

# ANDROID

ADOLFO SANZ DE DIEGO

JUNIO 2011 - ACTUALIZADO JULIO 2012

1 CRÉDITOS

# 1.1 AUTOR

- **Adolfo Sanz De Diego**
  - Correo: [asanzdiego@gmail.com](mailto:asanzdiego@gmail.com)
  - Twitter:  
[@asanzdiego](<http://twitter.com/asanzdiego>)
  - Linkedin: <http://www.linkedin.com/in/asanzdiego>
  - Blog: <http://asanzdiego.blogspot.com.es>

# 1.2 LICENCIA

- **Este obra está bajo una licencia:**
  - Creative Commons Reconocimiento-CompartirIgual 3.0
- **El código fuente de los programas están bajo una licencia:**
  - GPL 3.0

# 1.3 COFINANCIADO



*Europa impulsa  
nuestro crecimiento*



Logos

# 2 INTRODUCCIÓN

# 2.1 LOS INICIOS DE ANDROID

- 2005
  - Google adquiere **Android, Inc.**
    - Pequeña empresa que desarrolla software para móviles.
- 2007
  - Nace la **Open Handset Alliance**.
    - Consorcio de empresas (operadoras, fabricantes, software) unidas con el objetivo de desarrollar estándares abiertos para móviles.
    - Google, Intel, ARM, HTC, LG, Motorola, Samsung, T-Mobile, Vodafone, etc.
- 2008
  - Publicado Android como **Open Source**:

- Licencia Apache 2.0 + otras licencias (GPL v2 para el núcleo).
- Se abre el **Android Market**.
- HTC Dream (**G1**), primer teléfono con Android.

## 2.2 LA ESCALADA DE ANDROID

- 2009
  - Motorola Droid vende 1 millón de unidades en 74 días
    - superando el record del iPhone de Apple.
  - **16.000 aplicaciones** en el Android Market: (66% gratuitas, 33% de pago)
- 2010
  - Se activan **cada día 300.000 nuevos** dispositivos con Android.
  - Por primera vez la venta de Adroids supera a la de iPhones en EEUU.
- 2011
  - Se activan **cada día 700.000 nuevos** dispositivos con Android.

- 200.000 aplicaciones en el Android Market
- 2012
  - Se activan **cada día 1 millón de nuevos** dispositivos con Android.
  - **600.000 aplicaciones** en Google Play.

## 2.3 COMPARATIVA MUNDIAL

- 2011:
  - Android: 37%
  - iOS: 19%
  - Otros: 44%
- 2012:
  - Android: 59%
  - iOS: 23%
  - Otros: 18%

**Top Six Smartphone Operating Systems, Shipments, and Market Share, 2012 Q1 (Units in Millions)**

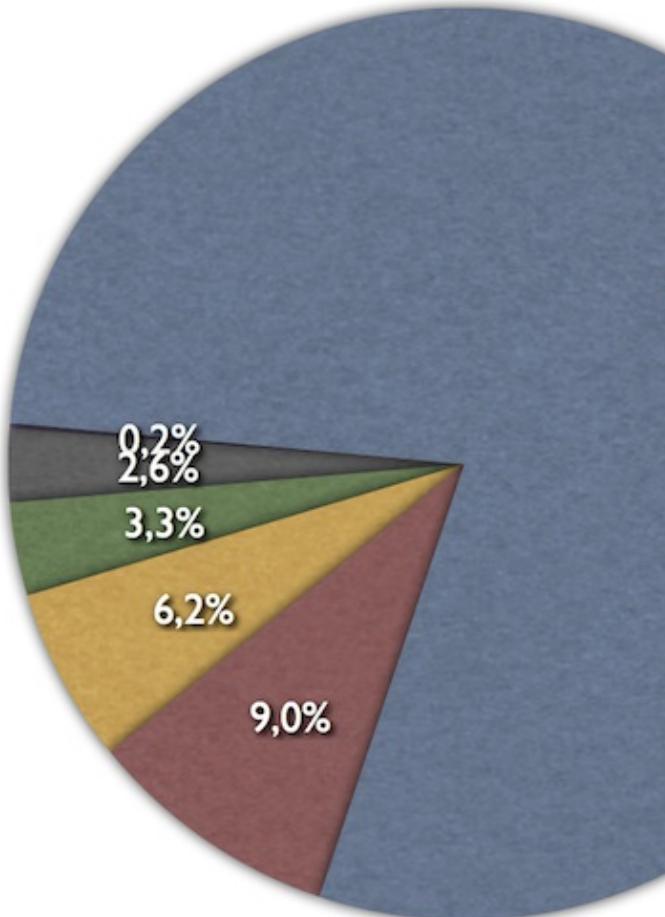
Mobile Operating System	1Q12 Unit Shipments	1Q12 Market Share	1Q11 Unit Shipments	1Q11 Market Share	Year-over-Year Change
Android	89.9	59.0%	36.7	36.1%	145.0%
iOS	35.1	23.0%	18.6	18.3%	88.7%
Symbian	10.4	6.8%	26.4	26.0%	-60.6%
BlackBerry OS	9.7	6.4%	13.8	13.6%	-29.7%
Linux	3.5	2.3%	3.2	3.1%	9.4%
Windows Phone 7/Windows Mobile	3.3	2.2%	2.6	2.6%	26.9%
Other	0.4	0.3%	0.3	0.3%	33.3%
Total	152.3	100.0%	101.6	100.0%	49.9%

**Source:** IDC Worldwide Mobile Phone Tracker, May 24, 2012

## Venta Mundial Smartphones

## 2.4 COMPARATIVA ESPAÑOLA (VENTA)

Ventas smartphones España (feb-may 2012)

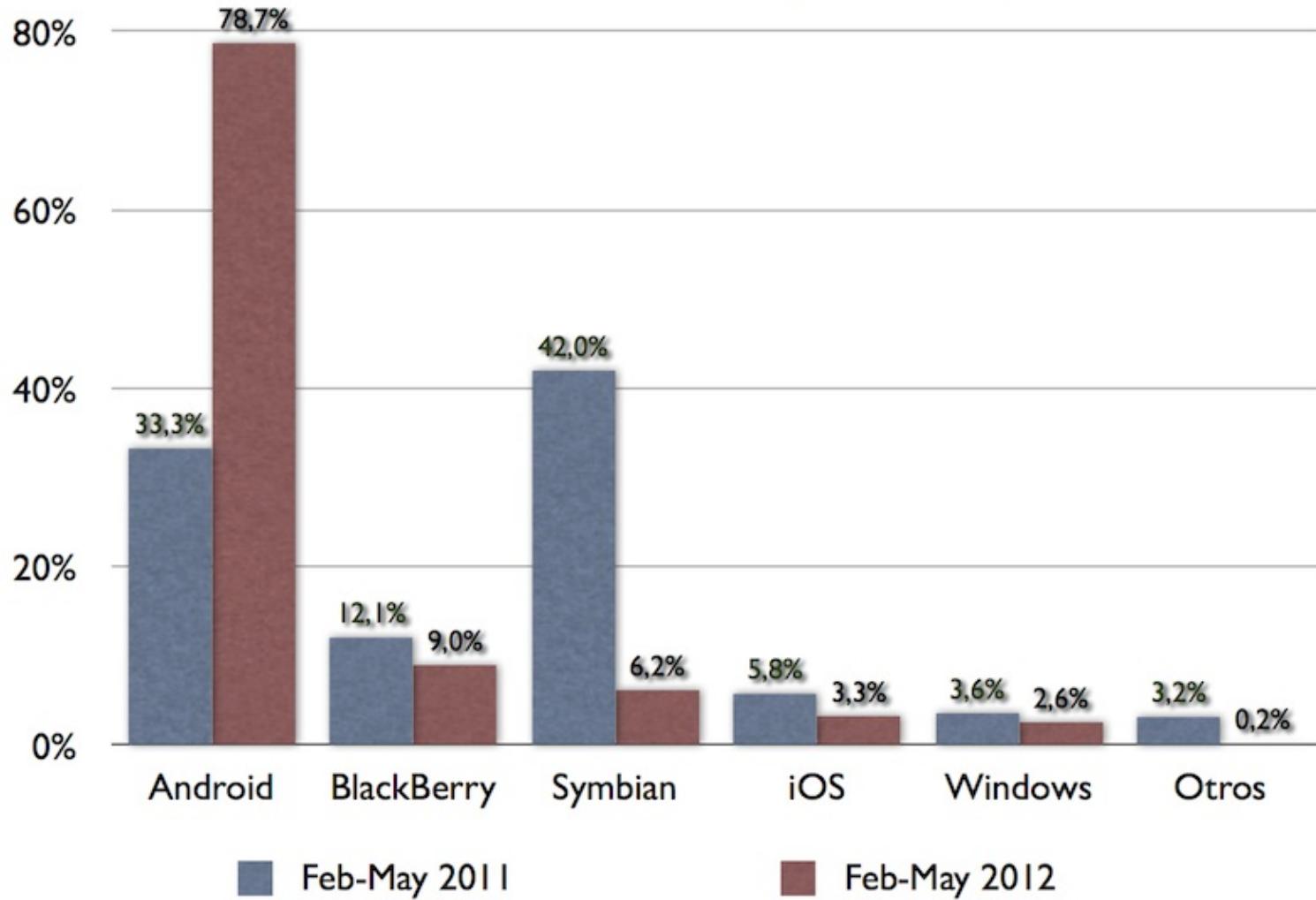


● Android   ● BlackBerry   ● Symbian   ● iOS   ● Windows   ● Otros

Venta Española Smartphones

## 2.5 COMPARATIVA ESPAÑOLA (EVOLUCIÓN)

## Evolución ventas smartphones España



Evolución Venta Española Smartphones

## 2.6 ¿QUÉ ES ANDROID?

- Es un **Sistema Operativo**, basado en el kernel de **linux**, diseñado para **móviles y tablets**.
- No es sólo eso, es además:
  - Conjunto de aplicaciones (agenda, contactos, navegador, etc.)
  - Conjunto de bibliotecas (OpenGL, Media, SQLite, etc.)
  - Framework (Activity, Intent, Service, Broadcast receivers, Content Provider, etc.)
  - Entorno de trabajo (compilador, emulador, debuguer, etc.)
- Leer  
<http://developer.android.com/guide/basics/what-is-android.html>

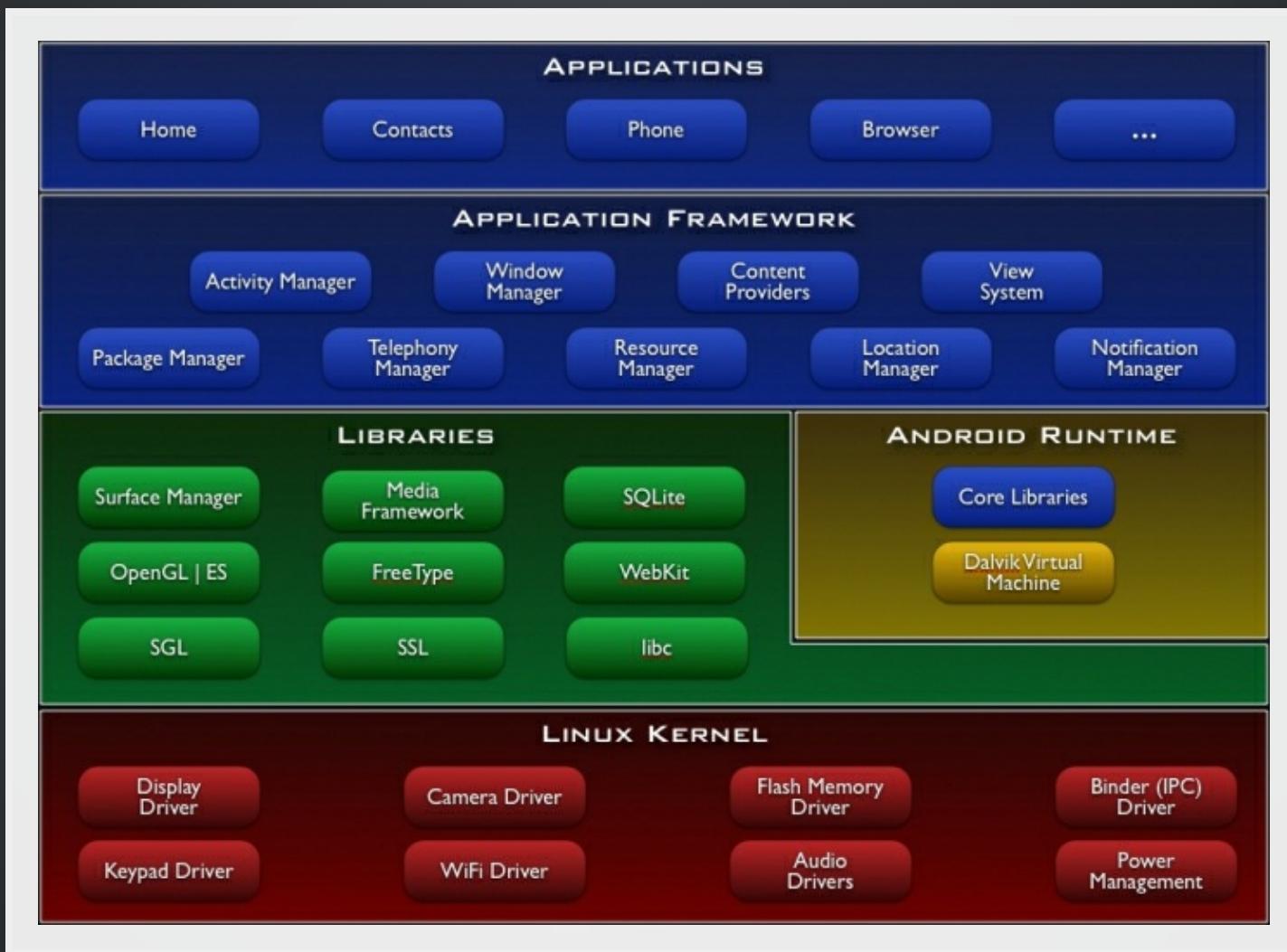
## 2.7 RECURSOS LIMITADOS

- Android está diseñado para ejecutarse en dispositivos limitados:
  - memoria, almacenamiento, CPU, pero sobre todo **banda ancha**
- Hay que esforzarse en desarrollar bien desde el primer día:
  - Usar herramientas como PMD, CPD, FindBugs para detectar errores y mejorar.
  - Leer
    - <http://developer.android.com/guide/practices/design.html>
- Hay que esforzarse en hacer aplicaciones usables para dispositivos:
  - Distintos tamaños de pantalla.
  - Distintas densidades de píxeles.
  - Distintos componentes hardware (GPS, brújula, cámara, bluetooth, etc. )
  - Leer
    - [http://developer.android.com/guide/practices/screens\\_support.html](http://developer.android.com/guide/practices/screens_support.html)

## 2.8 ¿CÓMO SE PROGRAMA ANDROID?

- De forma nativa en C/C++ (bajo nivel)
- En **Java**:
  - Android dispone de una máquina virtual **Dalvik** que puede ejecutar aplicaciones hechas en Java.

# 2.9 ARQUITECTURA



# Arquitectura - Esquema

# 2.10 ARQUITECTURA - KERNEL LINUX



## Arquitectura - Kernel Linux

- Android depende de **Linux** para los servicios base del sistema:
  - seguridad, gestión de memoria, gestión de procesos, controladores, etc.
- El núcleo también actúa como una capa de abstracción entre el **hardware** y el resto de la pila de software.

## 2.11 ARQUITECTURA - BIBLIOTECAS



### Arquitectura - Bibliotecas

- Android incluye un conjunto de **bibliotecas de utilidades** escritas en C/C++:
  - contenidos multimedia, gráficos 2D y 3D, base de datos SQLite, etc.

# 2.12 ARQUITECTURA - RUNTIME



Arquitectura - Runtime

- Android incluye un conjunto de **bibliotecas base** con:
  - el API casi completo de Java,
  - otras API muy usadas en entornos Java.
- Cada aplicación Android corre su propio proceso independiente, con su propia memoria, y con su propia instancia de la máquina virtual **Dalvik**:
  - Dalvik está optimizada para requerir poca memoria.
  - Dalvik ejecuta archivos en el formato Dalvik Executable (.dex).
  - Dalvik incluye una herramienta para transformar los .class de Java a .dex

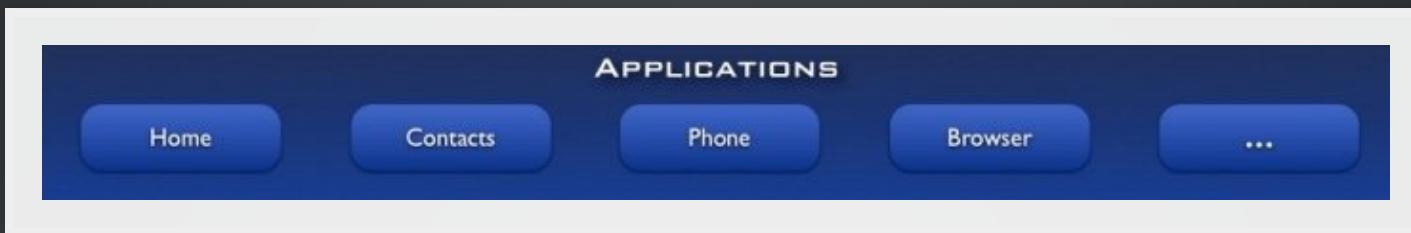
# 2.13 ARQUITECTURA - FRAMEWORK



Arquitectura - Framework

- Los desarrolladores tienen acceso a los mismos APIs usados por las aplicaciones base.
- El framework está diseñado 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).

# 2.14 ARQUITECTURA - APLICACIONES

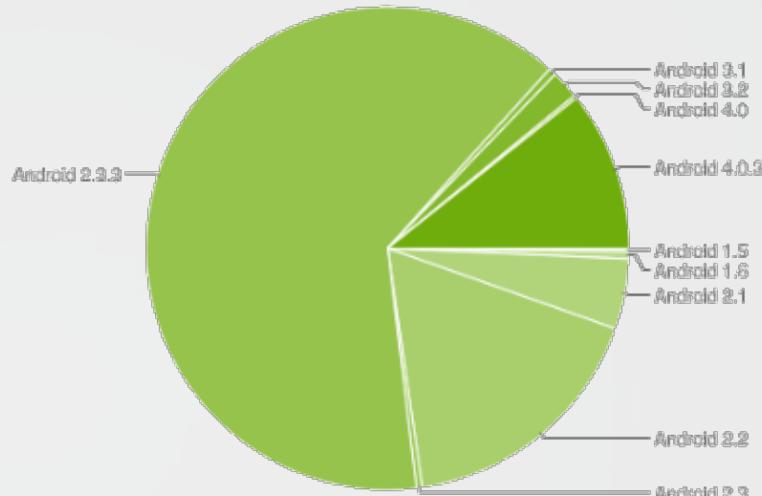


## Arquitectura - Aplicaciones

- Las **aplicaciones** base incluyen:
  - un cliente de correo electrónico, un programa de SMS, un calendario, mapas, navegador, contactos y otros.
- Estas aplicaciones están **disponibles** para el resto de aplicaciones.

