Android

Victor Herrero Cazurro

Table of Contents

1. Instalación

Los requisitos para la instalacion, son

- JDK
- SDK Android
- Android Studio

Se distribuye un Bundle con Android Studio y el SDK

El SDK, incluye una herramienta de descarga del API **SDK Manager**, que permite gestionar la descarga de las distintas version de Android, asi como imagenes para emuladores y herramientas de desarrollo.

2. Introducción

Android es un sistema operativo basado en Linux diseñado principalmente para dispositivos móviles con pantalla táctil, como smartphones o tablets, en las ultimas versiones se ha dado soporte a otro tipo de terminales como televisiones o relojes (wearables).

2.1. Historia

Julio 2005

- Google adquiere Android, Inc.
- Pequeña empresa que desarrolla software para móviles (hasta entonces una gran desconocida)

Noviembre 2007

- Nace la Open Handset Alliance, consorcio de empresas unidas con el objetivo de desarrollar estándares abiertos para móviles Texas Instruments, Broadcom co., Google, HTC, Intel, LG, Marvel Tech., Motorola, Nvidia, Qualcomm, Samsumg Electonics, Sprint Nextel, T-Mobile
- Se anuncia su primer producto, Android, plataforma para móviles construida sobre el kernel de Linux 2.6

Septiembre 2008

- HTC Dream. Primer móvil con Android.

Abril 2009

- Lanzamiento de Cupcake (Android 1.5). Primera gran actualización de Android.

Septiembre 2009

- Lanzamiento de Donut (1.6)

Octubre 2009

- Lanzamiento Eclair (2.0)

Noviembre 2009

- Motorola Droid. Consigue vender 1.05 millones de unidades en 74 días, superando el record establecido por el iPhone de Apple.

Diciembre 2009

- 16.000 aplicaciones en el Market, 60% gratuitas, 30% de pago aprox.

Enero 2010

- Google Nexus One (HTC). Malas cifras de ventas, apenas 135 mil unidades en 74 días

Febrero 2010

- 60.000 teléfonos con Android vendidos al día.

Junio 2010

- Lanzamiento Froyo (2.2).

Diciembre 2010

- Lanzamiento Gingerbread (2.3).
- Nexus S. Vende mas de 400,000 unidades en el primer cuatrimestre.

Febrero 2011

- Lanzamiento Honecomb (3.0).

Mayo 2011

- Lanzamiento Ice Cream Sandwich.

Junio 2012

- Nexus 7

Julio 2012

- Lanzamiento Jelly Bean (4.1)
- Samsung Galaxy III. Segundo smartphone mas vendido solo superado por Iphone.

Noviembre 2012

- Nexus 4 y Nexus 10.

Octubre 2013

- Lanzamiento Kit Kat (4.4)

Noviembre 2013

- Nexus 5

2.2. Caracteristicas

- Framework de aplicación que habilita la reutilización y reemplazo de componentes.

Máquina virtual *Dalvik optimizada para móviles (se cambia a partir de KitKat por **ART**, que dota a la ejecución de mayor rendimiento).

