0100a5;
        /** <p>Must be a reference to another resource, in the form "<code>@[+][<i>package</i>:]<i>type</i>:<i>name</i></code>"
or to a theme attribute in the form "<code>?[<i>package</i>:][<i>type</i>:]<i>name</i></code>".
         */
        public static final int actionBarItemBackground=0x7f0100a6;
        /** <p>Must be a reference to another resource, in the form "<code>@[+][<i>package</i>:]<i>type</i>:<i>name</i></code>"
or to a theme attribute in the form "<code>?[<i>package</i>:][<i>type</i>:]<i>name</i></code>".
         */
```

Clase R.java

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package com.eps.giraldez.appejemplo;

public final class R {
    public static final class anim {
        public static final int abc_fade_in=0x7f050000;
        public static final int abc_fade_out=0x7f050001;
        public static final int abc_grow_fade_in_from_bottom=0x7f050002;
        public static final int abc_popup_enter=0x7f050003;
        public static final int abc_popup_exit=0x7f050004;
        public static final int abc_shrink_fade_out_from_bottom=0x7f050005;
        public static final int abc_slide_in_bottom=0x7f050006;
        public static final int abc_slide_in_top=0x7f050007;
        public static final int abc_slide_out_bottom=0x7f050008;
        public static final int abc_slide_out_top=0x7f050009;
        public static final int design_fab_in=0x7f05000a;
        public static final int design_fab_out=0x7f05000b;
        public static final int design_snackbar_in=0x7f05000c;
        public static final int design_snackbar_out=0x7f05000d;
    }
    public static final class attr {
        /** <p>Must be a reference to another resource, in the form "<code>@[+][<i>package</i>:]<i>type</i>:<i>name</i></code>"
or to a theme attribute in the form "<code>?[<i>package</i>:][<i>type</i>:]<i>name</i></code>".
         */
        public static final int actionBarDivider=0x7f0100a5;
        /** <p>Must be a reference to another resource, in the form "<code>@[+][<i>package</i>:]<i>type</i>:<i>name</i></code>"
or to a theme attribute in the form "<code>?[<i>package</i>:][<i>type</i>:]<i>name</i></code>".
         */
        public static final int actionBarItemBackground=0x7f0100a6;
        /** <p>Must be a reference to another resource, in the form "<code>@[+][<i>package</i>:]<i>type</i>:<i>name</i></code>"
or to a theme attribute in the form "<code>?[<i>package</i>:][<i>type</i>:]<i>name</i></code>".
         */
```

ACTIVIDADES Y PILA DE ACTIVIDADES

Aplicaciones y Procesos

- Una aplicación se ejecuta en su propio proceso Linux.
- Este proceso se crea la primera vez que ejecutamos la aplicación y se va a mantener en memoria (en la **pila de aplicaciones**) hasta que el sistema necesite memoria para otras aplicaciones.
- Aunque cerremos la aplicación, ésta se mantiene “abierta” y en la pila de aplicaciones mientras su proceso está vivo.
- La destrucción de un proceso no es controlado directamente por la aplicación. En lugar de esto, es el sistema quien determina cuando destruir el proceso, basándose en el conocimiento que tiene el sistema de las partes de la aplicación que están corriendo (actividades y servicios): frecuencia de uso, tiempo desde la última, memoria que consume, memoria disponible …

Aplicaciones y Procesos

- Android es sensible al ciclo de vida de una aplicación.
- Por ejemplo:
  - Si tras eliminarse un proceso el usuario abre de nuevo la aplicación, se instanciará un nuevo proceso, pero el estado anterior se habrá perdido.
  - Igualmente, cuando cambiamos la orientación del terminal, se crea un nuevo proceso con unas características gráficas diferentes.
- Por tanto, para crear aplicaciones estables, es necesario comprender cómo funciona el **ciclo de vida de una aplicación** y saber los eventos relacionados con el mismo para gestionar correctamente los cambios de estado.

Actividades

- Una aplicación en Android está formada por un conjunto de elementos básicos de interacción con el usuario, conocidos como **actividades**.
- Una aplicación también puede contener servicios, aunque veremos más adelante.
- Las actividades controlan **el ciclo de vida** de las aplicaciones, ya que el usuario no cambia de aplicación, sino de actividad dentro de la aplicación.
- Conforme vamos navegando por la aplicación, pasando de actividad en actividad, el sistema va a mantener una **pila de las actividades** previamente visualizadas/visitadas. De esta forma, podemos regresar a la actividad anterior pulsando la tecla de retorno del terminal.

Pila de Actividades