# 2.15 VERSIONES DE ANDROID

Version	Codename	API Level	Distribution
1.5	Cupcake	3	0.2%
1.6	Donut	4	0.5%
2.1	Eclair	7	4.7%
2.2	Froyo	8	17.3%
2.3 - 2.3.2	Gingerbread	9	0.4%
2.3.3 - 2.3.7		10	63.6%
3.1	Honeycomb	12	0.5%
3.2		13	1.9%
4.0 - 4.0.2	Ice Cream Sandwich	14	0.2%
4.0.3 - 4.0.4		15	10.7%



*Data collected during a 14-day period ending on July 2, 2012*

# Versiones de Android

<http://developer.android.com/resources/dashboard/platform-versions.html>

# 2.16 JAVA & ANDROID

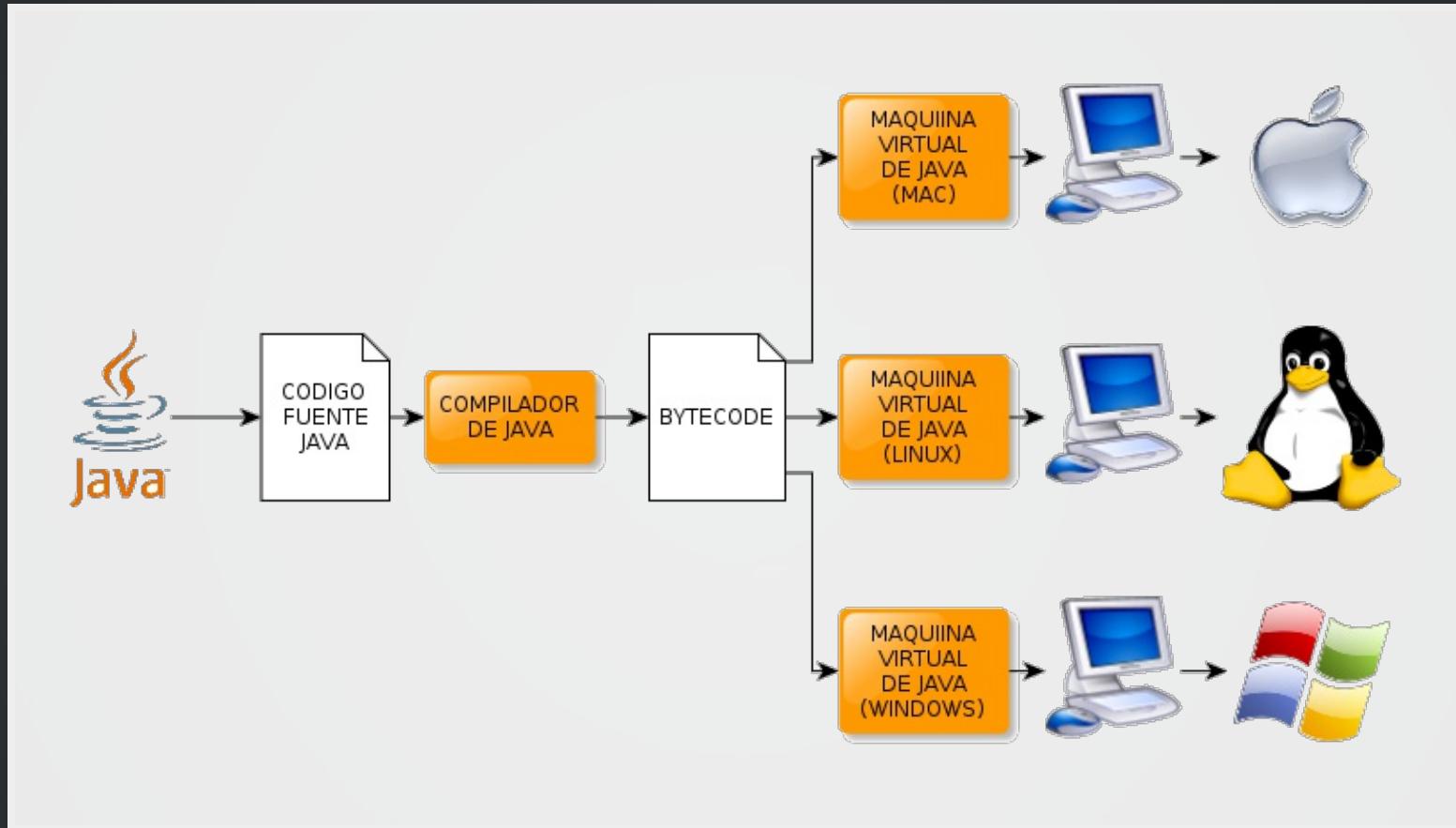
- En 1991 **James Gosling** creó una plataforma para ser usada en **pequeños dispositivos**.
- En 1994 se reorientó hacia la **web**.
- En 1995 **Netscape** anunció que sería soportado en sus **navegadores**.
- En 1996 aparece **JDK 1.0**.
- En 2001 ya es el lenguaje de programación más popular según el **Índice TIOBE**.
- En 2004 aparece **J2SE 5.0**
- En 2006 cambia su licencia a **GPL**.
  - En 2008 se convierte en el lenguaje de programación de **Android** (versión 5.0)
- En 2009 **Oracle Corporation compra Sun Microsystems**.
  - En 2010 **Oracle demanda a Google** por el uso de patentes de Java en Android.

# 2.17 ENTORNO DE JAVA



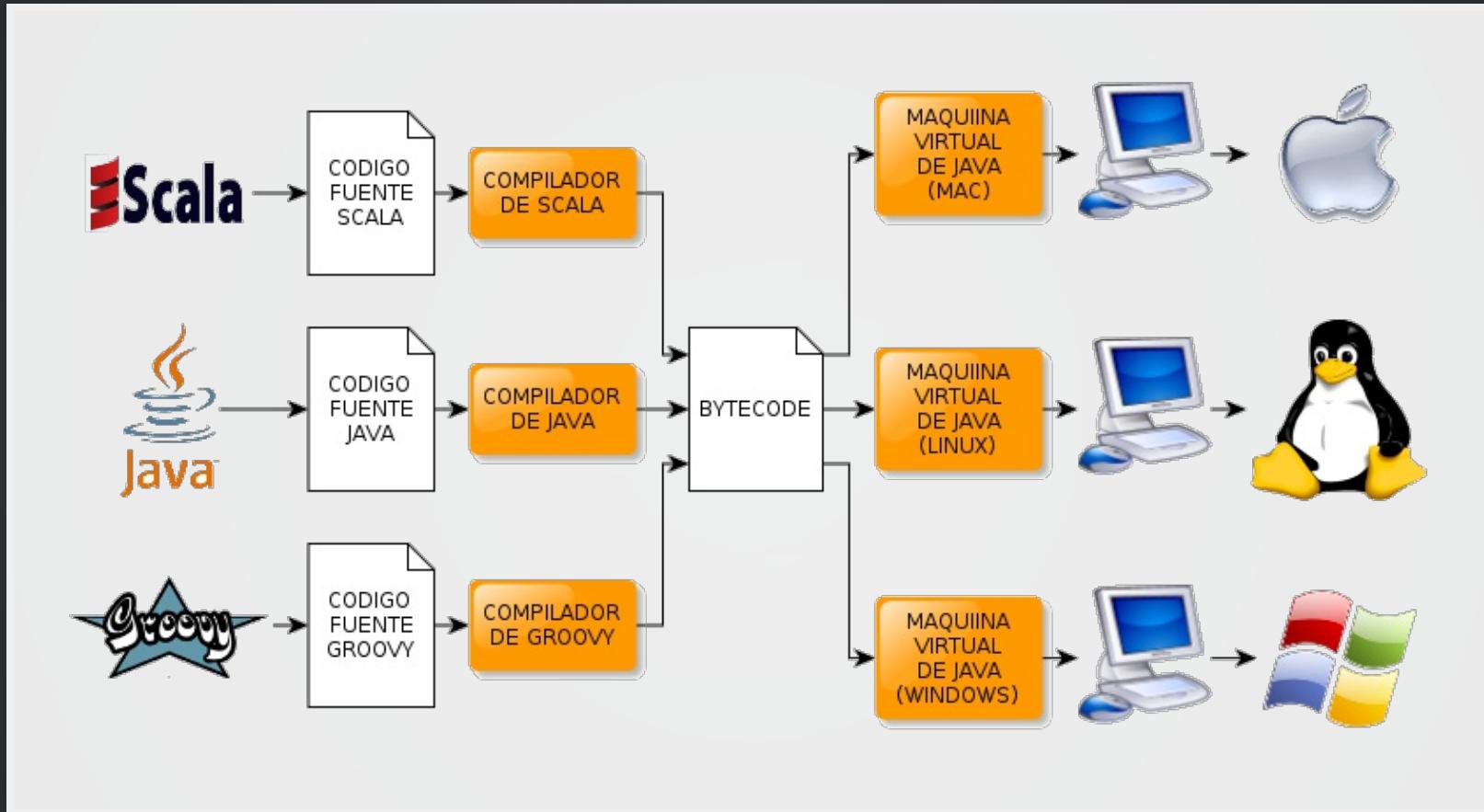
Entorno de Java

# 2.18 EJECUCIÓN DE JAVA



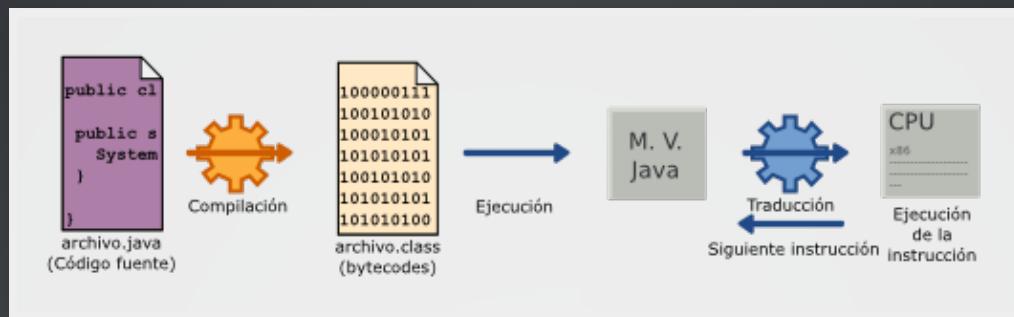
Ejecución de Java

# 2.19 EJECUCIÓN DE OTROS LENGUAJES

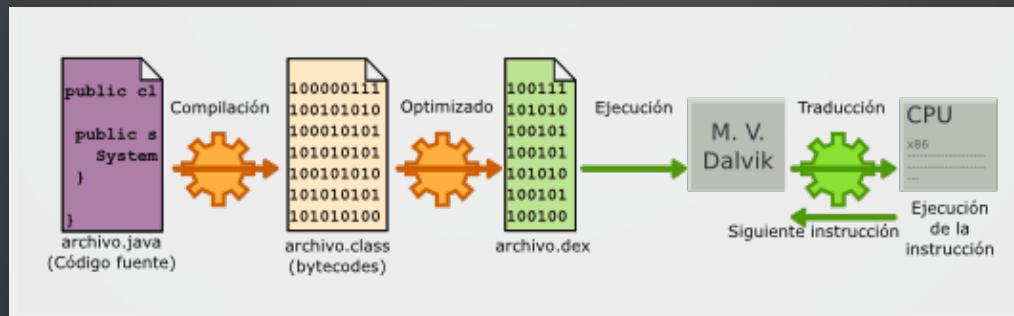


Ejecución de otros lenguajes

# 2.20 JAVA VS ANDROID



Compilación y ejecución en la Máquina Virtual de Java

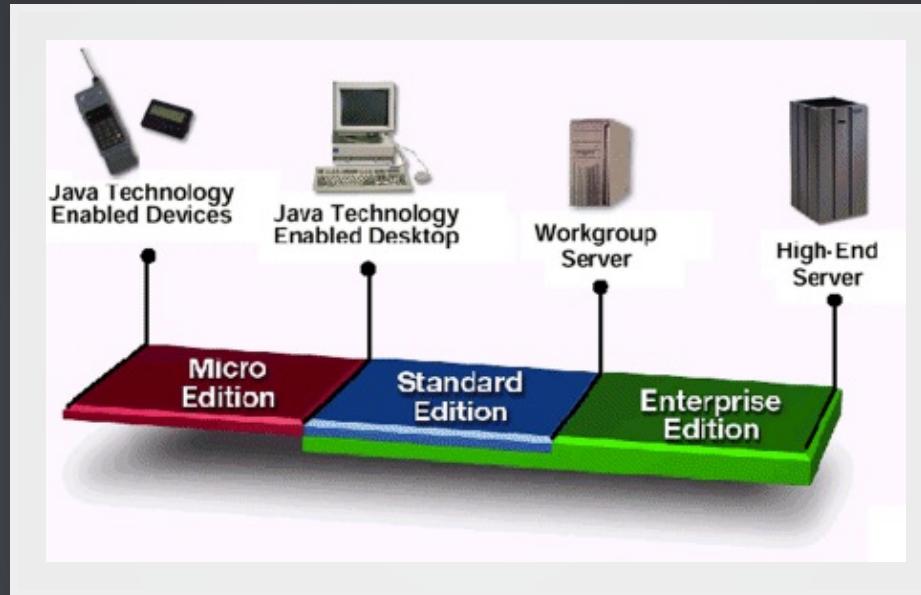


Compilación y ejecución en la Máquina Virtual de Dalvik



## Ejecución de aplicaciones en la MV de Java y en la MV de Dalvik

## 2.21 DISTINTAS PLATAFORMAS JAVA



Distintas plataformas Java

- **Android** usa sus propia máquina virtual, con sus propias APIs (muy parecidas a JSE pero no al 100% pues no tiene ni AWT ni SWING)

# 3 INSTALACIÓN Y CONFIGURACIÓN

# 3.1 REQUISITOS HARDWARE

- Sobre todo mucha **RAM**:
  - Si queremos utilizar un IDE como Eclipse y el plugin ADT de Android:
    - Windows: mínimo 3 GB, recomendable 4 GB
    - Linux: mínimo 2 GB, recomendable 3 GB

## 3.2 REQUISITOS SOFTWARE

- Imprescindible:
  - **JDK** (Java Development Kit)
  - **Android SDK** (Android Software Development Kit)
  - **Android SDK Components:**
    - SDK tools, SDK platform-tools, documentation, samples.
    - SDK platform(s) par el **AVD** (Android Virtual Device)
- Recomendable:
  - **Eclipse**
  - **Plugin ADT de Android para Eclipse**

# 3.3 INSTALACIÓN JDK Y ECLIPSE EN WINDOWS

- Descargar el JDK de:
  - <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- Instalar el JDK:
  - Siguiente, siguiente, siguiente...
- Descargar "Eclipse Classic":
  - <http://eclipse.org/downloads/>
- Descomprimir Eclipse en una carpeta:
  - Botón derecho, descomprimir...

# 3.4 INSTALACIÓN JDK Y ECLIPSE EN LINUX

- Instalar Eclipse (con su dependencia de Java - openjdk-)

```
apt-get install eclipse
```

# 3.5 INSTALACIÓN DEL SDK EN WINDOWS

- Descargar el SDK (\*.exe) de:
  - <http://developer.android.com/sdk/index.html>
- Instalar el SDK:
  - Siguiente, siguiente, siguiente...

## 3.6 INSTALACIÓN DEL SDK EN LINUX

- Descargar el SDK (\*.tgz) de:
  - <http://developer.android.com/sdk/index.html>
- Descomprimir SDK en una carpeta:
  - Botón derecho, descomprimir...

# 3.7 INSTALACIÓN DEL ADT EN EL ECLIPSE

- Instalar el plugin ADT (Android Development Tool) en Eclipse:
  - Abrir el Eclipse
  - **Help > Install New Software...**
  - <http://dl-ssl.google.com/android/eclipse/>
  - Siguiente, siguiente, siguiente...
- Configurar el plugin ADT (Android Development Tool) en Eclipse:
  - Abrir el Eclipse
  - **Window > Preferences > Android > SDK Location**
  - Poner la ruta completa a la carpeta del SDK
- Descargas adicionales:
  - Abrir el Eclipse

- **Window > Android SDK and AVD Manager**
- En **Available Packages** > seleccionar y descargar.

## 3.8 CONFIGURAR UN AVD

- Podemos probar las aplicaciones en el emulador AVD (**Android Virtual Device**)
- Para crear un AVD:
  - Abrir el Eclipse
  - **Window > Android SDK and AVD Manager**
  - En **Virtual Devices** > pinchamos en **New**:
    - Añadimos nombre descriptivo,
    - seleccionamos el target (versión) de Android,
    - seleccionamos **snapshot** para que se cargue más rápido
    - y añadimos características hardware (resolución de pantalla, cámara, GPS, etc.)

# 3.9 CONFIGURACIÓN

- Recomendable actualizar el PATH con los siguientes directorios:
  - <sdk>/tools
  - <sdk>/platform-tools
- En Windows:
  - Botón derecho en "Mi PC" > "Propiedades" > Pestaña "Avanzado" > Botón "Variables de Entorno"
  - Doble click en la variable PATH que está en "Variables del Sistema"
  - Añadir a la variable PATH la ruta completa a los directorios: /tools y /platform-tools
- En Linux:
  - Editar el fichero ~/.bashrc file
  - Añadir:

```
export PATH=$PATH:<sdk>/tools:<sdk>/platform-tools
```

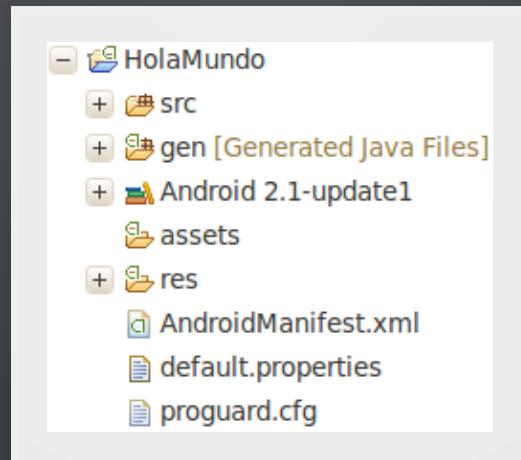
# 3.10 CÓDIGO FUENTE ANDROID.JAR

- Para **depurar**, si queremos también depurar las clases de android, nos hace falta el código fuente.
- El **código fuente** se puede descargar directamente desde <http://source.android.com/> pero es complicado, pues está dividido en ramas, versiones y módulos.
- Lo mejor es buscar un **zip** con todos los fuentes para un target en concreto como por ejemplo desde <http://android.opensourceror.org/2010/01/18/android-source/>.
- Ese zip hay que enlazarlo al android.jar que está dentro del target android del proyecto:
  - Botón derecho en //android.jar
  - Properties
  - Java Source Attachment
  - Rellenar Location path

# 4 ESTRUCTURA GENERAL DE UN PROYECTO

# 4.1 CARPETAS

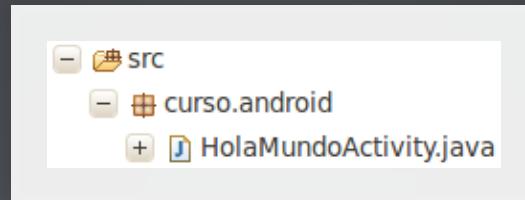
- Cuando creamos un nuevo proyecto Android en Eclipse genera automáticamente la estructura de carpetas necesarias.
- Esta estructura será común a cualquier aplicación, independientemente de su tamaño y complejidad.
- Leer  
<http://developer.android.com/guide/topics/resources/index.html>



Carpetas de un proyecto

## 4.2 CARPETA SRC

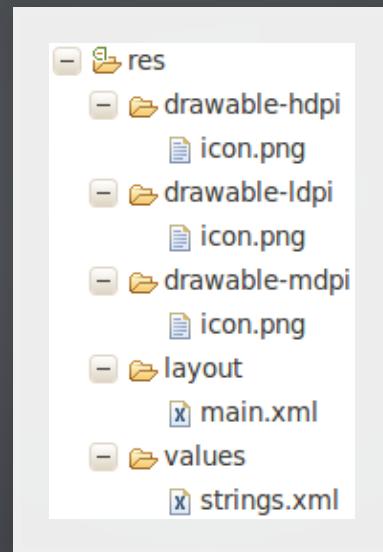
- Contiene todo el **código fuente** de la aplicación, código de la interfaz gráfica, clases auxiliares, etc.