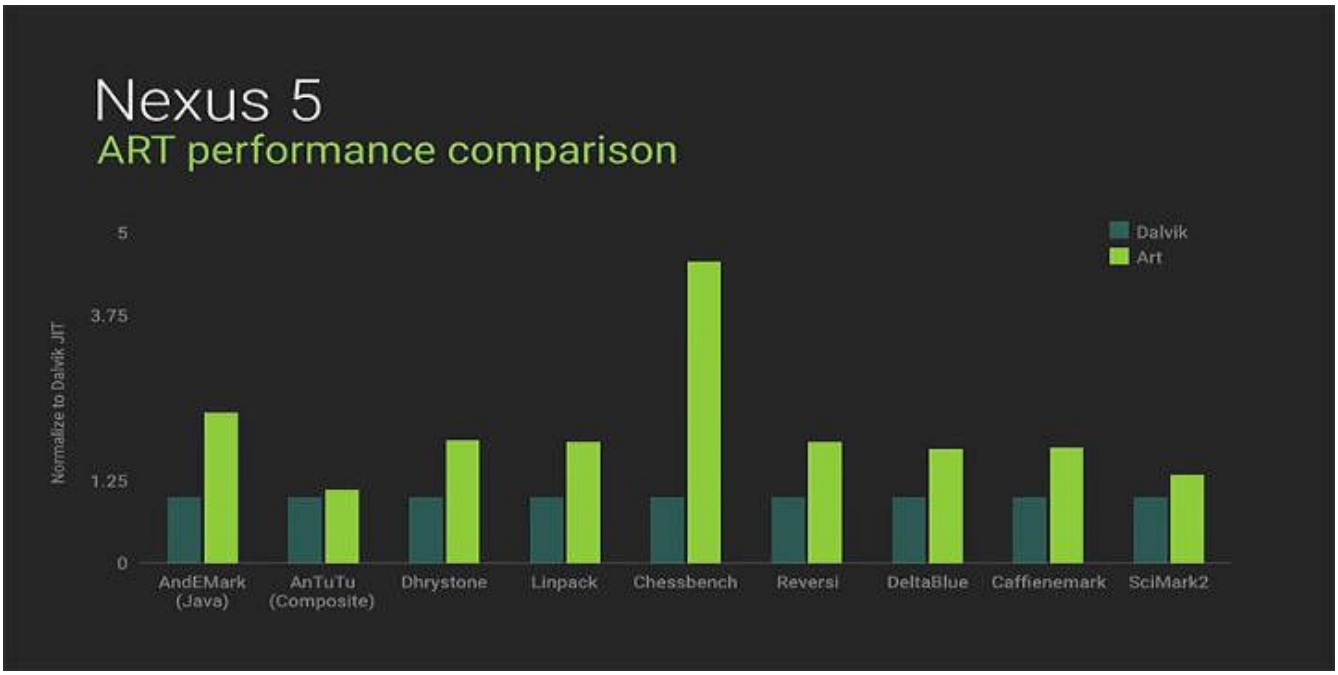


Figure 1. Comparativa de tiempo de arranque en Nexus 5 con Dalvik y ART

La principal diferencia entre Dalvik y ART es en que momento se realiza la compilación, mientras en Dalvik es en tiempo real, en ART es en el momento de la instalación, por lo que ART no lastra la ejecución con la compilación.

- Navegador integrado basado en WebKit.
- Gráficos optimizados por una librería gráfica 2D propia. Gráficos 3D basados en la especificación OpenGL ES 1.0.
- SQLite para almacenamiento de datos estructurados.
- Soporte para gran variedad de archivos multimedia (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- Telefonía GSM
- Bluetooth, EDGE, 3G y WiFi (4G, WiMAX,…)
- Cámara, GPS, compás, acelerómetro,…
- Entorno de desarrollo completo basado en **Intellij** incluyendo emulador, herramientas de depuración, profiling de memoria y rendimiento.

2.3. Versiones

Las versiones de Android reciben el nombre de postres en inglés, en cada versión el postre elegido empieza por una letra distinta siguiendo un orden alfabético.

- Septiembre 2008 – Android 1.0 - Apple Pie - API Level 1
- Febrero 2009 – Android 1.1 - Banana Bread - API Level 2
- Abril 2009 – Android 1.5 – Cupcake - API Level 3
- Septiembre 2009 – Android 1.6 – Donut - API Level 4
- Noviembre 2009 – Android 2.0 – Éclair - API Level 5
- Diciembre 2009 – Android 2.0.1 - Éclair - API Level 6
- Enero 2009 - Android 2.1 - Éclair - API Level 7
- Mayo 2010 – Android 2.2 - Froyo - API Level 8
- Diciembre 2010 – Android 2.3 - Gingerbread - API Level 9 y 10
- Mayo 2011 - Android. 3.0 - Honeycomb - API Level 11, 12 y 13
- Octubre 2011 - Android 4.0 - Ice Cream Sandwich - API Level 14 y 15
- Julio 2012 – Android 4.1 – Jelly Bean - API Level 16, 17 y 18
- Octubre 2013 – Android 4.4 – Kit Kat - API Level 19 y 20
- - Android 5 - Lolipop - API Level 21 y 22
- - Android 6 - Marshmallow - API Level 23
- - Android 7 - N - API Level N