La carpeta src

# 4.3 CARPETA RES (I)

- Contiene todos los **ficheros de recursos** necesarios para el proyecto:
  - imágenes, vídeos, cadenas de texto, etc.



La carpeta res

## 4.4 CARPETA RES (II)

- Los diferentes tipos de recursos se deberán distribuir entre las siguientes carpetas:
  - **/res/drawable/** Contienen las imágenes de la aplicación.
    - Se puede dividir en función de la resolución del dispositivo:
      - **/drawable-ldpi** para resoluciones bajas
      - **/drawable-mdpi** para resoluciones medias
      - **/drawable-hdpi** para resoluciones altas
  - **/res/layout/** Contienen los ficheros de definición de la interfaz gráfica.
    - Se puede dividir en función de la orientación del dispositivo:
      - **/layout-port** para orientaciones tipo

**/layout-port** para orientaciones tipo  
'portrait'

- **/layout-land** para orientaciones tipo  
'landscape'

# 4.5 CARPETA RES (III)

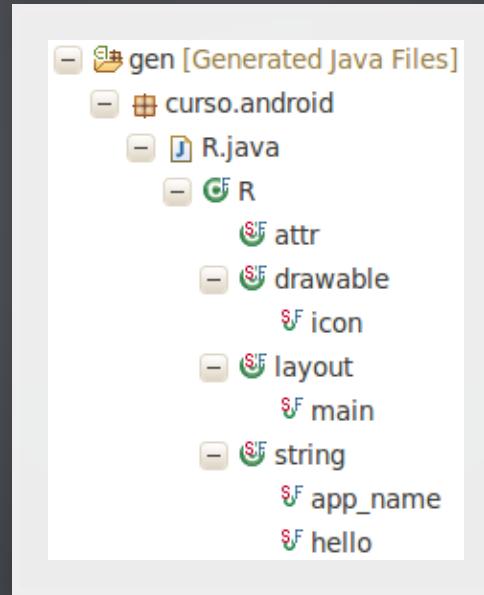
- Otra carpeta muy importante es la de **values**:
  - **/res/values/**
    - Se puede dividir por idiomas:
      - **/res/values-es/**
      - **/res/values-fr/**
    - Contiene otros recursos de la aplicación:
      - **strings.xml** para las cadenas de texto
      - **styles.xml** para los estilos
      - **colors.xml** para los colores

# 4.6 CARPETA RES (III)

- Hay otras carpetas menos importantes que también se llegan a usar:
  - **/res/anim/** Contiene la definición de las animaciones de la aplicación.
  - **/res/menu/** Contiene la definición de los menús de la aplicación.
  - **/res/xml/** Contiene los ficheros XML utilizados por la aplicación.
  - **/res/raw/** Contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyan en el resto de las carpetas de recursos.
- Ver  
<http://developer.android.com/guide/topics/resources/providing-resources.html>

# 4.7 CARPETA GEN

- Contiene una clase **generada automáticamente** al compilar el proyecto (no tocar), que referencia cada uno de los recursos de nuestra aplicación.



La carpeta gen

## 4.8 CARPETA ASSETS

- Contiene todos los demás ficheros auxiliares necesarios para la aplicación, como ficheros de configuración, de datos, etc.

# 4.9 FICHERO ANDROIDMANIFEST.XML

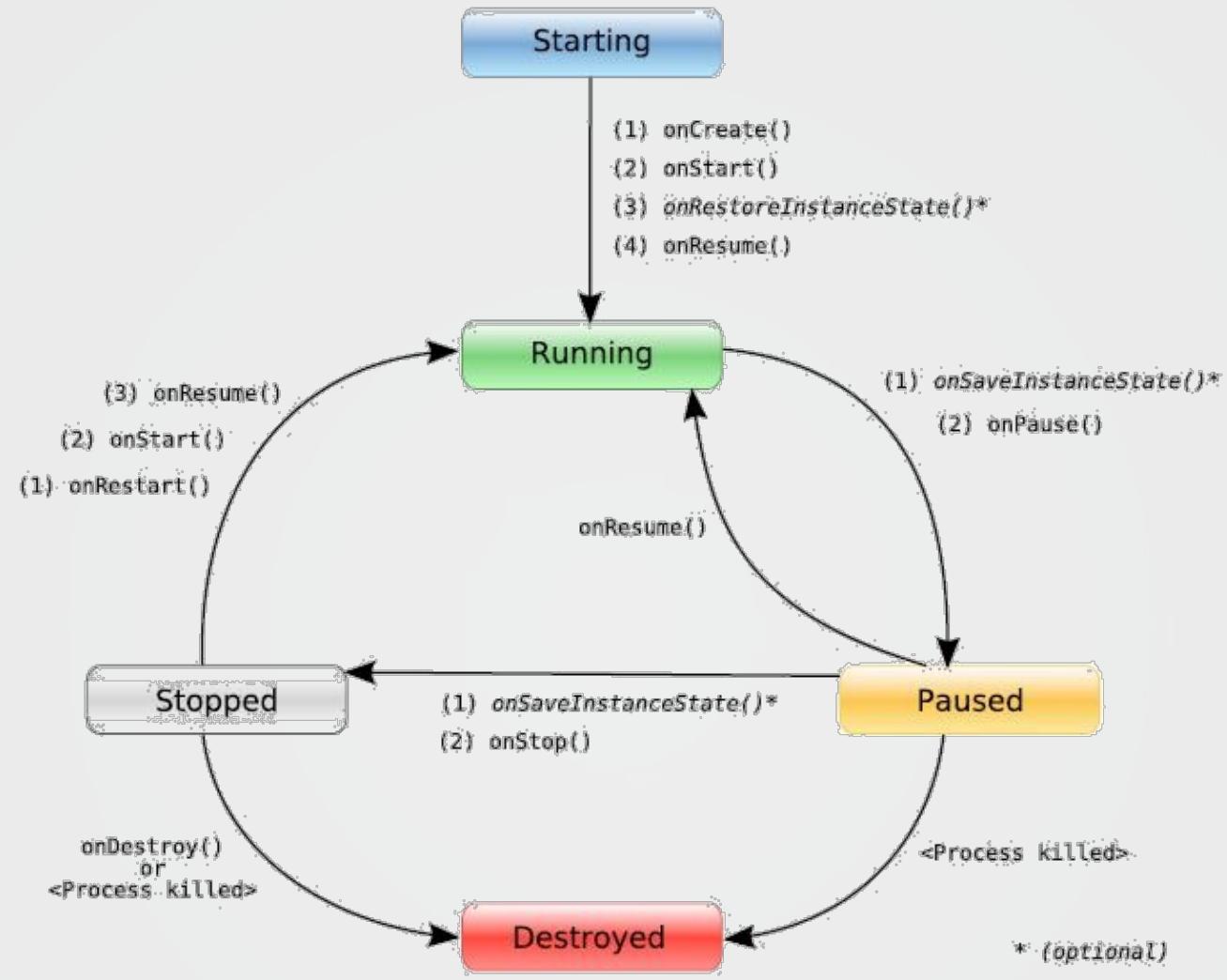
- Contiene la definición de los **aspectos principales** de la aplicación, como por ejemplo:
  - su identificación (nombre, versión, icono, etc.),
  - sus componentes (pantallas, mensajes, etc.),
  - o los **permisos** necesarios para su ejecución.

# 5 COMPONENTES PRINCIPALES

# 5.1 ACTIVITY

- Una **actividad** representa una ventana de la aplicación
- Está asociada con una vista.
- Puede escuchar los eventos originados en los distintos gráficos.
- Leer  
<http://developer.android.com/guide/topics/fundamentals>

## 5.2 CICLO DE VIDA DE UN ACTIVITY



## Ciclo de vida de un Activity

## 5.3 VIEW

- Una **vista** se puede programar por código o mediante XML (aconsejable).
- Existen un gran número de controles (cuadros de texto, botones, listas desplegables, etc.).
- Podemos crearnos nuestros propios controles.
- A los distintos controles se les puede asociar eventos.
- Existen también distintos layouts (linear layout, table layout, relative layout, etc.) donde agrupar distintos controles.
- Leer  
<http://developer.android.com/guide/topics/ui/index.html>

## 5.4 SERVICE

- Un **servicio** es un componente sin interfaz gráfica que segundo plano.
- Leer  
<http://developer.android.com/guide/topics/fundamentals/services.html>

# 5.5 CONTENT PROVIDER

- Un **proveedor de contenidos**, comparte datos con otras aplicaciones.
- Leer  
<http://developer.android.com/guide/topics/providers/content-providers.html>

## 5.6 BROADCAST RECEIVER

- Un **receptor de difusiones**, es un componente destinado a interceptar determinados mensajes generados por el sistema.
- Leer  
<http://developer.android.com/reference/android/content/BroadcastReceiv>

# 5.7 APP WIDGET

- Un **reproductor** es un elemento visual que se coloca en la pantalla principal del dispositivo y muestra información que puede ser actualizada periódicamente.
- Leer  
<http://developer.android.com/guide/topics/appwidgets>

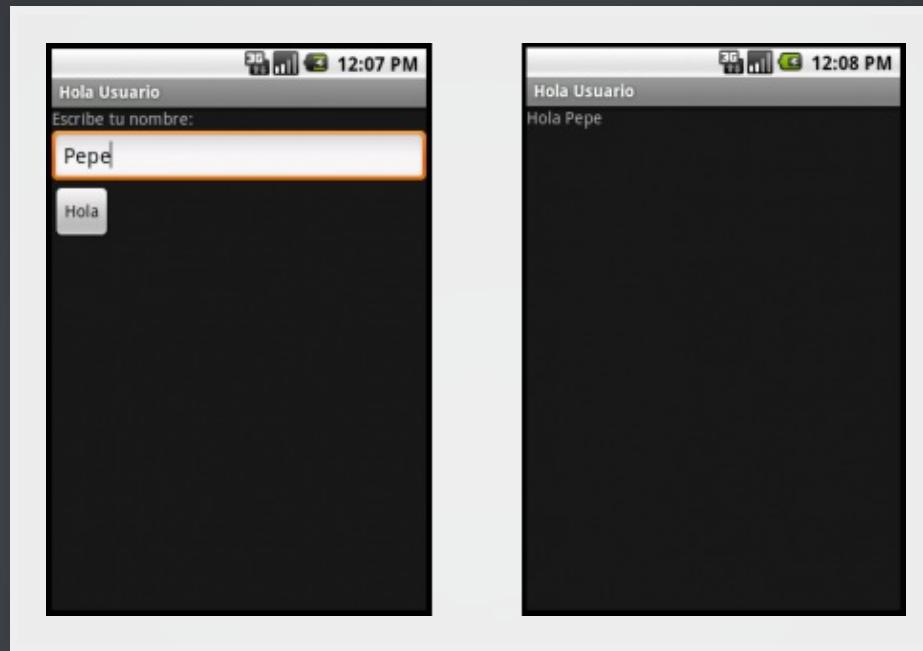
## 5.8 INTENT

- Un **propósito** es el elemento básico de comunicación entre los distintos elementos que hemos descrito anteriormente.
- Son los mensajes que son enviados entre los distintos componentes de una aplicación, entre distintas aplicaciones o entre el sistema y otras aplicaciones.
- Leer

<http://developer.android.com/guide/topics/intents/intents-filters.html>

# 6 PRIMERA APLICACIÓN

# 6.1 HOLA USUARIO



Hola Usuario

## 6.2 NUEVO PROYECTO ANDROID

- Abrimos el Eclipse
- File > New > Other.. > Android Project
- Nombre "**HolaUsuario**"
- Actividad **Main**

# 6.3 /RES/VALUES/STRINGS.XML

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Hola Usuario</string>
    <string name="nombre">Escribe tu nombre:</string>
    <string name="hola">Hola</string>
</resources>
```

# 6.4 /RES/LAYOUT/MAIN.XML

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

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

    <TextView android:id="@+id/LblNombre"
        android:text="@string/nombre"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent" />

    <EditText android:id="@+id/TxtNombre"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent" />

    <Button android:id="@+id/BtnHola"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

```

# 6.5 /RES/LAYOUT/SALUDO.XML

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

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <TextView android:id="@+id/TxtSaludo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</LinearLayout>
```

# 6.6 USUARIOACTIVITY.JAVA

```
public class UsuarioActivity extends Activity {

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

        // Localizamos los controles
        final EditText txtNombre = (EditText) this.findViewById(R.id.TxtNombre);
        final Button btnHola = (Button) this.findViewById(R.id.BtnHola);

        // Añadimos un Listener al botón
        btnHola.setOnClickListener(new OnClickListener() {

            public void onClick(final View view) {
                // Creamos el Intent
                final Intent intent = new Intent(UsuarioActivity.this, Saludo.class);
                ...
            }
        });
    }
}
```

# 6.7 SALUDOJAVA

```
public class Saludo extends Activity {  
  
    public void onCreate(final Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        this.setContentView(R.layout.saludo);  
  
        // Localizamos los controles  
        final TextView txtSaludo = (TextView) this.findViewById(R.id.TxtSaludo);  
  
        // Recuperamos la información pasada en el Intent  
        final Bundle bundle = this.getIntent().getExtras();  
  
        // Construimos el mensaje a mostrar  
        final String mensaje = this.getString(R.string.hola) + " " + bundle.  
  
        // Mostramos el mensaje  
        txtSaludo.setText(mensaje);  
    }  
}
```

# 6.8 ANDROIDMANIFEST.XML

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

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="curso.android" android:versionCode="1" android:versionName="1.0.0"

    <uses-sdk android:minSdkVersion="7" />

    <application android:icon="@drawable/icon" android:label="@string/app_name"
        <activity android:name=".UsuarioActivity" android:label="@string/app_name"
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".Saludo" />
```

# 7 LAYOUTS

# 7.1 FRAME LAYOUT

- Coloca todos sus controles hijos alineados con su esquina superior izquierda, de forma que cada control quedará oculto por el control siguiente (a menos que éste último tenga transparencia).
- Suele utilizarse para mostrar un único control en su interior.
- Los componentes deberán establecer
  - sus propiedades:
    - **android:layout\_width** y
    - **android:layout\_height**
  - con los valores:
    - **fill\_parent**: para que el componente tome la dimensión de su layout.
    - **wrap\_content**: para que el componente tome la dimensión de su contenido.

# 7.2 EJEMPLO FRAMELAYOUT

```
<FrameLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent">  
  
    <EditText android:id="@+id/Texto"  
        android:layout_width="fill_parent"  
        android:layout_height="wrap_content" />  
  
</FrameLayout>
```

# 7.3 LINEARLAYOUT

- Apila uno tras otro los componentes de forma horizontal o vertical según se establezca su propiedad **android:orientation**.
- Al igual que en FrameLayout, los componentes deberán establecer sus propiedades **android:layout\_width** y **android:layout\_height** para determinar sus dimensiones.
- Además podrán establecer la propiedad **android:layout\_weight**:
  - Nos permite dimensionar los componentes de forma proporcional.
  - En el ejemplo **Texto2** ocupará el doble que **Texto1**.
- Leer  
<http://developer.android.com/resources/tutorials/views/layoutlinear.html>

# 7.4 EJEMPLO LINEARLAYOUT

```
<LinearLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:orientation="vertical">  
  
    <EditText android:id="@+id/Texto1"  
        android:layout_width="fill_parent"  
        android:layout_height="fill_parent"  
        android:layout_weight="1" />  
  
    <EditText android:id="@+id/Texto2"  
        android:layout_width="fill_parent"  
        android:layout_height="fill_parent"  
        android:layout_weight="2" />  
  
</LinearLayout>
```

# 7.5 TABLELAYOUT

- Permite distribuir los componentes en filas y columnas
- Los componentes podrán establecer las propiedades:
  - **android:layout\_span** si quieren expandirse por más una columna,
  - **android:gravity="left|center|right"** si quieren alinear texto.
- Leer  
<http://developer.android.com/resources/tutorials/views/tablelayout.html>

# 7.6 EJEMPLO TABLELAYOUT

```
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <TableRow>
        <TextView android:text="Celda 1.1" />
        <TextView android:text="Celda 1.2" android:gravity="right" />
    </TableRow>

    <TableRow>
        <TextView android:text="Celda 2" android:layout_span="2" />
    </TableRow>

</TableLayout>
```

# 7.7 RELATIVELAYOUT

- Permite especificar la posición de cada componente de forma relativa.
- En el ejemplo, el botón **BtnAceptar**:

```
<!-- debajo del cuadro de texto TxtNombre -->  
    android:layout_below="@+id/TxtNombre")
```

```
<!-- a la derecha del layout padre -->  
    android:layout_alignParentRight="true")
```

```
<!-- dejará un margen a su izquierda de 10 density-independent-pixels -->  
    android:layout_marginLeft="10dp") .
```

- Leer  
<http://developer.android.com/resources/tutorials/views/relativelayout.html>

# 7.8 EJEMPLO RELATIVELAYOUT

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <EditText android:id="@+id/TxtNombre"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

    <Button android:id="@+id/BtnAceptar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/TxtNombre"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dp" />

</RelativeLayout>
```

# 7.9 PROPIEDADES DEL RELATIVELAYOUT (I)

- Posición relativa a otro control:

```
android:layout_above.  
android:layout_below.  
android:layout_toLeftOf.  
android:layout_toRightOf.  
android:layout_alignLeft.  
android:layout_alignRight.  
android:layout_alignTop.  
android:layout_alignBottom.  
android:layout_alignBaseline.
```

# 7.10 PROPIEDADES DEL RELATIVELAYOUT (II)

- Posición relativa al layout padre:

```
android:layout_alignParentLeft  
android:layout_alignParentRight  
android:layout_alignParentTop  
android:layout_alignParentBottom  
android:layout_centerHorizontal  
android:layout_centerVertical  
android:layout_centerInParent
```

# 7.11 PROPIEDADES DE LOS LAYOUTS

- Opciones de margen:

```
android:layout_margin  
android:layout_marginBottom  
android:layout_marginTop  
android:layout_marginLeft  
android:layout_marginRight
```

- Opciones de espaciado o padding :

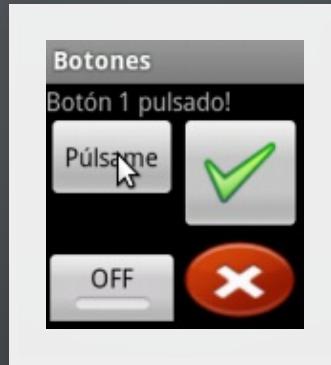
```
android:padding  
android:paddingBottom  
android:paddingTop  
android:paddingLeft  
android:paddingRight
```

**8 BOTONES**

# 8.1 BUTTON

- Botón con texto 'Púlsame' definido en la propiedad **android:text**.

```
<Button android:id="@+id/Boton1"  
        android:text="Púlsame"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content" />
```

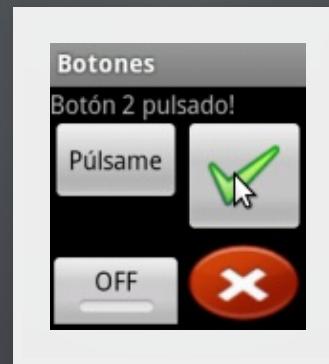


Button

## 8.2 IMAGEBUTTON

- Botón con una 'imagen' definida en la propiedad **android:src**.

```
<ImageButton android:id="@+id/Boton2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/validate" />
```

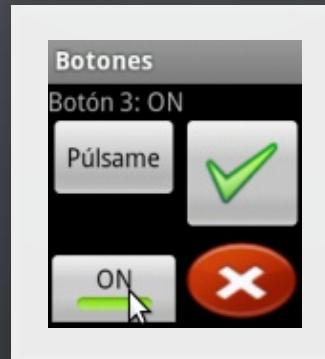


ImageButton

# 8.3 TOGGLEBUTTON

- Tiene dos estados:
  - **pulsado** cuyo texto se asigna en la propiedad **android:textOn** y
  - **no pulsado** cuyo texto se asigna en la propiedad **android:textOff**.

```
<ToggleButton android:id="@+id/Boton3"  
    android:textOn="ON"  
    android:textOff="OFF"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```



# ToggleButton

# 8.4 EVENTOS (I)

```
final ToggleButton boton3 = (ToggleButton) findViewById(R.id.Boton3);

boton3.setOnClickListener(new View.OnClickListener() {
    public void onClick(final View view) {
        if(boton3.isChecked()) {
            mensaje.setText("Botón 3: ON");
        } else {
            mensaje.setText("Botón 3: OFF");
        }
    }
});
```

# 8.5 EVENTOS (II)

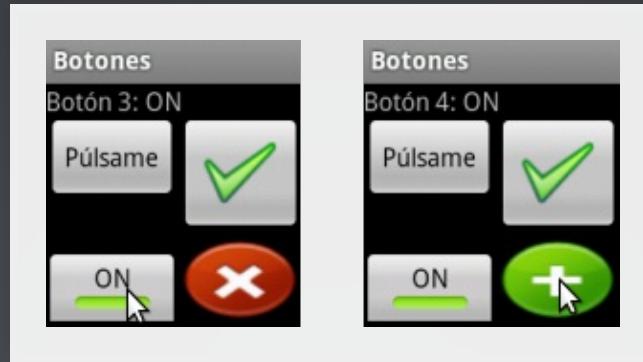
```
final ToggleButton boton3 = (ToggleButton) findViewById(R.id.Boton3);

boton3.setOnClickListener(new View.OnClickListener() {
    public void onClick(final View view) {
        final ToggleButton botonEvento = (ToggleButton) view;
        if(botonEvento.isChecked()) {
            mensaje.setText("Botón 3: ON");
        } else {
            mensaje.setText("Botón 3: OFF");
        }
    }
});
```

## 8.6 OTRAS PROPIEDADES:

- color de fondo (android:background),
- estilo de fuente (android:typeface),
- color de fuente (android:textcolor),
- tamaño de fuente (android:textSize),
- etc.

# 8.7 EJEMPLO PERSONALIZACIÓN



Botón personalizado

# 8.8 PERSONALIZAR EL ASPECTO

- /res/drawable/toggle-style.xml

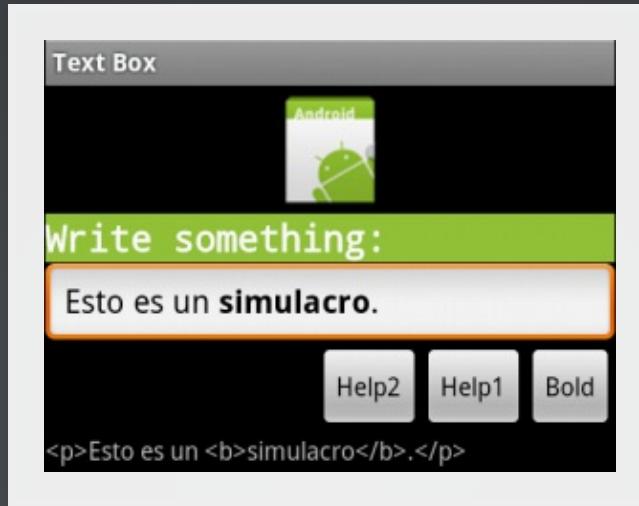
```
<?xml version="1.0" encoding="UTF-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_checked="false" android:drawable="@drawable/ko"
    <item android:state_checked="true" android:drawable="@drawable/ok" />
</selector>
```

- /res/layout/main.xml

```
<ToggleButton android:id="@+id/Boton4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn=""
    android:textOff=""
    android:background="@drawable/toggle-style" />
```

# 9 IMÁGENES Y TEXTOS

# 9.1 IMÁGENES Y TEXTOS EN ANDROID



Imágenes y textos en Android

## 9.2 IMAGEVIEW

- Permite mostrar imágenes en la aplicación.
- Propiedades:
  - **android:src** indica la imagen a mostrar,
  - **android:maxWidth** indica el ancho máximo,
  - **android:maxHeight** indica la altura máxima,
  - etc.
- Se puede cambiar la imagen mediante código:

```
final ImageView foto = (ImageView) findViewById(R.id.Foto);  
foto.setImageResource(R.drawable.otra_foto);
```

## 9.3 EJEMPLO IMAGEVIEW

```
<ImageView android:id="@+id/Foto"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/foto" />
```

## 9.4 TEXTVIEW

- Se utiliza para mostrar un texto al usuario, mediante propiedad **android:text**.
- Otras propiedades:
  - **android:background** (color de fondo),
  - **android:textColor** (color del texto),
  - **android:textSize** (tamaño de la fuente),
  - **android:textStyle** (estilo del texto: normal, negrita, cursiva),
  - **android:typeface** (tipo de letra: normal, sans, serif, monospace),
  - etc.
- También se puede cambiar el texto mediante código:

```
final TextView label = (TextView) findViewById(R.id.Label);  
label.setText(texto);
```

# 9.5 EJEMPLO TEXTVIEW

```
<TextView android:id="@+id/Label"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/Icon"
    android:text="@string/write"
    android:textSize="20dp"
    android:textColor="#FFFFFF"
    android:textStyle="bold"
    android:background="#97C03D"
    android:typeface="monospace" />
```

# 9.6 EDITTEXT

- Permite la introducción y edición de texto por parte del usuario.
- El texto a mostrar por defecto en la caja de texto se establece en la propiedad **android:text**.
- También se puede cambiar el texto mediante código:

```
final EditText cuadroTexto = (EditText) findViewById(R.id.Label);  
cuadroTexto.setText(texto);
```

# 9.7 EJEMPLO EDITTEXT

```
<EditText android:id="@+id/TextBox"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_below="@id/Label" />
```

# 9.8 SPANNED, SPANNABLE Y EDITABLE

- Un **Spanned** es una cadena de caracteres a la que podemos insertar otros objetos a modo de marcas o etiquetas (spans) asociados a rangos de caracteres.
- La interfaz **Spannable**, hereda de Spanned, y permite la modificación de esas marcas.
- La interfaz **Editable**, hereda de Spannable, y permite además la modificación del texto.

# 9.9 SPANS

- Existen muchos tipos de **spans** predefinidos, entre otros:
  - TypefaceSpan. Modifica el tipo de letra (normal, sans, serif, monospace).
  - StyleSpan. Modifica el estilo del texto (normal, negrita, cursiva).
  - ForegrouudColorSpan. Modifica el color del texto.
  - AbsoluteSizeSpan. Modifica el tamaño de fuente.

```
// Creamos un texto con parte en negrita
final Editable texto = Editable.Factory.getInstance().newEditable("Esto
texto.setSpan(new StyleSpan(Typeface.BOLD), 11, 20, Spanned.SPAN_EXCLUSI
textBox.setText(texto);
```

# 9.10 LA CLASE HTML

- Muestra el texto con etiquetas de formato HTML:

```
final String html = Html.toHtml(textBox.getText());
```

- Asignar texto con formato HTML (sólo funciona con las etiquetas básicas):

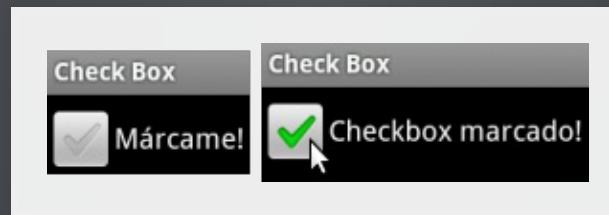
```
textBox.setText(Html.fromHtml("Otro <b>texto</b> de ejemplo."), BufferType
```

# 10 CHECKBOXS Y RADIOPUTTONS

# 10.1 CHECKBOX

- Se suele utilizar para marcar o desmarcar opciones en una aplicación.

```
<CheckBox android:id="@+id/ChkMarcame"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Márcame!" />
```



CheckBox

# 10.2 MÉTODOS CHECKBOX

- Podremos hacer uso de los métodos:
  - **isChecked()** para conocer el estado del control, y
  - **setChecked(estado)** para establecer un estado concreto para el control.

```
if (checkBox.isChecked()) {  
    checkBox.setChecked(false);  
}
```

# 10.3 EVENTOS CHECKBOX

```
final CheckBox cb = (CheckBox) findViewById(R.id.ChkMarcame);

cb.setOnCheckedChangeListener(new CheckBox.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            cb.setText("Checkbox marcado!");
        } else {
            cb.setText("Checkbox desmarcado!");
        }
    }
});
```

# 10.4 RADIobutton

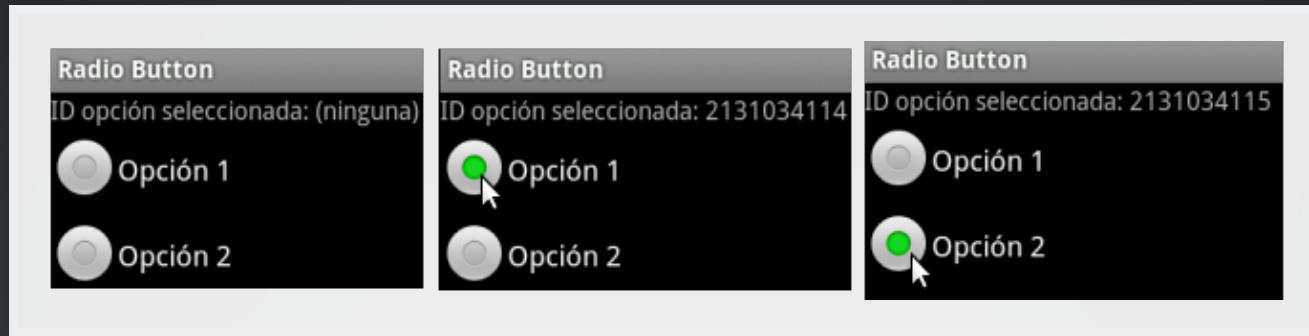
- Puede estar marcado o desmarcado.
- Se utilizan dentro de un **RadioGroup** donde sólo una puede estar marcada.

```
<RadioGroup android:id="@+id/gruporb" android:orientation="vertical"
    android:layout_width="fill_parent"    android:layout_height="fill_parent"

    <RadioButton android:id="@+id/radio1" android:text="Opción 1"
        android:layout_width="wrap_content" android:layout_height="wrap_content"

    <RadioButton android:id="@+id/radio2" android:text="Opción 2"
        android:layout_width="wrap_content" android:layout_height="wrap_content"

</RadioGroup>
```



## RadioButton

# 10.5 MÉTODOS RADIOGROUP

- Los más importantes:
  - **check(id)** para marcar una opción determinada mediante su ID,
  - **clearCheck()** para desmarcar todas las opciones, y
  - **getCheckedRadioButtonId()** que devolverá el ID de la opción marcada (o el valor -1 si no hay ninguna marcada).

```
final RadioGroup rg = (RadioGroup) findViewById(R.id.gruporb);
rg.clearCheck();
rg.check(R.id.radio1);
int idSeleccionado = rg.getCheckedRadioButtonId();
```

# 10.6 EVENTOS RADIOGROUP

```
final RadioGroup rg = (RadioGroup) findViewById(R.id.gruporb);

rg.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
    public void onCheckedChanged(RadioGroup group, int checkedId) {
        lblMensaje.setText("ID opcion seleccionada: " + checkedId);
    }
});
```

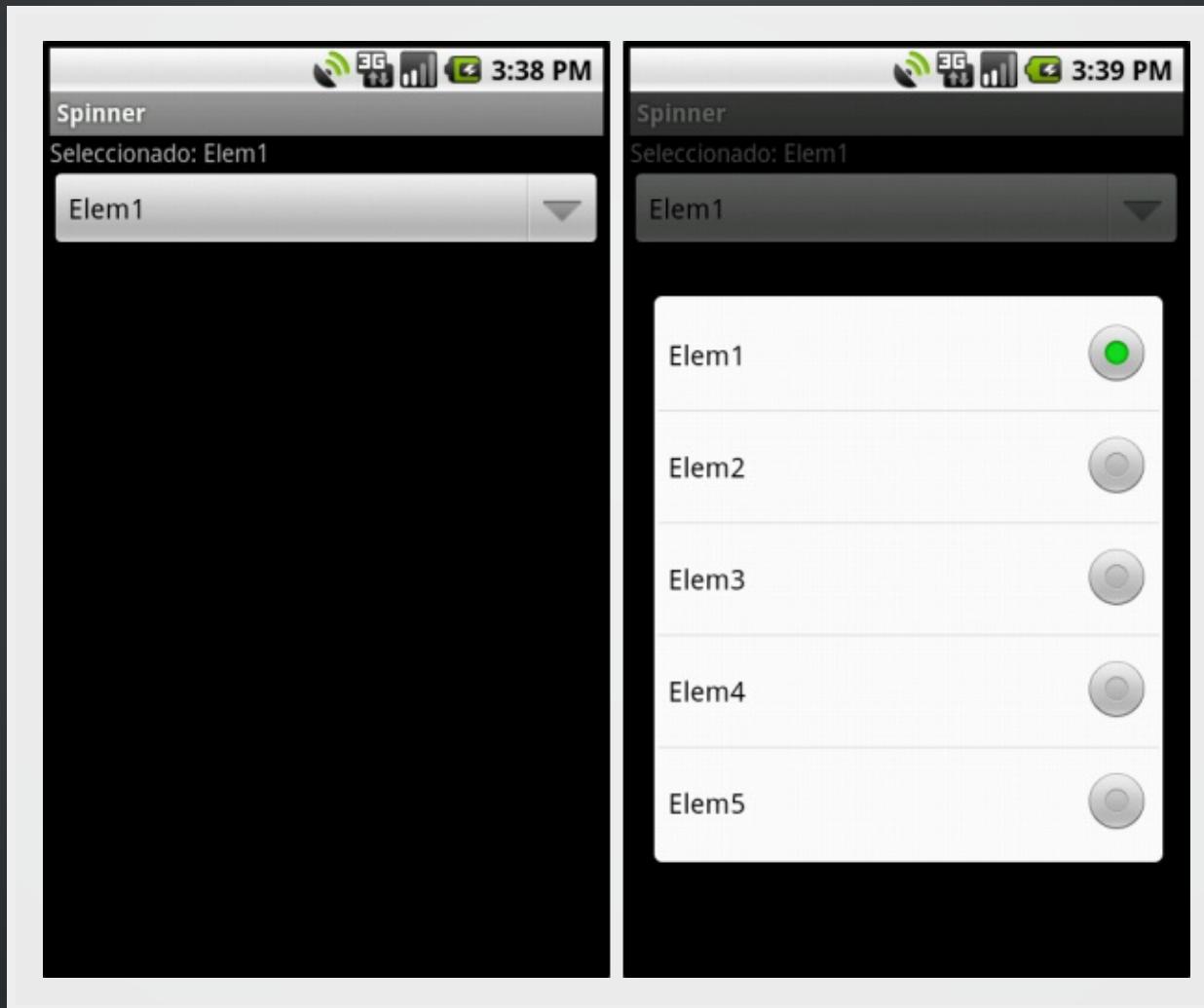
**11 LISTAS**

# 11.1 ADAPTERS

- Necesarios para renderizar los items de las listas.
- Aunque podemos crear los nuestros propios, existen adaptadores sencillos:
  - **ArrayAdapter**: El más sencillo. Utiliza array de objetos.
  - **SimpleAdapter**: Para mapear datos sobre los controles de un fichero XML de layout.
  - **SimpleCursorAdapter**: Para mapear las columnas de un cursor sobre elementos visuales.

# 11.2 SPINNER

- Son listas desplegables.



# Spinner

# 11.3 CÓDIGO SPINNER

```
<Spinner  
    android:id="@+id/CmbOpciones"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content" />
```

```
final String[] datos = new String[]{"Elem1", "Elem2", "Elem3", "Elem4", "Elem5"};  
  
final Spinner cmbOpciones = (Spinner) findViewById(R.id.CmbOpciones);  
  
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(context,  
    android.R.layout.simple_spinner_item,  
    datos);  
  
// para indicar como mostrar la lista de elementos  
adaptador.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);  
  
cmbOpciones.setAdapter(adaptador);
```

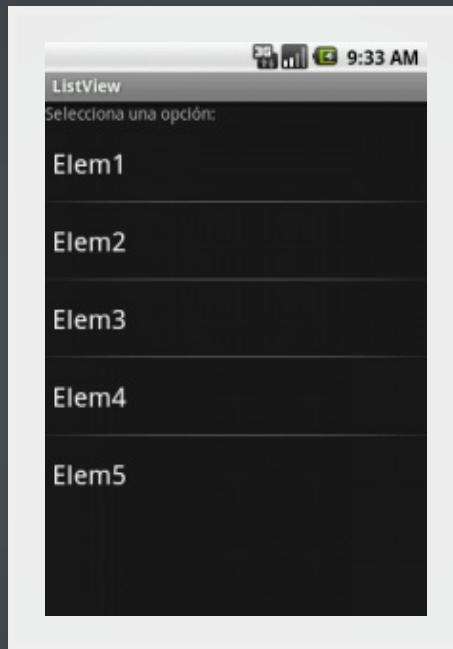
# 11.4 EVENTOS SPINNER

```
cmbOpciones.setOnItemSelectedListener(new OnItemSelectedListener() {
    public void onItemSelected(
        AdapterView<?> parent, View view, int position, long id) {
        lblMensaje.setText("Seleccionado: " + datos[position]);
    }

    public void onNothingSelected(AdapterView<?> parent) {
        lblMensaje.setText("");
    }
});
```

# 11.5 LISTVIEW

- Muestra una lista seleccionable:



ListView

# 11.6 CÓDIGO LISTVIEW

```
<ListView  
    android:id="@+id/LstOpciones"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```

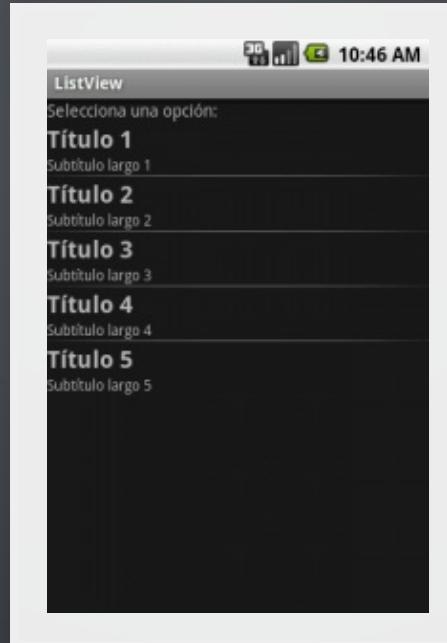
```
final String[] datos = new String[]{"Elem1", "Elem2", "Elem3", "Elem4", "Elem5"};  
  
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(context,  
    android.R.layout.simple_list_item_1,  
    datos);  
  
ListView lstOpciones = (ListView) findViewById(R.id.LstOpciones);  
lstOpciones.setAdapter(adaptador);
```

# 11.7 EVENTOS LISTVIEW

```
lstOpciones.setOnItemClickListener(new OnItemClickListener() {  
    public void onItemClick(  
        AdapterView<?> parent, View view, int position, long id) {  
        //Acciones necesarias al hacer click  
    }  
});
```

# 11.8 DATOS COMPLEJOS

```
public class Titular {  
  
    private String titulo;  
    private String subtítulo;  
  
    // constructor y getters  
}
```



Datos complejos en una lista

# 11.9 ITEM LAYOUT PERSONALIZADO

- /res/layout/listitem-titular.xml

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

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

    <TextView android:id="@+id/Titulo"
        android:layout_width="fill_parent"    android:layout_height="wrap_content"
        android:textStyle="bold"                android:textSize="20dp" />

    <TextView android:id="@+id/Subtitulo"
        android:layout_width="fill_parent"    android:layout_height="wrap_content"
        android:textStyle="normal"            android:textSize="12dp" />

</LinearLayout>
```

# 11.10 ADAPTADOR PERSONALIZADO

```
public class TitularesAdapter extends ArrayAdapter<Titular> {

    public View getView(final int position, final View listItem, final View

        // recojemos el titular para esa posición
        final Titular titular = this.getItem(position);

        // inflamos la vista desde el xml
        final Activity context = (Activity) this.getContext();
        listItem = context.getLayoutInflater().inflate(R.layout.listitem_tit

        // cogemos los elementos de la vista
        final TextView titulo = (TextView) listItem.findViewById(R.id.Titulo);
        final TextView subtitle = (TextView) listItem.findViewById(R.id.Sub

        // modificamos los textos de la vista (el titulo y el el subtitle)
        titulo.setText(titular.getTitulo());
        subtitle.setText(titular.getSubtitulo());
```

# 11.11 CÓDIGO LISTVIEW PERSONALIZADO

```
Titular[] datos = new Titular[] {  
    new Titular("Título 1", "Subtítulo largo 1"),  
    new Titular("Título 2", "Subtítulo largo 2"),  
    new Titular("Título 3", "Subtítulo largo 3"),  
    new Titular("Título 4", "Subtítulo largo 4"),  
    new Titular("Título 5", "Subtítulo largo 5")};  
  
AdaptadorTitulares adaptador = new AdaptadorTitulares(this, datos);  
  
ListView lstOpciones = (ListView) findViewById(R.id.LstOpciones);  
  
lstOpciones.setAdapter(adaptador);
```

# 11.12 GRIDVIEW

- Muestra opciones divididas en filas y columnas.