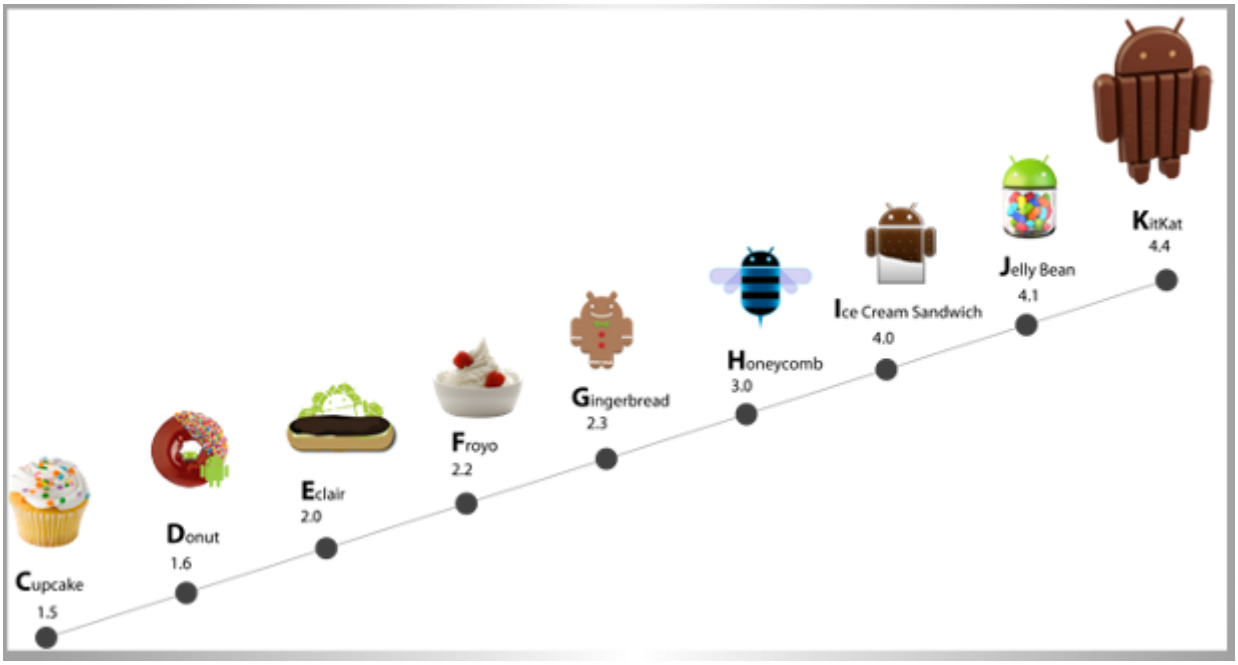


Figure 2. Versiones de Android 2013

2.3.1. Android 1.0 (HTC Dream)

Android Market Nevegador Web (sin HTML5). Camara (sin cambio de resolución y otras opciones) Carpetas para agrupar iconos en el escritorio. Acceso a servidores de correo POP3, IMAP4, SMTP. Sincronización con GMAIL, Google Contact, Google Calendar. Google Maps con Latitude y Street View Soporte GPS. Google Talk SMS y MMS. Reproductor multimedia (sin soporte de video y estéreo por Bluetooth) Notificaciones en barra de estado. Alertas por timbre, led y vibración. Marcación por voz. Fondo de escritorio. Soporte Wi-Fi y Bluetooth.

2.3.2. Android 1.1 (HTC Dream)

Detalles y reseñas disponibles cuando un usuario busca en los mapas. Habilidad de mostrar/esconder el marcador en las llamadas. Posibilidad de guardar archivos adjuntos en los mensajes.

2.3.3. Android 1.5

Soporte para teclados virtuales de terceros con predicción de texto y diccionario de usuarios para palabras personalizadas. Soporte para Widgets. Grabación y reproducción en formatos MPEG-4 y 3GP. Auto-sincronización y soporte para Bluetooth estéreo añadido. Características de Copiar y pegar agregadas al navegador web. Fotos de los usuarios son mostradas en los contactos. Marcas de fecha/hora mostradas para eventos en registro de llamadas y acceso con un toque a la tarjeta de un contacto desde un evento del registro de llamadas. Pantallas de transiciones animadas. Agregada opción de auto-rotación. Agregada la animación de inicio por defecto actual. Habilidad de subir vídeos a YouTube. Habilidad de subir fotos a Picasa.