# GridView

# 11.13 CÓDIGO GRIDVIEW

```
<GridView android:id="@+id/GridOpciones" android:stretchMode="columnWidth"
          android:layout_width="fill_parent"
          android:layout_height="fill_parent"
          android:numColumns="auto_fit"
          android:horizontalSpacing="5dp"
          android:columnWidth="80dp"
          android:verticalSpacing="10dp"
```

```
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(
    context,
    android.R.layout.simple_list_item_1,
    datos);

final GridView grdOpciones = (GridView) findViewById(R.id.GridOpciones);

grdOpciones.setAdapter(adaptador);
```

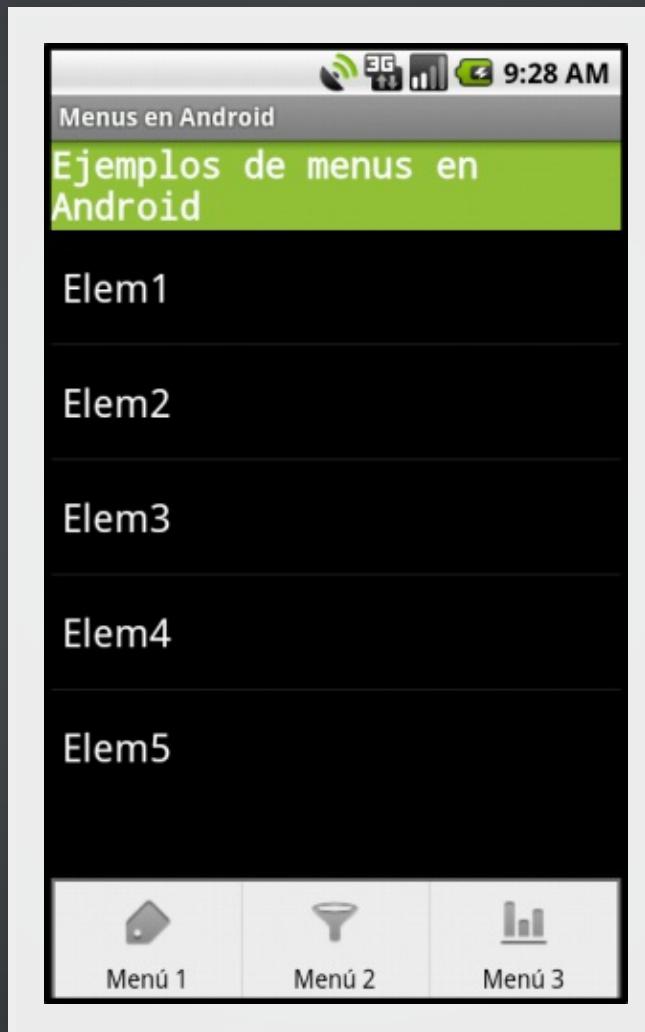
# 11.14 EVENTOS GRIDVIEW

```
grdOpciones.setOnItemSelectedListener(new OnItemSelectedListener() {
    public void onItemSelected(
        AdapterView<?> parent, View view, int position, long id) {
        lblMensaje.setText("Seleccionado: " + datos[position]);
    }

    public void onNothingSelected(AdapterView<?> parent) {
        lblMensaje.setText("");
    }
});
```

# 12 MENÚS

# 12.1 MENÚ NORMAL



Menú normal

# 12.2 CÓDIGO

- Se puede hacer (como todas las vistas) por código o mediante XML:

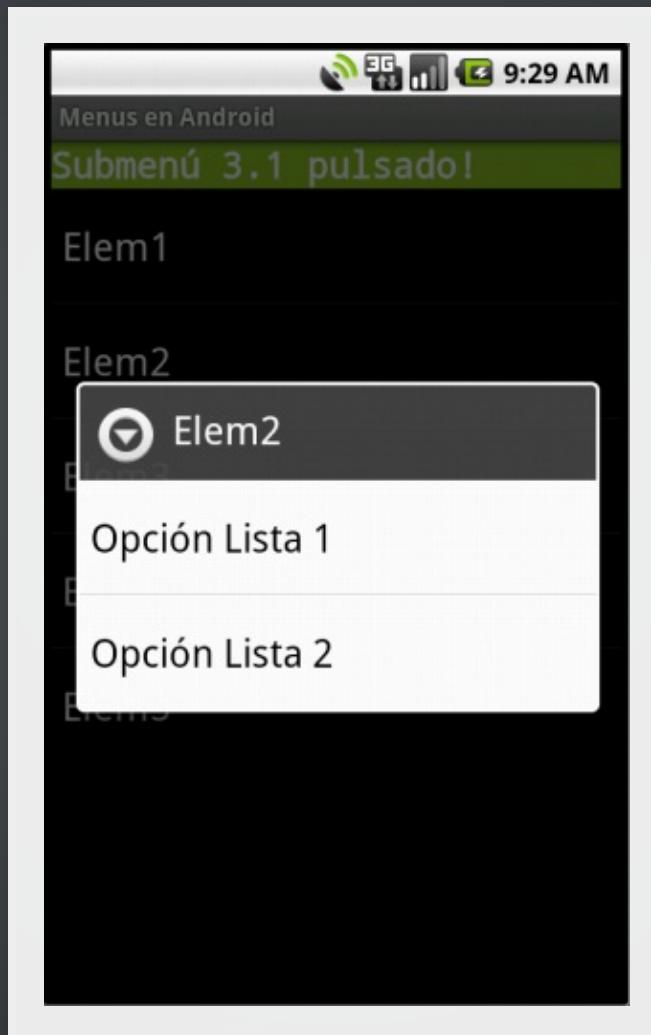
```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/Menu1" android:title="Menú 1" android:icon="@dr
    <item android:id="@+id/Menu2" android:title="Menú 2" android:icon="@dr
    <item android:id="@+id/Menu3" android:title="Menú 3" android:icon="@dr
        <menu>
            <item android:id="@+id/Submenu31" android:title="Submenú 3.1" />
            <item android:id="@+id/Submenu32" android:title="Submenú 3.2" />
        </menu>
    </item>
</menu>
```

```
public boolean onCreateOptionsMenu(Menu menu) {
    final MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu_principal, menu);
    return true;
}
```

# 12.3 EVENTO DE MENÚ

```
public boolean onOptionsItemSelected(final MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.Menu1:  
            this.mensaje.setText("Menú 1 pulsado!");  
            return true;  
        case R.id.Menu2:  
            this.mensaje.setText("Menú 2 pulsado!");  
            return true;  
        case R.id.Menu3:  
            this.mensaje.setText("Menú 3 pulsado!");  
            return true;  
        case R.id.Submenu31:  
            this.mensaje.setText("Submenú 3.1 pulsado!");  
            return true;  
        case R.id.Submenu32:  
            this.mensaje.setText("Submenú 3.2 pulsado!");  
            return true;  
    }  
}
```

# 12.4 MENÚ CONTEXTUAL



Menú contextual

# 12.5 LAYOUT MENÚ CONTEXTUAL

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/OpcionLista1" android:title="Opción Lista 1" />
    <item android:id="@+id/OpcionLista2" android:title="Opción Lista 2" />
</menu>
```

# 12.6 ASOCIAR MENÚ CONTEXTUAL

```
public void onCreate(final Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    this.setContentView(R.layout.main);  
  
    // Obtenemos las referencias a los controles  
    this.lista = (ListView) this.findViewById(R.id.Lista);  
  
    ...  
  
    // Asociamos los menús contextuales a los controles  
    this.registerForContextMenu(this.lista);  
}
```

# 12.7 CREAR MENÚ CONTEXTUAL

```
public void onCreateContextMenu(ContextMenu menu, View view, ContextMenu
super.onCreateContextMenu(menu, view, menuInfo);

final AdapterContextMenuInfo info = (AdapterContextMenuInfo) menuInfo;
menu.setHeaderTitle(this.lista.getAdapter().getItem(info.position).toS

final MenuInflater inflater = this.getMenuInflater();

inflater.inflate(R.menu.menu_contextual_lista, menu);
}
```

# 12.8 EVENTO MENÚ CONTEXTUAL

```
public boolean onContextItemSelected(final MenuItem item) {  
  
    final AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();  
  
    switch (item.getItemId()) {  
        case R.id.OpcionLista1:  
            this.mensaje.setText("Lista[" + info.position + "]: Opcion 1 pulsada");  
            return true;  
        case R.id.OpcionLista2:  
            this.mensaje.setText("Lista[" + info.position + "]: Opcion 2 pulsada");  
            return true;  
        default:  
            return super.onContextItemSelected(item);  
    }  
}
```

13 DIALOGOS

# 13.1 ALERTDIALOG

- Muestra un mensaje pidiendo confirmación al usuario para continuar.

```
final AlertDialog.Builder alert = new AlertDialog.Builder(context);
alert.setTitle("Alerta!");
alert.setMessage("Este es el texto de la alerta");
alert.setIcon(R.drawable.icon);
alert.setPositiveButton(android.R.string.ok, null);
alert.show();
```



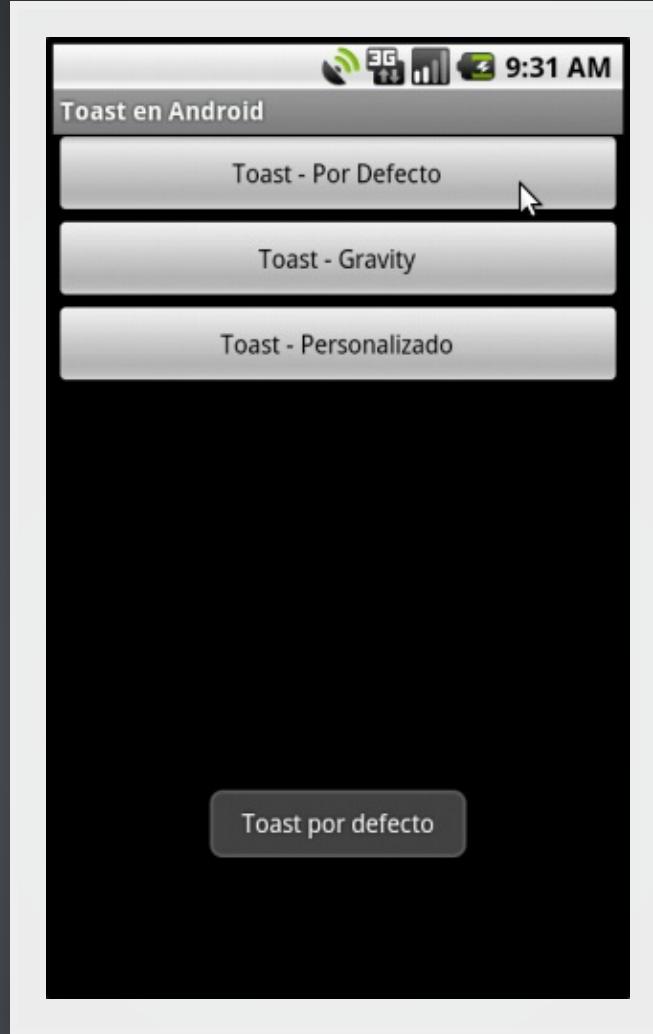
# AlertDialog

# 13.2 TOAST

- Mensaje que se muestra en pantalla durante unos segundos al usuario.
- No deberían utilizarse para hacer notificaciones demasiado importantes.
- La clase Toast dispone de los siguientes métodos:
  - **makeText(contexto, texto, duracion)** con duración LENGTH\_LONG o LENGTH\_SHORT
  - **show()** para mostrarlo
  - **setGravity()** con los valores CENTER, LEFT, RIGHT, TOP, BOTTOM
  - **setDuration(duracion)** con duración LENGTH\_LONG o LENGTH\_SHORT
  - **setView(layout)** para personalizarlo con un layout.

# 13.3 TOAST POR DEFECTO

```
Toast toast = Toast.makeText(  
    getApplicationContext(), "Toast por defecto", Toast.LENGTH_SHORT);  
toast.show();
```



# Toast por defecto

# 13.4 TOAST CON GRAVITY

```
Toast toast = Toast.makeText(  
    getApplicationContext(), "Toast por defecto", Toast.LENGTH_SHORT);  
toast.setGravity(Gravity.LEFT, 0, 0);  
toast.show();
```



# Toast con gravity

# 13.5 TOAST PERSONALIZADO (I)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/lytLayout"
    android:layout_width="fill_parent"
    android:background="#555555"
    android:orientation="horizontal"
    android:layout_height="fill_parent"
    android:padding="5dp" >

    <ImageView android:id="@+id/imgIcono"
        android:layout_height="wrap_content"
        android:src="@drawable/marcador"
        android:layout_width="wrap_content" />

    <TextView android:id="@+id/txtMensaje"
        android:layout_width="wrap_content"
        android:textColor="#FFFFFF"
        android:layout_gravity="center"
        android:layout_height="wrap_content"
        android:paddingLeft="10dp" />

</LinearLayout>
```

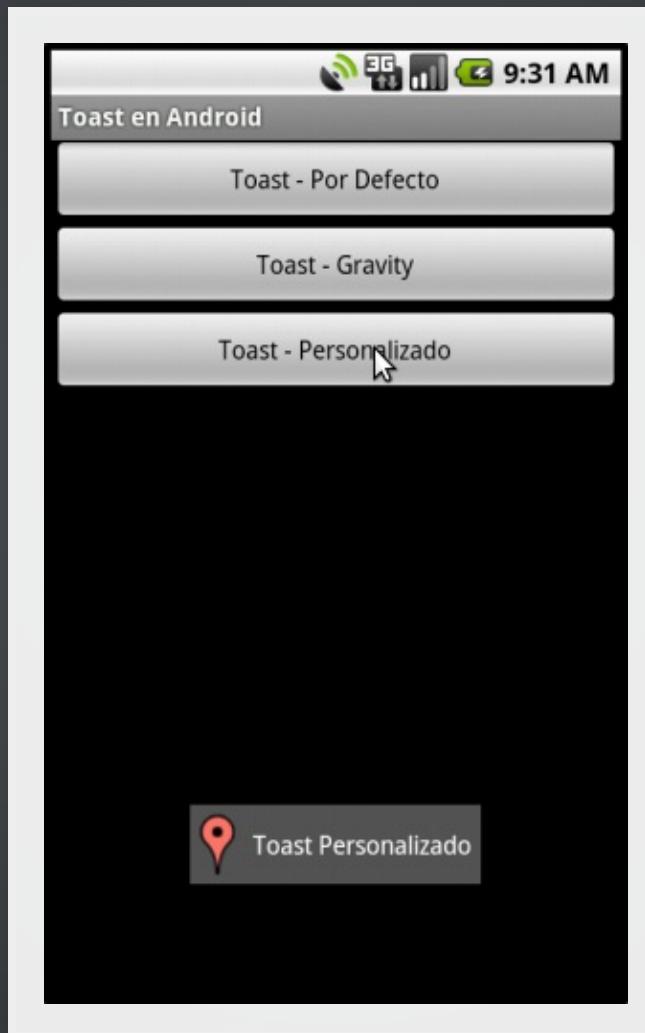
# 13.6 TOAST PERSONALIZADO (II)

```
// inflamos el layout
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(
    R.layout.toast_layout, (ViewGroup) findViewById(R.id.lytLayout));

// cogemos el campo de texto del layout inflado y ponemos el texto
TextView txtMsg = (TextView) layout.findViewById(R.id.txtMensaje);
txtMsg.setText("Toast Personalizado");

// nos creamos el toast, lo configuramos y lo mostramos
Toast toast = new Toast(getApplicationContext());
toast.setDuration(Toast.LENGTH_SHORT);
toast.setView(layout);
toast.show();
```

# 13.7 TOAST PERSONALIZADO (III)



Toast personalizado

## 13.8 NOTIFICACIÓN EN LA BARRA DE ESTADO

- Aparece un ícono en la **barra de estado**.
- Pinchando y arrastrando la barra de estado hacia abajo podemos ver los textos de las notificaciones.
- Suele usarse con servicios.

# 13.9 NOTIFICACIÓN SIMPLE

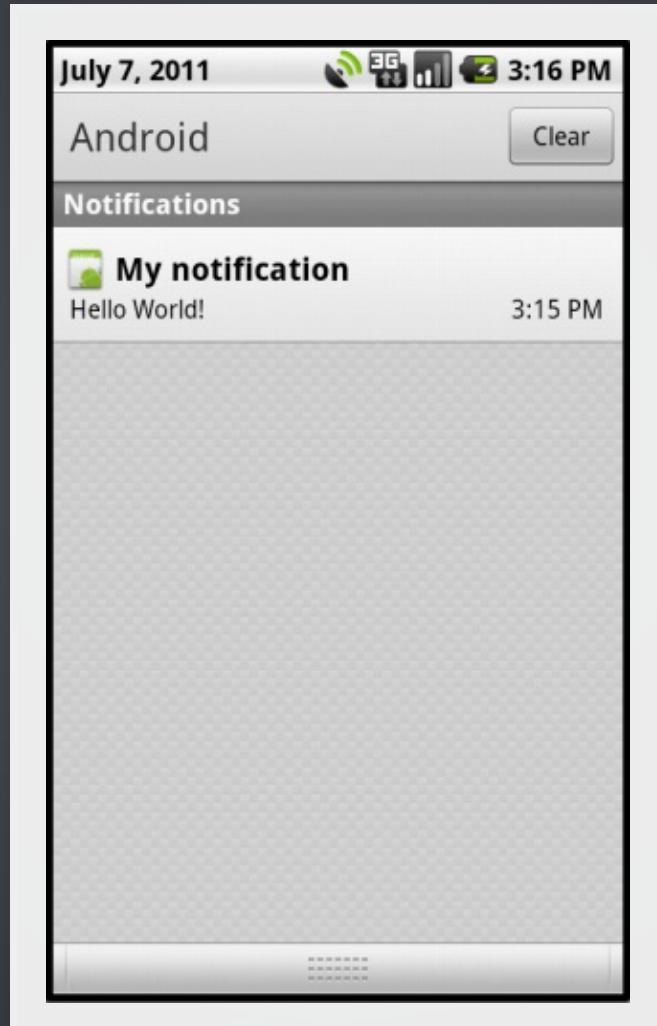
```
final Notification notification = new Notification(
    R.drawable.icon, "Texto barra de estado", System.currentTimeMillis());

final Intent notificationIntent = new Intent(this, MyClass.class);
final PendingIntent contentIntent = PendingIntent.getActivity(this, 0, r

notification.setLatestEventInfo(context, "Título", "Texto del mensaje",

final NotificationManager notificationManager =
    (NotificationManager) context.getSystemService(Context.NOTIFICATION_SE
notificationManager.notify(ID_NOTIFICACION, notification);
```

# 13.10 PANTALLA NOTIFICACIÓN SIMPLE



Notificación simple

# 13.11 NOTIFICACIÓN PERSONALIZADA

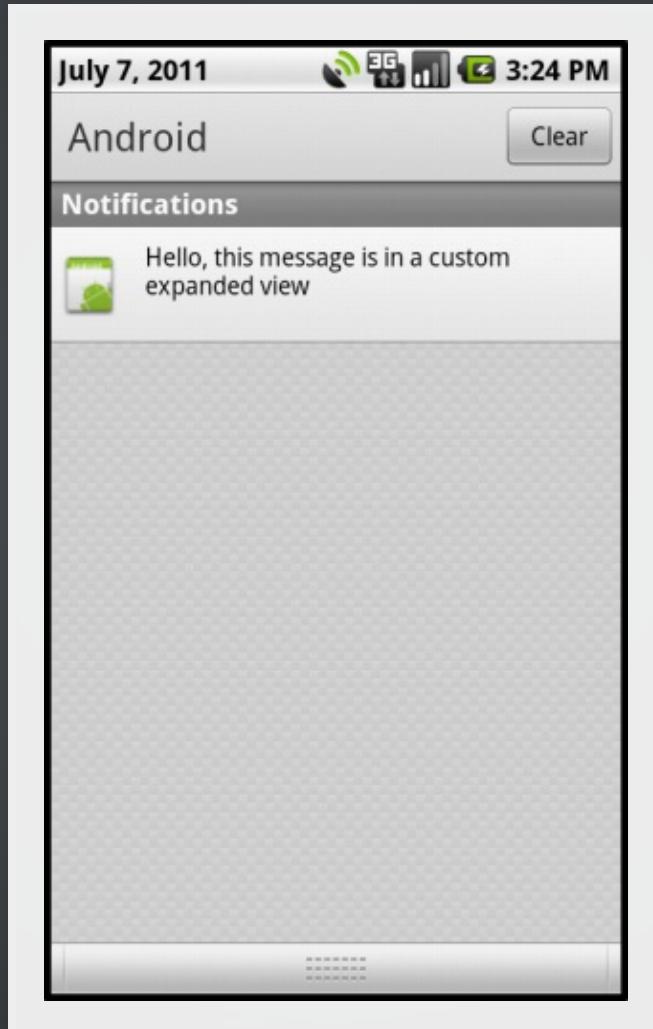
```
final Notification notification = new Notification(
    R.drawable.icon, "Texto barra de estado", System.currentTimeMillis());

final RemoteViews contentView = new RemoteViews(this.getPackageName(), R.layout.notification_content);
contentView.setImageResource(R.id.image, R.drawable.icon);
contentView.setTextViewText(R.id.text, "Texto del mensaje");
notification.contentView = contentView;

final Intent notificationIntent = new Intent(this, MyClass.class);
final PendingIntent contentIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);

final NotificationManager notificationManager =
    (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);
notificationManager.notify(ID_NOTIFICACION_PERSONALIZADA, notification);
```

# 13.12 PANTALLA NOTIFICACIÓN PERSONALIZADA



Notificación personalizada

# 14 FICHEROS

# 14.1 ENTRADA/SALIDA EN JAVA

- Todas las clases relacionadas con la Entrada/Salida estén en el paquete **java.io**
- Java lo que maneja en realidad son flujos, ya sean de **flujos de bytes o de caracteres**.
- Los **flujos de entrada** son de entrada de datos del exterior hacia el sistema.
- Los **flujos de salida** son de salida de datos del sistema hacia el exterior.
- Esto se hizo así, de forma genérica, para poder **manejar igual** todos los flujos: el de entrada/salida estándar (teclado y monitor), el de los ficheros o el de red, etc.
- Las clases del paquete `java.io` implementan el **patrón Decorator**.

# 14.2 FLUJOS DE BYTES EN JAVA

- **InputStream/OutputStream**
  - Clases abstractas que definen las funciones básicas de lectura y escritura de un **flujo de bytes sin estructurar**.
- **FileInputStream/FileOutputStream**
  - Pensadas para trabajar con **archivos binarios**.
- **BufferedInputStream/BufferedOutputStream**
  - **Decoradores** que añaden un **buffer** a los flujos de bytes.

# 14.3 FLUJOS DE CARACTERES EN JAVA

- **Reader/Writer**
  - **Clases abstractas** que definen las funciones básicas de lectura y escritura de un **flujo de caracteres**.
- **InputStreamReader/OutputStreamWriter**
  - **Convierten flujos** de bytes en flujos de caracteres.
- **FileReader/FileWriter**
  - Pensadas para trabajar con **archivos de texto**.
- **BufferedReader/BufferedWriter**
  - **Decoradores** que añaden un **buffer** a los flujos de caracteres.
- **PrintWriter**
  - Posee los métodos **print** y **println** que otorgan gran potencia a la escritura.

## 14.4 LA CLASE FILE DE JAVA

- **No** proporciona métodos de **lectura/escritura** a los archivos.
- **Sólo** proporciona operaciones a nivel de **sistema de archivos**:
  - gestión de permisos,
  - listado de archivos,
  - crear carpetas,
  - borrar ficheros,
  - cambiar nombre,
  - etc.

# 14.5 FICHEROS EN ANDROID

- Podremos almacenar ficheros en:
  - La memoria interna del dispositivo.
  - La tarjeta SD externa, si disponible.
  - La propia aplicación, en forma de recurso.
- Existen distintos modos de apertura:
  - **MODE\_PRIVATE** para acceso privado desde nuestra aplicación,
  - **MODE\_APPEND** para añadir datos a un fichero ya existente,
  - **MODE\_WORLD\_READABLE** para permitir a otras aplicaciones leer el fichero y
  - **MODE\_WORLD\_WRITABLE** para permitir a otras aplicaciones escribir sobre el fichero.
- Se utiliza los métodos:

SE DETALLA LOS MÉTODOS:

- **openFileOutput()** que devuelve un **FileOutputStream**
- **openFileInput()** que devuelve un **FileInputStream**
- **getResources().openRawResource(idFichero)** que devuelve un **InputStream**

# 14.6 ESCRIBIR FICHERO MEMORIA INTERNA

```
OutputStreamWriter fichero = null
try {
    // abrimos el fichero
    fichero = new OutputStreamWriter(
        openFileOutput("prueba_int.txt", Context.MODE_PRIVATE));
    // escribimos en el fichero
    fichero.write("Texto de prueba.");
} catch (final Exception e) {
    Log.e("Ficheros", "Error al escribir fichero en memoria interna", e);
} finally {
    try {
        if (fichero != null) fichero.close();
    } catch (final Exception e) {
        Log.e("Ficheros", "Error cerrando el fichero en memoria interna");
    }
}
```

# 14.7 DIRECTORIO MEMORIA INTERNA

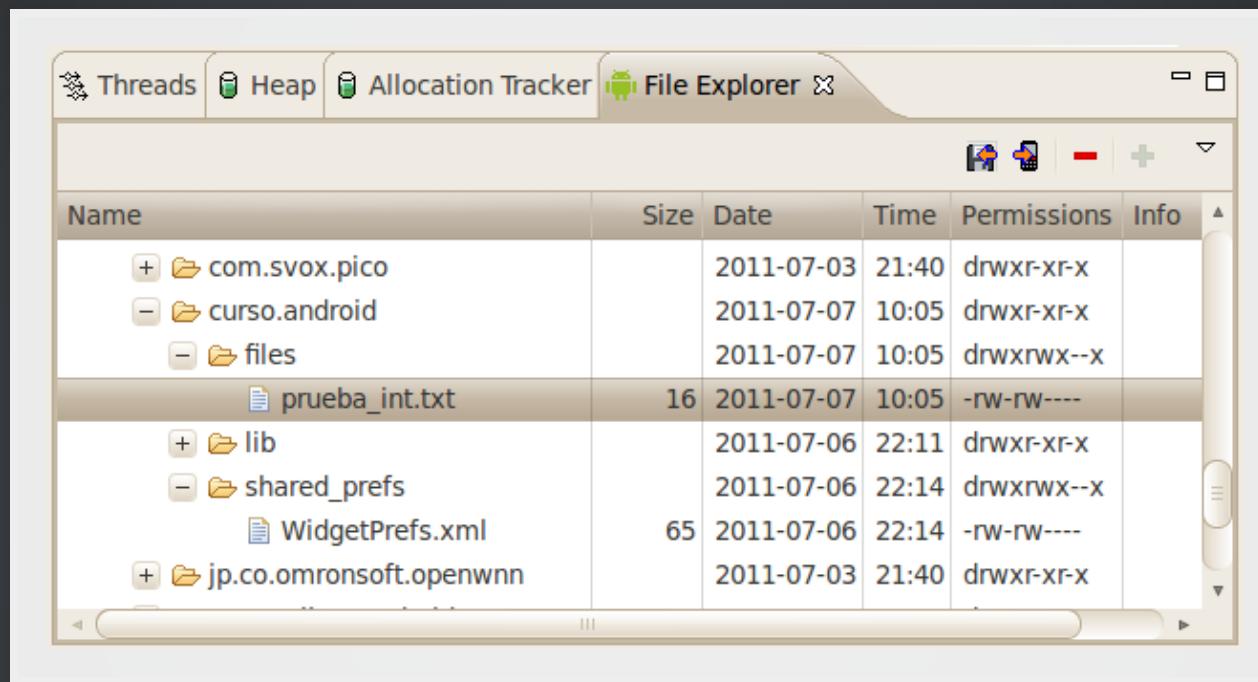
- Android almacena por defecto los ficheros creados en una ruta determinada:

```
/data/data/paquete.java.de.la.aplicacion/files/nombre_fichero
```

- En nuestro caso:

```
/data/data/curso.android/files/prueba_int.txt
```

# 14.8 MEMORIA INTERNA EN EL DDMS



Memoria interna en el DDMS

# 14.9 LEER FICHERO MEMORIA INTERNA

```
BufferedReader fichero = null;
try {
    // abrimos el fichero para lectura
    fichero = new BufferedReader(new InputStreamReader(
        openFileInput("prueba_int.txt")));
    // leemos el fichero linea a linea
    final StringBuilder texto = new StringBuilder();
    String linea = fichero.readLine();
    while (linea != null) {
        texto.append(linea);
        linea = fichero.readLine();
    }
} catch (final Exception e) {
    Log.e("Ficheros", "Error al leer fichero desde memoria interna", e);
} finally {
    if (fichero != null)
        try {
            fichero.close();
        } catch (IOException e) {
            Log.e("Ficheros", "Error al cerrar el fichero");
        }
}
```

# 14.10 LEER FICHERO DESDE RECURSO

```
BufferedReader fichero = null;
try {

    // abrimos el fichero para lectura
    fichero = new BufferedReader(new InputStreamReader(
        FicherosActivity.this.getResources().openRawResource(R.raw.prueba_ra

    // leemos el fichero linea a linea
    final StringBuilder texto = new StringBuilder();
    String linea = fichero.readLine();
    while (linea != null) {
        texto.append(linea);
        linea = fichero.readLine();
    }
} catch (final Exception e) {
    Log.e("Ficheros", "Error al leer fichero desde recurso raw", e);
} finally {
    try {
        if (fichero != null)
            fichero.close();
    } catch (IOException e) {
        Log.e("Ficheros", "Error al cerrar el fichero", e);
    }
}
```

# 14.11 LEER/ESCRIBIR EN TARJETA SD

- Tenemos que comprobar el estado con la función `Environment.getExternalStorageState()`:
  - Si es igual a `Environment.MEDIA_MOUNTED`
    - Podremos leer y escribir.
  - Si es igual a `Environment.MEDIA_MOUNTED_READ_ONLY`
    - Sólo podremos leer.
  - Ver otros estados en `Environment`
- Además habrá que dar permisos en el `AndroidManifest.xml`:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
```

# 14.12 ESCRIBIR EN TARJETA SD

```
// Si la memoria externa está disponible y se puede escribir
if (Environment.MEDIA_MOUNTED.equals(Environment.getExternalStorageState()))

    OutputStreamWriter fichero = null;
    try {

        // abrimos el fichero para escritura
        fichero = new OutputStreamWriter(new FileOutputStream(
            new File(Environment.getExternalStorageDirectory().getAbsolutePath()

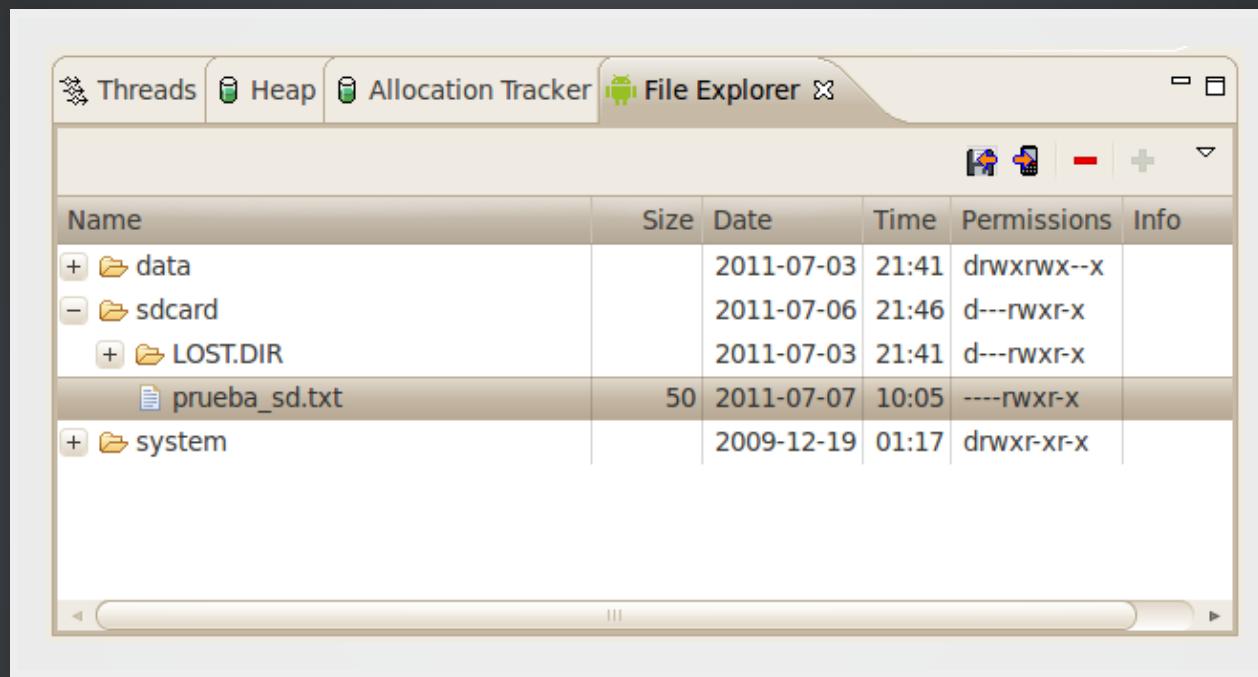
        // escribimos en el fichero
        fichero.write("Texto de prueba para el fichero guardado en la SD.");

    } catch (final Exception e) {
        Log.e("Ficheros", "Error al escribir fichero a tarjeta SD", e);
    } finally {
        try {
            if (fichero != null)
                fichero.close();
        } catch (IOException e) {
            Log.e("Ficheros", "Error al cerrar el fichero");
        }
    }
}
```

# 14.13 LEER EN TARJETA SD

```
// Si la memoria externa está disponible y se puede leer
if (Environment.MEDIA_MOUNTED.equals(Environment.getExternalStorageState())
    || Environment.MEDIA_MOUNTED_READ_ONLY.equals(Environment.getExternalStorageState()))
{
    BufferedReader fichero = null;
    try {
        // abrimos el fichero para lectura
        fichero = new BufferedReader(new InputStreamReader(new FileInputStream(
            new File(Environment.getExternalStorageDirectory()).getAbsolutePath())));
        // leemos el fichero linea a linea
        final StringBuilder texto = new StringBuilder();
        String linea = fichero.readLine();
        while (linea != null) {
            texto.append(linea);
            linea = fichero.readLine();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

# 14.14 TARJETA SD EN EL DDMS



Tarjeta SD en el DDMS

# 15 PREFERENCIAS

# 15.1 PREFERENCIAS EN ANDROID

- Son datos que una aplicación guarda para personalizar la experiencia del usuario.
- Cada preferencia se almacenará en forma de **clave-valor**.
- Se guardan en fichero XML.
- La clase **SharedPreferences** gestiona colecciones de preferencias, que se diferenciarán mediante un identificador único.
- Para obtener una colección utilizaremos el método **getSharedPreferences()** al que pasaremos el identificador de la colección y un modo de acceso:
  - **MODE\_PRIVATE**: Sólo nuestra aplicación tiene acceso.
  - **MODE\_WORLD\_READABLE**: Todas las

- **MODE\_WORLD\_READABLE**: Todas las aplicaciones pueden leer.
- **MODE\_WORLD\_WRITABLE**: Todas las aplicaciones pueden leer y modificar.