2.3.4. Android 1.6

Mejora en la búsqueda por entrada de texto y voz para incluir historial de favoritos, contactos y la web. Habilidad de los desarrolladores de incluir su contenido en los resultados de búsqueda. Motor multi-lenguaje de Síntesis de habla para permitir a cualquier aplicación de Android "hablar" una cadena de texto. Búsqueda facilitada y habilidad para ver capturas de las aplicaciones en el Android Market(Google Play). Galería, cámara y videocámara con mejor integración, con rápido acceso a la cámara. La galería ahora permite a los usuarios seleccionar varias fotos para eliminarlas. Soporte para resoluciones de pantalla WVGA. Mejoras de velocidad en búsqueda y aplicaciones de cámara.

2.3.5. Android 2.0 (Motorola Droid)

Múltiples cuentas al dispositivo para sincronización de correo y contactos. Soporte intercambio de correo, con bandeja combinada para buscar correo desde múltiples cuentas en la página. Soporte Bluetooth 2.1. Habilidad para tocar un foto de un contacto y seleccionar llamar, enviar SMS o correo a la persona. Habilidad para en todos los mensajes SMS y MMS guardados, con eliminación de mensajes más antiguos en una conversación automáticamente cuando un límite definido se ha alcanzado. Nuevas características para la cámara, incluyendo soporte de flash, zoom digital, modo escena, balance de blancos, efecto de colores y enfoque macro. Mejorada velocidad de tipeo en el teclado virtual, con diccionario inteligente que aprende el uso de palabras e incluye nombres de contactos como sugerencias. Renovada interfaz de usuario del navegador con imágenes en

miniatura de marcador, zoom de toque-doble y soporte para HTML5. Vista agenda del calendario mejorada, que muestra el estado asistiendo a cada invitado, y la capacidad de invitar a nuevos invitados a los eventos. Optimización en velocidad de hardware y GUI renovada. Soporte para más tamaños de pantalla y resoluciones, con mejor ratio de contraste. Mejorado Google Maps 3.1.2. Clase MotionEvent mejorada para rastrear eventos multi-touch. Adición de fondos de pantalla animados, permitiendo la animación de imágenes de fondo de la pantalla inicio para mostrar movimiento. Framework de gestos ampliado y una nueva herramienta de desarrollo GestureBuilder.

2.3.6. Android 2.2

Optimizaciones en velocidad, memoria y rendimiento Mejoras adicionales de rendimiento de aplicación, implementadas mediante compilación Just-in-time. Integración del motor de JavaScript V8 de Chrome en el navegador. Soporte para el servicio Android Cloud to Device Messaging (C2DM), habilitando notificaciones push Soporte para Microsoft Exchange mejorado, incluyendo políticas de seguridad, auto-descubrimiento, consulta a la Global Access List (GAL), sincronización de calendario, y borrado remoto. Mejoras en la aplicación del lanzador con accesos directos de las aplicaciones teléfono y navegador web Funcionalidad de anclaje de red por USB y Wi-Fi hotspot Agregada opción para deshabilitar acceso de datos sobre red móvil Actualizada la aplicación Market con características de grupo y actualizaciones automáticas Cambio rápido entre múltiples lenguajes de teclado y diccionario Intercambio de contactos por Bluetooth Soporte para contraseñas numéricas y alfanuméricas Soporte para subida de archivos en la aplicación del navegador Soporte para instalación de aplicaciones en la memoria expandible Soporte para Adobe Flash Soporte para pantallas de alto número de PPI (320 ppi), como 4" 720p Galería permite a los usuarios ver pilas de imágenes mediante un gesto de zoom

2.3.7. Android 2.3 (Nexus S)

Actualizado el diseño de la interfaz de usuario con incrementos en velocidad y simpleza. Soporte para tamaños y resoluciones de pantalla extra-grandes (WXGA y mayores). Soporte nativo para SIP y telefonía por internet VoIP. Entrada de texto del teclado virtual más rápida e intuitiva, con mejoras en precisión, texto sugerido y entrada por voz. Mejoras en la funcionalidad de copiar/pegar, permitiendo a los usuarios seleccionar una palabra al presionar-mantener, copiar y pegar. Soporte para Near Field Communication (NFC), permitiendo al usuario leer la etiqueta NFC incrustada en un póster, sticker o anuncio publicitario. Nuevos efectos de audio tales como reverberación, ecualizador, virtualización de audífonos y aumento de bajos. Nuevo gestor de descargas, que da a los usuarios fácil acceso a cualquier archivo descargado del navegador, correo electrónico u otra aplicación. Soporte para múltiples cámaras en el dispositivo, incluyendo cámara frontal-facial, si está disponible. Soporte para reproducción de video por WebM/VP8, encoding de audio por AAC. Mejoras en la administración de la energía, con un mayor rol activo en aplicaciones de administración que se mantienen activas en el dispositivo por mucho tiempo. Mejorado soporte para el desarrollo de código nativo. Mejoras en audio, gráficos y entrada para desarrolladores de juegos. Recolector basura concurrente para incrementar el rendimiento. Soporte nativo para más sensores (tales como giroscopio y barómetro).

2.3.8. Android 3.0 (Motorola Xoom)

Soporte optimizado para tablets, con una nueva y "virtual" interfaz de usuario holográfica. Agregada barra de sistema, con características de acceso rápido a notificaciones, estados y botones

de navegación suavizados, disponible en la parte inferior de la pantalla. Añadida barra de acción (Action Bar en inglés), entregando acceso a opciones contextuales, navegación, widgets u otros tipos de contenido en la parte superior de la pantalla. Multitarea simplificada – tocando Aplicaciones recientes en la barra del sistema permite a los usuarios ver instantáneas de las tareas en curso y saltar rápidamente de una aplicación a otra. Teclado rediseñado, permitiendo una escritura rápida, eficiente y acertada en pantallas de gran tamaño. Interfaz simplificada y más intuitiva para copiar/pegar. Las pestañas múltiples reemplazan las ventanas abiertas en el navegador web, además de la característica de auto completado texto y un nuevo modo de "incógnito" permitiendo la navegación de forma anónima. Acceso rápido a las características de la cámara como la exposición, foco, flash, zoom, cámara facial-frontal, temporizador u otras. Habilidad para ver álbumes y otras colecciones de fotos en modo pantalla completa en galería, con un fácil acceso a vistas previas de las fotografías. Nueva interfaz de contactos de dos paneles y desplazamiento rápido para permitir a los usuarios organizar y reconocer contactos fácilmente. Nueva interfaz de correo de dos paneles para hacer la visualización y organización de mensajes más eficiente, permitiendo a los usuarios seleccionar uno o más mensajes. Soporte para videochat usando Google Talk. Aceleración de hardware. Soporte para microprocesadores multi-núcleo. Habilidad para encriptar todos los datos del usuario. Mejoras en el uso de HTTPS con Server Name Indication (SNI).

2.3.9. Android 3.1

Refinamiento a la interfaz de usuario. Conectividad para accesorios USB. Lista expandida de aplicaciones recientes. Widgets redimensionables en la pantalla de inicio. Soporte para teclados externos y dispositivos punteros. Soporte para joysticks y gamepads. Soporte para reproducción de audio FLAC57 Bloqueo de Wi-Fi de alto rendimiento, manteniendo conexiones Wi-Fi de alto rendimiento cuando la pantalla del dispositivo está apagada. Soporte para proxy HTTP para cada punto de acceso Wi-Fi conectado.

2.3.10. Android 4.0 (Galaxy Nexus)

Botones suaves Android 3.x están ahora disponibles para usar en los teléfonos móviles. Separación de widgets en una nueva pestaña, listados de forma similar a las aplicaciones. Facilidad para crear carpetas, con estilo de arrastrar y soltar. Lanzador personalizable. Buzón de voz mejorado con la opción de acelerar o retrasar los mensajes del buzón de voz. Funcionalidad de pinch-to-zoom en el calendario. Captura de pantalla integrada (manteniendo presionado los botones de bloqueo y de bajar volumen). Corrector ortográfico del teclado mejorado. Habilidad de acceder a aplicaciones directamente desde la pantalla de bloqueo. Funcionalidad copiar-pegar mejorada. Mejor integración de voz y dictado de texto en tiempo real continuo. Desbloqueo facial, característica que permite a los usuarios desbloquear los equipos usando software de reconocimiento facial. Nuevo navegador web con pestañas bajo la marca de Google Chrome, permitiendo hasta 15 pestañas. Sincronización automática del navegador con los marcadores de Chrome del usuario. Nueva tipografía para la interfaz de usuario, Roboto. Sección para el uso de datos dentro de la configuración que permite al usuario poner avisos cuando se acerca a cierto límite de uso, y desactivar los datos cuando se ha excedido dicho límite. Capacidad para cerrar aplicaciones que están usando datos en segundo plano. Aplicación de la cámara mejorada sin retardo en el obturador, ajustes para el time lapse, modo panorámico y la posibilidad de hacer zoom durante la grabación. Editor de fotos integrado. Nuevo diseño de la galería, organizada por persona y localización. Aceleración por hardware de la interfaz de usuario Wi-Fi Direct Grabación de vídeo a 1080P para dispositivos con Android de serie. Android VPN Framework (AVF).