# 15.2 RECUPERAR PREFERENCIAS

```
SharedPreferences preferencias = getSharedPreferences("MIS_PREFERENCIAS"  
String correo = preferencias.getString("email", "por_defecto@email.com")
```

# 15.3 GUARDAR PREFERENCIAS

```
SharedPreferences preferencias = getSharedPreferences("MIS_PREFERENCIAS"

SharedPreferences.Editor editor = preferencias.edit();
editor.putString("email", "modificado@email.com");
editor.commit();
```

# 15.4 DIRECTORIO PREFERENCIAS

- Android almacena las preferencias en ficheros XML en una ruta determinada:

```
/data/data/paquete.java.de.la.aplicacion/shared_prefs/nombre_coleccion.xml
```

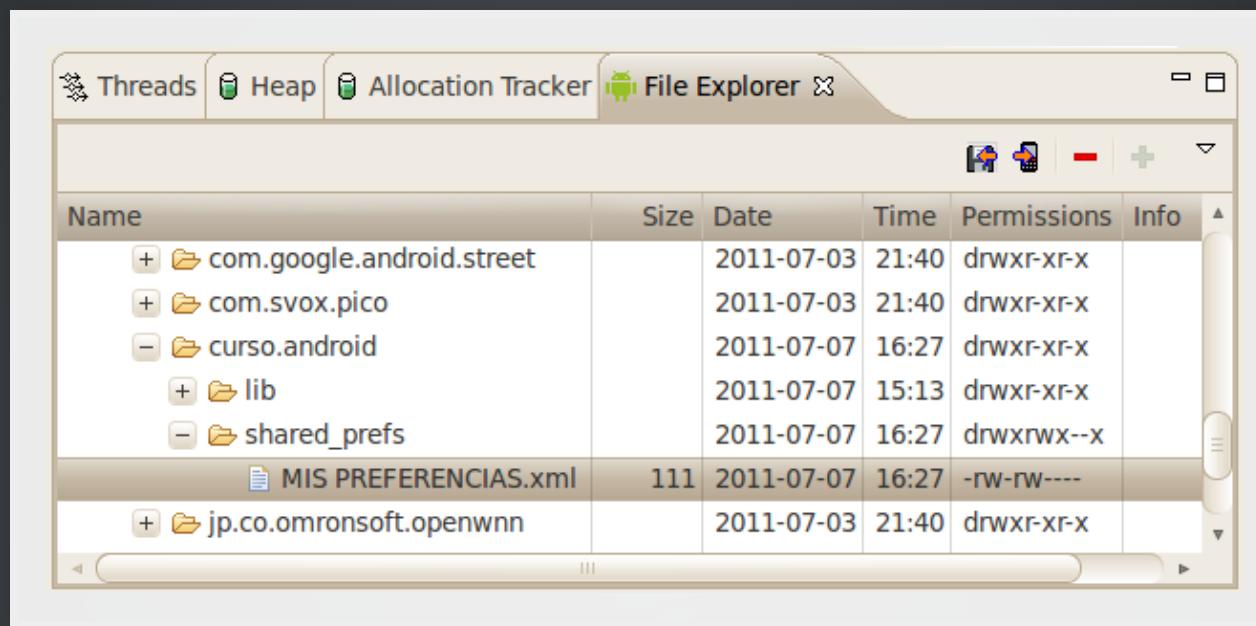
- En nuestro caso:

```
/data/data/curso.android/shared_prefs/MIS_PREFERENCIAS.xml
```

- El XML:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="email">modificado@email.com</string>
</map>
```

# 15.5 PREFERENCIAS EN EL DDMS



Preferencias en el DDMS

# 16 XML

# 16.1 PARSERS XML

- Los más conocidos son:
  - SAX (Simple API for XML):
    - Lee el fichero XML secuencialmente y va generando los eventos para capturar.
    - <http://developer.android.com/reference/javax/xml/parsers/SAXParser.html>
  - Android SAX:
    - Idem que el SAX pero los listener se asocian a etiquetas.
    - <http://developer.android.com/reference/android/xml/XmlPullParser.html>
  - XmlPullParser:
    - Parecido a SAX sólo que es el programador el que lee el fichero.
    - <http://developer.android.com/reference/org/xmlpull/v1/XmlPullParser.html>
  - DOM (Document Object Model):
    - Lee el fichero XML completamente y devuelve un documento jerárquico.
    - <http://developer.android.com/reference/javax/xml/parsers/DocumentBuilder.html>
- No los vamos a estudiar detenidamente, sólo a utilizarlos.

# 16.2 RSS DE EUROPAPRESS

- Nosotros vamos a utilizar la RSS de <http://www.europapress.es/rss/rss.aspx>:

```
<rss version="2.0">
  <channel>
    ...
    <item>
      <title>Título de la noticia 1</title>
      <link>http://link_de_la_noticia_1.es</link>
      <description>Descripción de la noticia 1</description>
      ...
    </item>
    <item>
      <title>Título de la noticia 2</title>
      ...
    </item>
    ...
  </channel>
</rss>
```

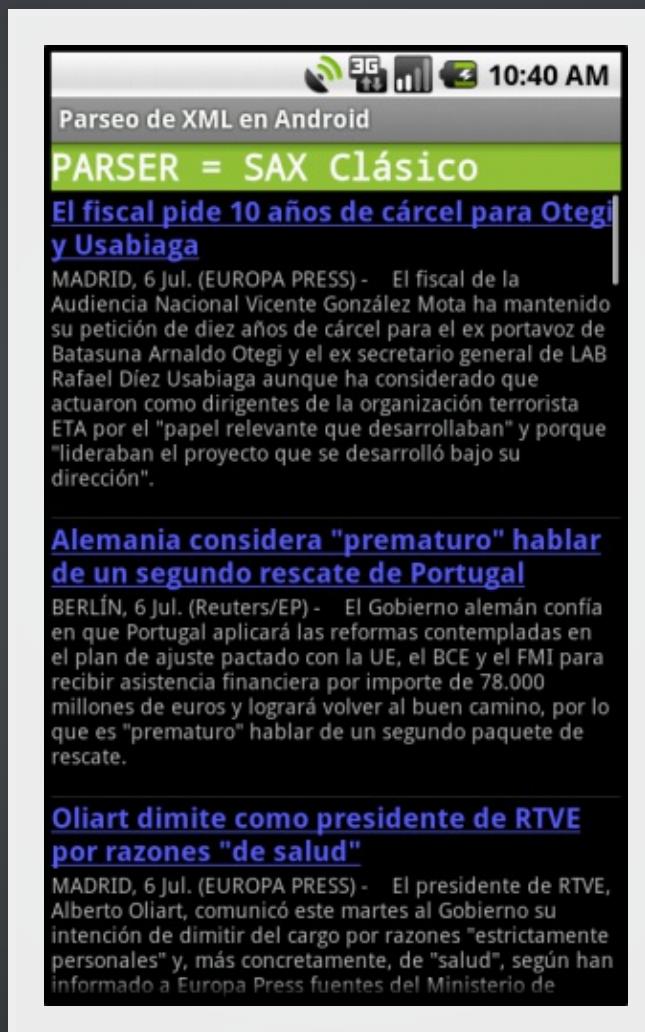
# 16.3 CÓDIGO

```
public class Noticia {  
  
    private String titulo;  
    private String link;  
    private String descripcion;  
  
    // getters y setters  
}
```

```
// donde Xxx será el parser que codifiquemos (SAX, StAX, XmlPullParser o DOM)  
RssParserXxx parserXxx = new RssParserXxx("http://www.europapress.es/rss");  
  
// parseamos el XML del RSS  
List<Noticia> noticias = parserXxx.parse();  
  
// Manipulación del array de noticias (ListView)
```

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

# 16.4 EJEMPLO XML



Ejemplo XML

# 17 JSON

# 17.1 APIs REST

- Hoy en día, cada vez hay más APIs REST públicas:
  - <http://dev.twitter.com/>
  - <http://developers.facebook.com/>
- REST utiliza los métodos definidos en el protocolo HTTP:
  - **GET**: para recuperar un recurso.
  - **POST**: para insertar un recurso.
  - **PUT**: para actualizar un recurso.
  - **DELETE**: para borrar un recurso.

# 17.2 JSON VS XML

- En las APIs REST el intercambio de información se hace sobre todo con JSON porque es más sencillo y más rápido de procesar que el XML.

```
{  
  "menu": {  
    "id": "file",  
    "value": "File",  
    "menuitems": [ "Open", "Close" ]  
  }  
}
```

```
<menu id="file" value="File">  
  <menuitems>  
    <item>Open</item>  
    <item>Close</item>  
  </menuitems>  
</menu>
```

# 17.3 RECUPERAR JSON

```
final HttpResponse response = new DefaultHttpClient().execute(new HttpGet("http://twitter.com/statuses/user_timeline/" + twitterUserName + ".json"));

final int statusCode = response.getStatusLine().getStatusCode();

if (statusCode == 200) {

    final BufferedReader reader = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));

    final StringBuilder json = new StringBuilder();
    String line;
    while ((line = reader.readLine()) != null) {
        json.append(line);
    }

    JSONArray jsonArray = new JSONArray(json.toString());
```

# 18 SQLITE

# 18.1 ¿QUÉ ES SQLITE?

- Es un motor de **bases de datos**:
  - de código libre,
  - de tamaño pequeño,
  - transaccional y
  - que precisa poca configuración.

## 18.2 CREAR O ACTUALIZAR

- Se hace mediante una clase que herede **SQLiteOpenHelper** y sobreesciba 2 métodos:
  - **onCreate()** que se utiliza para crear la base de datos y
  - **onUpgrade()** que se ejecuta si la base de datos está creada y tienen una versión (pasada en el constructor) anterior.

# 18.3 EJECUTAR SENTENCIAS SQL

- Luego se puede llamar a los método **getReadableDatabase()** o **getWritableDatabase()** que devuelven un objeto de tipo **SQLiteDatabase** con los métodos siguientes:
  - **execSQL()** para ejecutar sentencias SQL de actualización (INSERT, UPDATE, DELETE),
  - **rawQuery()** para ejecutar sentencias SQL de consulta (SELECT),
  - **close()** para cerrar la conexión a la base de datos.
- Veremos que hay más métodos de ayuda:
  - **insert()**
  - **update()**
  - **delete()**
  - **query()**

# 18.4 DIRECTORIO

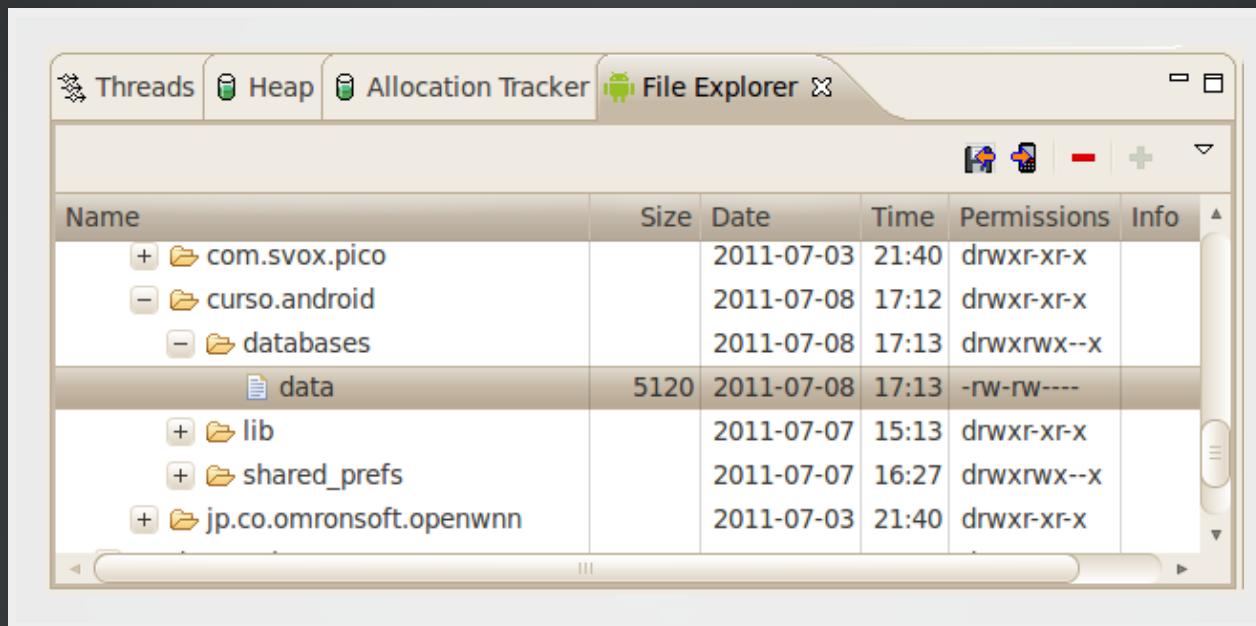
- Android almacena el fichero de la base de datos en una ruta determinada:

```
/data/data/paquete.java.de.la.aplicacion/databases/nombre_base_datos
```

En el caso de nuestro ejemplo, la base de datos se almacenaría por tanto en la ruta siguiente:

```
/data/data/curso.android/databases/data
```

# 18.5 SQLITE EN EL DDMS



SQLite en el DDMS

# 18.6 CONSOLA SQLITE

```
alumno@alumno-laptop:~$ cd android-sdk-linux_x86/platform-tools/
alumno@alumno-laptop:~/android-sdk-linux_x86/platform-tools$ adb -s emul
## sqlite3 /data/data/curso.android/databases/data
SQLite version 3.5.9
Enter ".help" for instructions
sqlite> select * from notes;
1|Primera nota |Texto primera nota
2|Segunda nota |Texto segunda nota
sqlite> .exit
## exit
alumno@alumno-laptop:~/android-sdk-linux_x86/platform-tools$
```

# 18.7 INSERT (NORMAL Y HELPER)

```
db.execSQL("INSERT INTO notes (title, body) VALUES ('Primera nota', 'Texto primera nota')");
```

```
// Indicamos el nombre de la tabla
final String nombreTabla = "notes"

// Creamos el registro a insertar como objeto ContentValues
final ContentValues valores = new ContentValues();
valores.put("title", "Primera nota");
valores.put("body", "Texto primera nota");

// Insertamos el registro en la base de datos
db.insert(nombreTabla, null, valores);
```

# 18.8 UPDATE (HELPER CON WHERE)

```
// Indicamos el nombre de la tabla
final String nombreTabla = "notes"

// Creamos el registro a actualizar como objeto ContentValues
final ContentValues valores = new ContentValues();
valores.put("body", "Texto primera nota");

// Indicamos una clausula where
final String where = "tittle='Primera nota'"

// Actualizamos el registro en la base de datos
db.update(nombreTabla, valores, where);
```

# 18.9 DELETE (HELPER CON PARÁMETROS)

```
// Indicamos el nombre de la tabla
final String nombreTabla = "notes"

// Indicamos una clausula where, esta vez con un parametro
final String where = "tittle=?";

final String[] parametros = new String[]{"Primera nota"};

// Borramos el registro en la base de datos
db.delete(nombreTabla, where, parametros);
```

# 18.10 CONSULTAS

```
final Cursor c = db.rawQuery("SELECT tittle, body FROM notes WHERE title
```

```
final String[] argumentos = new String[] {"Primera nota"};
final Cursor c = db.rawQuery("SELECT tittle, body FROM notes WHERE title
```

```
final String nombreTabla = "notes"
final String[] campos = new String[] {"tittle", "body"};
final String where = "tittle=?";
final String[] argumentos = new String[] {"usul"};
final String goupBy = null;
final String having = null;
final String orderBy = null;

final Cursor c = db.query(nombreTabla, campos, where, argumentos, goupBy
```

# 18.11 RECORRER EL CURSOR

- Métodos:
  - **moveToFirst()**: mueve el puntero del cursor al primer registro devuelto y devuelve TRUE si no hay errores.
  - **moveToNext()**: mueve el puntero del cursor al siguiente registro devuelto y devuelve TRUE si no hay errores.
  - **getXXX(indiceColumna)** donde XXX = String, Double, etc. NOTA: los índices comienzan en 0.
  - **getCount()**: indica el número total de registros devueltos en el cursor.
  - **getColumnNombre(i)**: devuelve el nombre de la columna con índice i.

```
// Nos aseguramos de que existe al menos un registro
if (c.moveToFirst()) {

    // Recorremos el cursor hasta que no haya más registros
    do {
        final String tittle = c.getString(0);
        final String body = c.getString(1);
    } while(c.moveToNext());
}
```

# 19 LOGGING

# 19.1 LOGGING EN ANDROID

- Android nos proporciona también su propio servicio de logging a través de la clase **android.util.Log**.
- Todos los mensajes de log llevarán asociada la siguiente información:
  - **Fecha/Hora**: Indica cuando se generó el mensaje.
  - **Criticidad**: Nivel de gravedad del mensaje.
  - **PID**: Código interno del proceso que ha introducido el mensaje.
  - **Tag**: Etiqueta identificativa del mensaje.
  - **Mensaje**: El texto completo del mensaje.

# 19.2 CRITICIDAD

Criticidad	Método
Error	e(tag, mensaje [, excepcion])
Warning	w(tag, mensaje [, excepcion])
Info	i(tag, mensaje [, excepcion])
Debug	d(tag, mensaje [, excepcion])
Verbose	v(tag, mensaje [, excepcion])

# 19.3 EJEMPLO

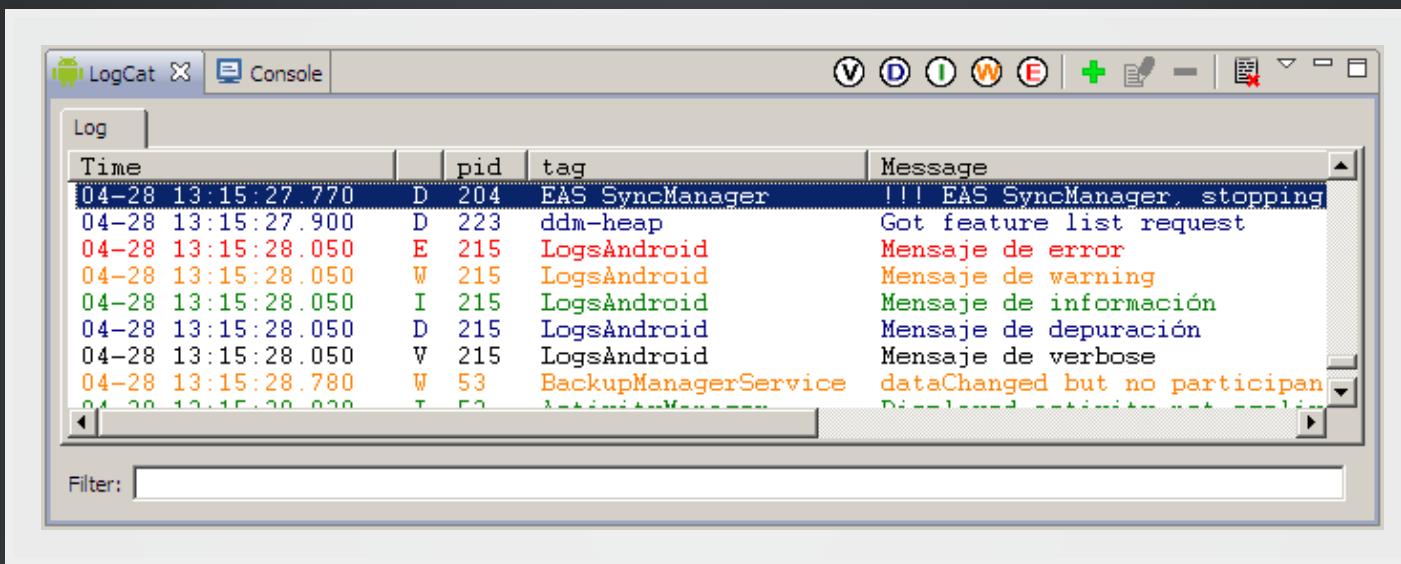
```
public class LogsAndroid extends Activity {

    private static final String LOGTAG = "LogsAndroid";