2.3.11. Android 4.3

Soporte para Bluetooth de Baja Energía OpenGL ES 3.0 Modo de perfiles con acceso restringido DRM APIs de mayor calidad Mejora en la escritura Cambio de usuarios más rápida Soporte para Hebreo y Árabe Locación de WiFi en segundo plano Añadido el soporte para 5 idiomas más Opciones para creadores de Apps Mejoras en la seguridad

2.3.12. Android 4.4 (Nexus 5)

Barra de botones, transparente. Desaparece la barra de notificaciones y los botones generales en una aplicación a pantalla completa. El buscador de google se integra con los contactos, de tal forma que se buscará no solo en los contactos, sino en entradas de Google. Hangouts integrado con SMS. Optimizado para dispositivos con poca RAM (a partir de 512Mb) y poco procesador. Tarjetas virtuales con NFC, puede permitir eliminar los soportes físicos para las tarjetas de fidelización de los comercios. Terminal como mando a distancia con Wi-Fi Miracast. Soporte para Bluetooth MAP, para sincronizar información con otros dispositivos (coche). Optimización de sensores de bajo consumo. Añade soporte para nuevos sensores, contador y detector de pasos. Capturas de videos a través de ADB.

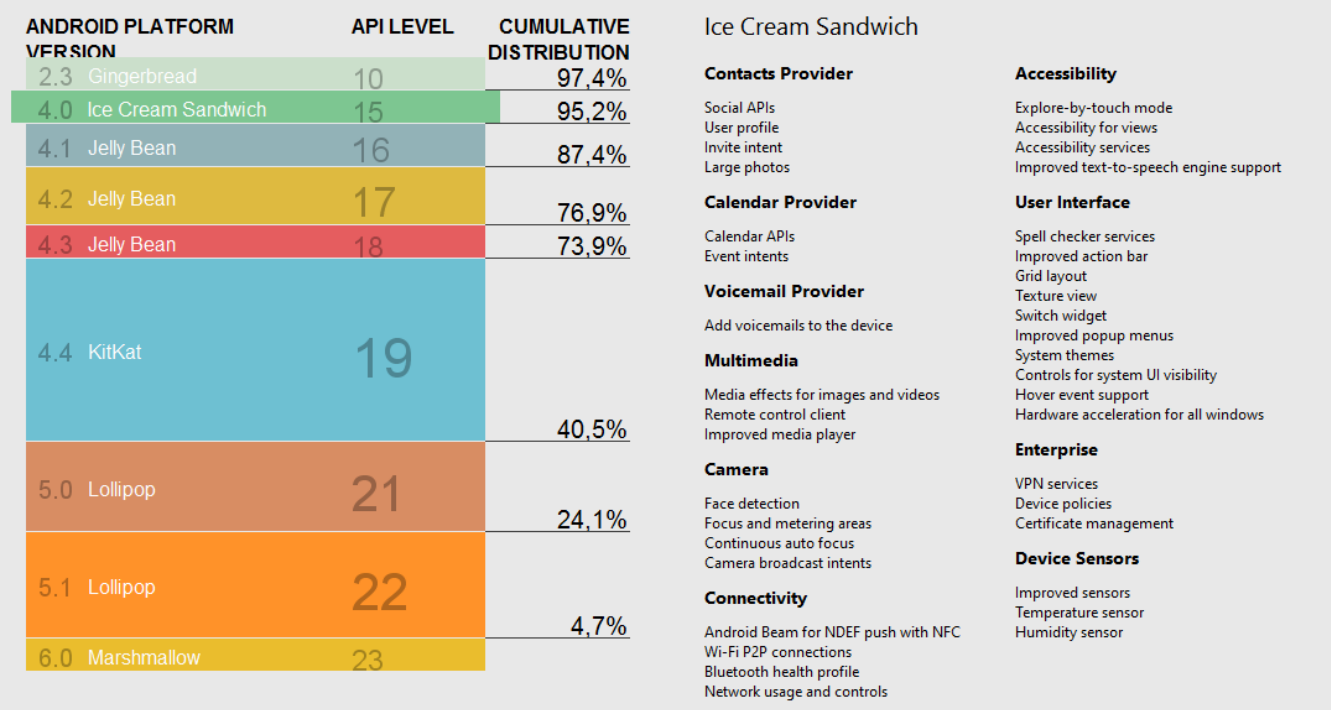


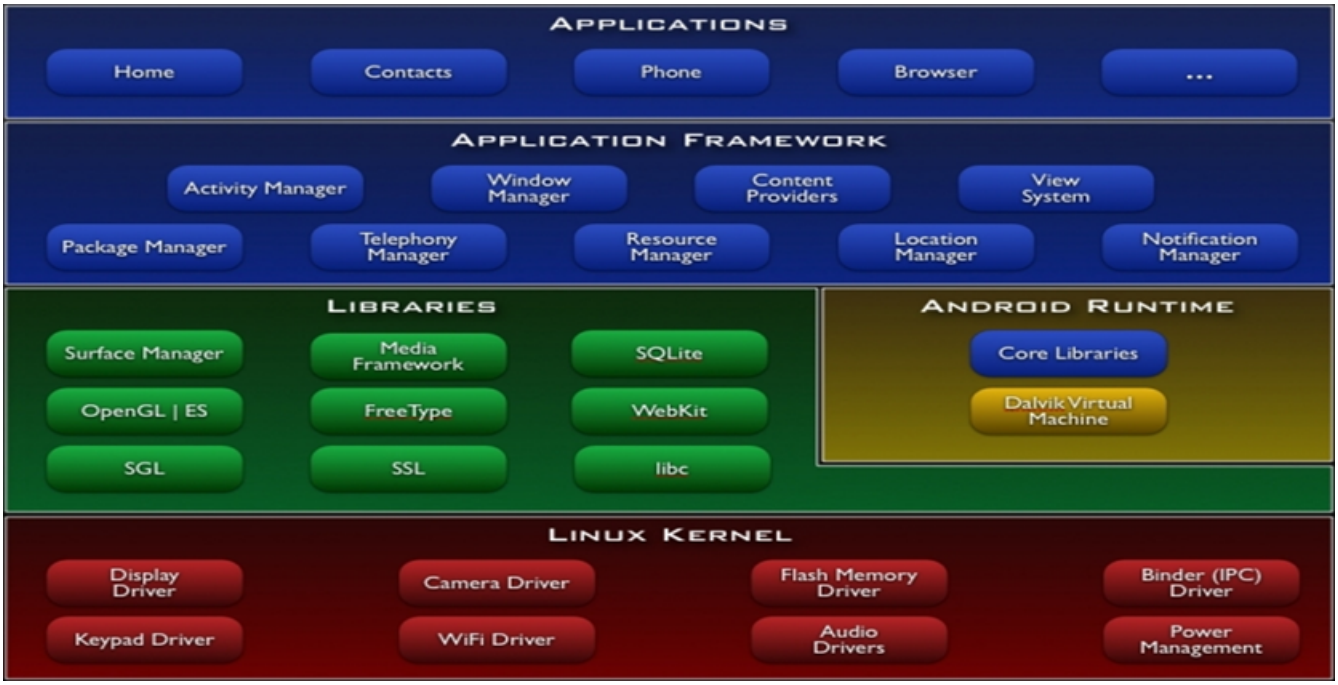
Figure 3. Cuota de mercado en % de terminales con las distintas versiones

3. Arquitectura

En los terminales Android encontramos

- Kernel linux 2.6
- Runtime basado en VM Dalvik o ART
- Conjunto de librerías C/C++
- Framework común a todas las aplicaciones basada en servicios.

- Aplicaciones base implementadas en Java (todas reemplazables).