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

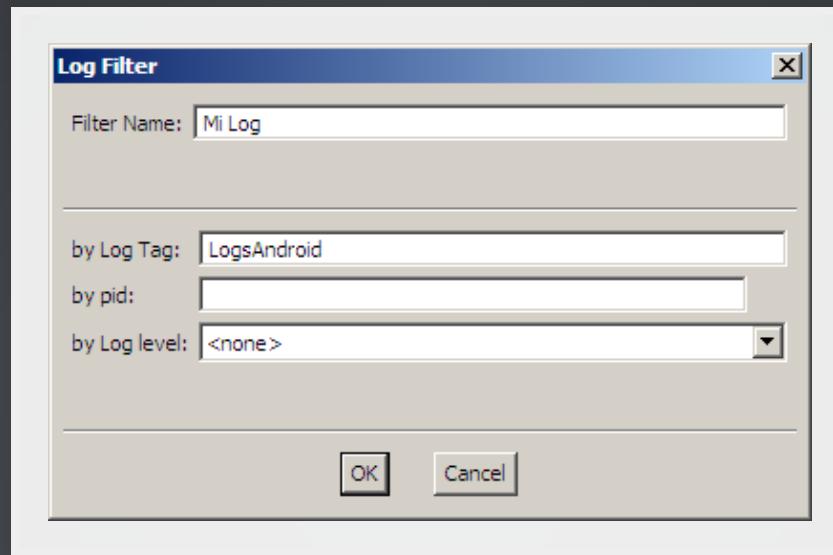
        Log.e(LOGTAG, "Mensaje de error");
        Log.w(LOGTAG, "Mensaje de warning");
        Log.i(LOGTAG, "Mensaje de información");
        Log.d(LOGTAG, "Mensaje de depuración");
        Log.v(LOGTAG, "Mensaje de verbose");
    }
}
```

# 19.4 LOGCAT



LogCat

# 19.5 FILTRADO DEL LOGCAT



## Filtro de LogCat

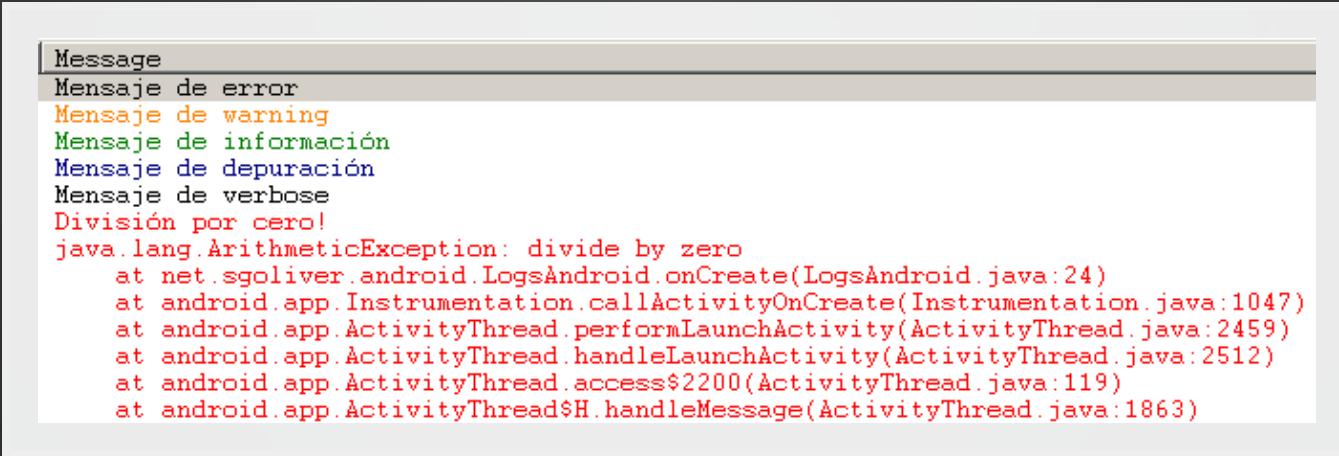
The screenshot shows the Android LogCat window with the 'Mi Log' tab selected. The table displays the following log entries:

Time	pid	tag	Message
04-28 15:19:27.087	E 214	LogsAndroid	Mensaje de error
04-28 15:19:27.087	W 214	LogsAndroid	Mensaje de warning
04-28 15:19:27.087	I 214	LogsAndroid	Mensaje de información
04-28 15:19:27.087	D 214	LogsAndroid	Mensaje de depuración
04-28 15:19:27.087	V 214	LogsAndroid	Mensaje de verbose

# LogCat filtrado

# 19.6 EXCEPCIONES

```
try
{
    int a = 1/0;
}
catch(final Exception e)
{
    Log.e(LOGTAG, "División por cero!", e);
}
```



The screenshot shows the Android LogCat application. At the top, there is a dropdown menu with the following options: Message (which is selected and highlighted in grey), Mensaje de error, Mensaje de warning, Mensaje de información, Mensaje de depuración, Mensaje de verbose, and División por cero!. Below the menu, the log output is displayed in red text, indicating an error. The output includes the error message 'División por cero!', the exception class 'java.lang.ArithmetricException', and the stack trace from 'LogsAndroid.onCreate()' down to 'ActivityThread.access\$2200'.

```
Message
Mensaje de error
Mensaje de warning
Mensaje de información
Mensaje de depuración
Mensaje de verbose
División por cero!
java.lang.ArithmetricException: divide by zero
    at net.sgoliver.android.LogsAndroid.onCreate(LogsAndroid.java:24)
    at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1047)
    at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2459)
    at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2512)
    at android.app.ActivityThread.access$2200(ActivityThread.java:119)
    at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1863)
```

## Excepción en el LogCat

# 20 APP WIDGETS

# 20.1 ¿QUÉ SON LOS APP WIDGETS?

- Son elementos visuales que se colocan en la pantalla principal del dispositivo y muestra información que puede ser actualizada periódicamente.
- Sólo es posible utilizar en su interfaz los siguientes:
  - **Contenedores:** FrameLayout, LinearLayout y RelativeLayout
  - **Controles:** Button, ImageButton, ImageView, TextView, ProgressBar, Chronometer y AnalogClock.

# 20.2 PROPIEDADES (I)

- Se definen en un fichero XML  
(mi\_widget\_provider.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:initialLayout="@layout/mi_widget"
    android:minWidth="146dp"
    android:minHeight="146dp"
    android:label="Mi Widget"
    android:updatePeriodMillis="3600000"
    android:configure="curso.android.WidgetConfigurationActivity" />
```

# 20.3 PROPIEDADES (II)

- Del ejemplo anterior:
  - **initialLayout**: referencia al layout XML que define la interfaz del Widget.
  - **minWidth** y **minHeight**: ancho y alto mínimo, en dp (density-independent pixels).
    - valor = (num-celdas \* 74) – 2
  - **label**: nombre del widget en el menú de selección de Android.
  - **updatePeriodMillis**: frecuencia de actualización del widget, en milisegundos
    - si automático, mínimo cada 30 minutos, pero se puede disminuir con Alarmas.
  - **configure**: clase de configuración, que se lanza la primera vez.

# 20.4 LA CLASE APPWIDGETPROVIDER

- Derivada de BroadcastReceiver, podemos sobreescribir:
  - **onEnabled()**: lanzado cuando se añade al escritorio la primera instancia de un widget.
  - **onUpdate()**: lanzado periodicamente cada vez que se debe actualizar un widget.
  - **onDeleted()**: lanzado cuando se elimina del escritorio una instancia de un widget.
  - **onDisabled()**: lanzado cuando se elimina del escritorio la última instancia de un widget.
  - **onReceive()**: lanzado cuando le llega un Intent.

# 20.5 ANDROIDMANIFEST.XML (I)

```
<activity android:name=".WidgetConfigurationActivity">
    <intent-filter>
        <action android:name="android.apwidget.action.APPWIDGET_CONFIGURE" />
    </intent-filter>
</activity>

<receiver android:name="curso.android.MiWidget" android:label="Mi Widget">
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>
    <intent-filter>
        <action android:name="curso.android.ACTUALIZAR_WIDGET" />
    </intent-filter>
    <meta-data
        android:name="android.appwidget.provider"
        android:resource="@xml/mi_widget_provider" />
</receiver>
```

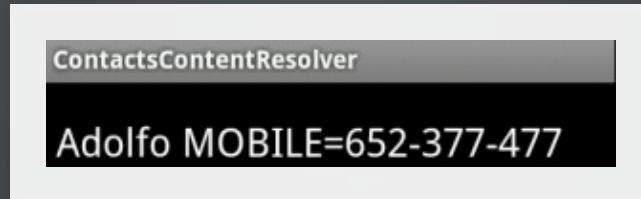
# 20.6 ANDROIDMANIFEST.XML (II)

- Hacemos referencia a la **Activity de configuración**, diciéndole que responda al Intent **APPWIDGET\_CONFIGURE**
- El Widget está dentro de un elemento **<receiver>**.
- Normalmente responderá al Intent **APPWIDGET\_UPDATE**, pero podemos definir nuestros propios Intent.
- En el elemento **<meta-data>** hacemos referencia a su XML de configuración.

# 21 CONTENT RESOLVERS

# 21.1 ¿PARA QUÉ SIRVEN LOS CONTENT RESOLVERS?

- Sirven para recoger datos de los **Content Providers**.
- Ejemplo: Acceder a tu lista de contactos.



Contact Content Resolver

# 21.2 QUERY

- A un Content Resolver hay que pasarle una **query**:

```
final URI uri = Contacts.CONTENT_URI; // especifica para cada ContentProvider
final String[] columns = null // columnas a retornar, null si todas
final String where = Contacts.HAS_PHONE_NUMBER + "=1";
final String[] parametros = null // parametros del where, null si no hay
final String columnSort = null // columna por la cual ordenar, null si ordenar por id

final ContentResolver contentResolver = this.getContentResolver();
final Cursor cursor = contentResolver.query(uri, columns, where, parametros, columnSort);
```

- El **cursor** que devuelve es el mismo explicado con **sqlite**

# 22 LOCALIZACIÓN

# 22.1 LOCALIZACIÓN EN ANDROID

- Mecanismo fácil de programar, pero **poco intuitivo**:
  - Distintos proveedores (GPS, antenas móviles, 3G, Wi-Fi)
  - Distintas propiedades (precisión, altura, velocidad, consumo, dirección)
  - No siempre disponibles, activos y permitidos.
  - No son síncronos.

## 22.2 LISTA DE PROVEORES DISPONIBLES

```
final LocationManager locationManager = (LocationManager) this.getSystemService(Context.LOCATION_SERVICE);
final List<String> locationProviders = locationManager.getAllProviders()
```

- Retorna el nombre de todos los provedores de localización disponibles en el dispositivo, estén activos o no y tenga o no permisos la aplicación para utilizarlos (permisos del AndroidManifest.xml)

## 22.3 RECUPERAR UN PROVEDOR

```
final LocationManager locationManager = (LocationManager) this.getSystemService(Context.LOCATION_SERVICE);
final LocationProvider locationProvider = locationManager.getLocatio
```

## 22.4 PROPIEDADES DE UN PROVEDOR

- La clase **LocationProvider** nos facilita, entre otros, los siguientes métodos:
  - **getAccuracy()**: Tipo de precisión del provedor.
    - Criteria.ACCURACY\_COARSE: precisión aproximada.
    - Criteria.ACCURACY\_FINE: precisión alta.
  - **getName()**: Nombre del provedor.
  - **getPowerRequirement()**: Consumo energético del provedor.
    - Criteria.POWER\_LOW: bajo consumo energético.
    - Criteria.POWER\_MEDIUM: consumo energético medio.
  - **requiresCell()**: Indica se requiere conexión

**requiresCell()**: Indica se requiere conexión telefónica (antenas móviles)

- **requiresNetwork()**: Indica se requiere conexión a Internet.
- **requiresSatellite()**: Indica se requiere conexión GPS.
- **supportsAltitude()**: Indica si es capaz de mostar la altitud.
- **supportsBearing()**: Indica si es capaz de mostar la dirección.
- **supportsSpeed()**: Indica si es capaz de mostar la velocidad.

# 22.5 MEJOR PROVEDOR PARA PARA MI APLICACIÓN

```
final Criteria criteria = new Criteria();  
  
// Indicamos los requisitos de nuestra aplicación  
criteria.setAccuracy(Criteria.ACCURACY_COARSE);  
criteria.setAltitudeRequired(false);  
criteria.setBearingRequired(true);  
criteria.setPowerRequirement(Criteria.POWER_LOW);  
criteria.setSpeedRequired(false);  
  
// Indicamos si queremos que además el provedor esté activo  
final boolean enabledOnly = false;  
  
// Mejor proveedor para los requisitos indicados  
final String bestLocationProvider = locationManager.getBestProvider(crit  
  
// Lista de proveedores que cumplen los requisitos indicados  
final List<String> criteriaLocationProviders = locationManager.getProvid
```

# 22.6 ¿ESTÁ ACTIVO EL PROVEDOR?

```
final String providerName = LocationManager.GPS_PROVIDER;

if ( ! locationManager.isProviderEnabled(providerName) ) {

    mostrarAdvertenciaProviderDesactivado(providerName);
}
```

## 22.7 ÚLTIMA LOCALIZACIÓN CONOCIDA

- Última localización conocida (hace una hora, un día, un mes o un año) para un provider concreto:

```
final Location location = locationManager.getLastKnownLocation(providerN
```

# 22.8 SUBSCRIBIRSE AL PROVEDOR

```
locationManager.requestLocationUpdates(
    providerName, // nombre del provedor de localización
    timeInMillis, // tiempo mínimo, en milisegundos, entre actualizaciones
    distanceInMeters, // distancia mínima, en metros, entre actualizaciones
    new LocalitationListener() {

        public void onLocationChanged(final Location location) {
            mostrarPosicion(location); }

        public void onProviderDisabled(final String providerName) {
            mostrarAdvertenciaProviderDesactivado(providerName); }

        public void onProviderEnabled(final String providerName) {
            mostrarAdvertenciaProviderActivado(providerName); }

        public void onStatusChanged(final String provider, final int status,
            // status puede tomar los valores OUT_OF_SERVICE, TEMPORARILY_UNAVAILABLE
            // y永続的amente deshabilitado si el proveedor no está disponible
            // o no se ha iniciado
            ...
        }
    }
}
```

## 22.9 BORRAR LA SUBSCRIPCIÓN AL PROVEDOR

```
locationManager.removeUpdates(providerName);
```

# 22.10 MOSTRAR LA LOCALIZACIÓN

- La clase **Location** nos facilita, entre otros, los siguientes métodos:
  - **distanceTo(final Location dest)**: devuelve la distancia en metros entre el punto que indica la propia Location y la pasada por parámetro.
  - **getAccuracy()**: devuelve la precisión en metros.
  - **getAltitude()**: devuelve la altitud en metros.
  - **getBearing()**: devuelve la dirección en grados Este desde el Norte verdadero.
  - **getLatitude()**: devuelve la latitud.
  - **getLongitude()**: devuelve la longitud.
  - **getProvider()**: nombre del proveedor de localización.
  - **getSpeed()**: velocidad medida en metros por segundo.

# 22.11 LOCALIZACIÓN EN EL EMULADOR (I)

- Se puede cambiar los datos de localización del emulador desde el **DDMS** en la vista **Emulator Control** dentro de **Location Controls**.
- Se puede cambiar, en sus pestañas correspondientes:
  - de forma **Manual**, modificando la longitud y la latitud en cajas de texto,
  - de forma automática, cargando archivos de datos geográficos
    - **GMX**
    - **KML**.

# 22.12 LOCALIZACIÓN EN EL EMULADOR (II)

The image displays two windows of the "Emulator Control" application side-by-side.

**Left Window (Telephony Controls):**

- Telephony Status:**
  - Voice: home
  - Speed: Full
- Data:** home
- Latency:** None
- Telephony Actions:**
  - Incoming number:
  - Voice
  - SMS
  - Message:
  - Call** **Hang Up**
- Location Controls:**
  - Manual** **GPX** **KML** (selected)
  - Decimal
  - Sexagesimal
  - Longitude: -122,084095
  - Latitude: 37,422006
  - Send**

**Right Window (Location Controls):**

- SMS**
- Message:**
- Call** **Hang Up**
- Location Controls:**
  - Manual** **GPX** **KML** (selected)
  - Load KML...**
  - | Name    | Longitude  | Latitude |
|---------|------------|----------|
| Punto 1 | -11,999999 | 7,000001 |
| Punto 2 | -11,999998 | 7,000002 |
| Punto 3 | -11,999997 | 7,000003 |
| Punto 4 | -11,999996 | 7,000004 |
| Punto 5 | -11,999995 | 7,000005 |
| Punto 6 | -11,999994 | 7,000006 |
| Punto 7 | -11,999993 | 7,000007 |
| Punto 8 | -11,999992 | 7,000008 |
  - Speed: 1X**

# Localización en el emulador

**23 MAPAS**

## 23.1 REQUISITOS PREVIOS

- Instalar **Google APIs** .
- Crear un AVD que utilice este paquete como target.
- Crear proyecto que utilice este paquete como target.

## 23.2 API KEY

- Para poder utilizar la API de Google Maps se requiere una **API Key**.
- Estará asociada al certificado con el que firmamos nuestra aplicación.
- Si cambiamos el certificado (publicación en el Market) habrá que cambiarla.
- Para instalar aplicaciones en el emulador, se usa certificado automático, aún así hay que solicitar una clave asociada.

# 23.3 SOLICITAR CLAVE

- Eclipse > Windows > Preferences > Android > Build
  - copiar la ruta de Default Debug Keystore
- Ejecutar comando (tanto Windows como Linux):

```
<path-jre>/bin/keytool -list  
  -alias androiddebugkey  
  -keystore "<path-default-debug-keystore>"  
  -storepass android  
  -keypass android
```

- Obtenemos Huella digital de certificado (MD5)
  - Introducirla en  
<http://code.google.com/android/maps-api-signup.html>

# 23.4 MAPVIEW

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

    <com.google.android.maps.MapView
        android:id="@+id/mapa"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:apiKey="ClAv3-0Bt3n1Da-3n-3L-pUnT0-aNt3rI0r"
        android:clickable="true" />

</LinearLayout>
```

# 23.5 MAPACTIVITY

```
public class MapasActivity extends MapActivity {  
  
    public void onCreate(Bundle savedInstanceState) {  
        ...  
        // Obtenemos una referencia al control MapView  
        final MapView mapa = (MapView) findViewById(R.id.mapa);  
        ...  
        // Añadimos la capa de marcadores (ver después)  
        mapa.getOverlays().add(new OverlayMapa());  
        mapa.postInvalidate();  
        ...  
    }  
  
    protected boolean isRouteDisplayed() { return false; }  
  
    public boolean onTap(GeoPoint point, MapView mapView) {  
        final String[] resultado = new String[1];  
        new Thread("Buscando").start();  
        resultado[0] = resultado[0].trim();  
        if (resultado[0].length() > 0) {  
            Intent i = new Intent(getApplicationContext(), Resultado.class);  
            i.putExtra("resultado", resultado[0]);  
            startActivity(i);  
        }  
        return true;  
    }  
}
```

# 23.6 ANDROIDMANIFEST.XML.

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

<application android:icon="@drawable/icon" android:label="@string/app_name">

    <uses-library android:name="com.google.android.maps" />

    <activity android:name=".AndroidMapas" android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

</application>
```

# 23.7 MÉTODOS DE LA CLASE MAPVIEW

- Disponemos de los siguientes métodos
  - **setBuiltInZoomControls(boolean)**: muestra los controles de zoom.
  - **setSatellite(boolean)**: muestra la vista tipo satelite.
  - **setStreetView(boolean)**: muestra la vita tipo StreetView.
  - **getMapCenter()**: retorna el GeoPoint central.
  - **setCenter(GeoPoint)**: establece el centro del mapa.
  - **animateTo(GeoPoint)**: mueve el mapa hasta el munto indicado.
  - **getZoomLevel()**: nivel de zoom (1 menos de talle - 21 más detalle)

- **setZomm()**: establece un nivel de zoom.
- **zoomIn()**: aumenta el zoom.
- **zoomOut()**: disminuye el zoom.
- **scrollBy(numPixelHorizontal, numPixelVertical)**: desplaza el mapa.
- **getProjection()**: retorna un objeto de tipo Projection.

## 23.8 LA CLASE PROJECTION

- La clase MapView retorna un objeto de tipo **Projection**, que tendrá en cuenta la posición central del mapa y su nivel de zoom.
- La clase Projection es capaz de convertir coordenadas expresadas en grados en píxeles.

# 23.9 CAPAS

```
public class OverlayMapa extends Overlay {  
  
    public void draw(Canvas canvas, MapView mapView, boolean shadow) {  
  
        Point centro = new Point();  
        mapView.getProjection().toPixels(mapView.getMapCenter(), centro);  
  
        // Definimos el pincel de dibujo  
        Paint paint = new Paint();  
        paint.setColor(Color.BLUE);  
  
        // Marca Ejemplo 1: Círculo y Texto  
        canvas.drawCircle(centro.x, centro.y, 5, paint);  
        canvas.drawText("Texto", centro.x+10, centro.y+5, paint);  
  
        // Marca Ejemplo 2: Bitmap  
        Bitmap bitMap = BitmapFactory.decodeResource(mapView.getResources(),  
            canvas.drawBitmap(bitmap, centro.x - bitmap.getWidth() / 2, centro.y - bitmap.getHeight() / 2, paint));  
    }  
}
```