4. Componentes

Los elementos basicos que forman las aplicaciones Android son

- Activity (Pantallas).
- View (Controles gráficos).
- Services (Procesos en segundo plano).
- Broadcast receivers (Listener).
- Content providers (Proveedores de contenido).
- Widget (Controles de escritorio).
- Intent (Navegación entre componentes)
- Notifications (Informacion de eventos)

4.1. Actividad (Activity)

Presenta una interfaz de usuario enfocada en algo que el usuario puede realizar: Elegir un contacto, seleccionar una fotografía, …

Una aplicación consistirá en un conjunto de actividades independientes que trabajan juntas

Una de las actividades se marca como la inicial al arrancar una aplicación.

4.2. Servicio (Service)

No tiene UI

Se ejecuta en background por periodo indefinido (Ej. Reproductor de música)

Expone una interface de métodos para interactuar (Ej. Parar la reproducción de música)

Es posible acceder desde otros componentes o aplicaciones

4.3. Broadcast receivers

No realiza ningún acción por si mismo.

Recibe y reacciona ante anuncios de tipo broadcast.

Existen muchos originados por el sistema (Ej. Batería baja)

Las aplicaciones puede lanzar un broadcast.

No tienen UI, aunque pueden iniciar una actividad para atender al anuncio.

4.4. Content providers

Expone un conjunto especifico de datos a otras aplicaciones.

Los datos pueden estar almacenados en cualquier lugar: fichero, SQLite, internet,…

Hace uso de un ContentResolver para acceder a los datos expuestos por un ContentProvider.

4.5. Widget

Aplicación visual, que normalmente pretende dar una información rápida y actualizada al usuario.

Se sitúan en el escritorio del terminal.

Normalmente tienen una única vista.

Suelen permitir acceder a funcionalidades de la aplicación asociada.

4.6. Intent

Mecanismo por el cual, se relaciona un evento que se produce en el terminal con un componente de las aplicaciones, ya sea una actividad, un servicio, …

Es el mecanismo por el que se relacionan los componentes entre si.

4.7. Notifications

Sistema de alertas del sistema, que permite informar al usuario de un evento que se haya producido.

Este sistema permite al usuario cambiar de aplicación.

5. Android Studio

Es el IDE de referencia para Android.

Esta basado en IntelliJ.

IntelliJ, al igual que Eclipse tiene un directorio de referencia workspace, donde se pueden crear varios proyectos, la diferencia con Eclipse radica en el concepto de **proyecto**, ya que en Eclipse un proyecto es la aplicación o librería de clases, en cambio en IntelliJ, es una agrupación de módulos, con los cuales se formará la aplicación, de echo al menos uno será la aplicación desplegable.

5.1. Configuracion

El menú File/settings de la barra de menú, da acceso a las configuraciones tanto del IDE, como del proyecto.

La configuración de las dependencias, se puede ver con botón derecho sobre el módulo

5.2. Ventanas

Para mostrar las ventanas, se puede acceder a la barra de menú View/Tool Window, se dispone entre otras de:

- Project, muestra el explorador del proyecto.
- Android, muestra la consola de ADB, con LogCat, el listado de Devices, ...
- Gradle, muestra las distintas tareas de Gradle que se pueden ejecutar sobre el proyecto.

5.3. Barras de Herramientas

Existen varias barras de herramientas, que se pueden activar o desactivar en la barra de menú View.

- Toolbar, muestra la botonera superior con opciones de Guardar, Compilar, Arrancar, Debug, ...
- Tool buttons, muestra botones para acceder a las ventanas Project, Android, ...
- Status bar, muestra la barra de estado inferior.
- Navigation bar, muestra la miga de pan.

5.4. Dependencias

Se pueden añadir dependencias gestionadas por **Gradle**, obtenidas desde un repositorio remoto

```
dependencies {
    compile 'com.android.support:appcompat-v7:21.0.3'
}
```

O tambien añadirlas al directorio **src/libs** del modulo del proyecto, referenciandolas desde la configuracion de Gradle

```
dependencies {
    compile files('libs/mysql-connector-java-5.1.29.jar')
}
```

O hacer referencia a otro modulo dentro del proyecto, estos será de tipo **Modulo de Librería**, que no es ejecutable.

```
dependencies {
    compile project(':dependencia')
}
```

Android Studio ofrece una forma de hacerlo a traves de los menus del propio IDE, accediendo a **Open Module Settings (F4)** y ahí en la seccion dependencies.

5.5. Accesos Rapidos

- Ctrl + Intro → Muestra menu para crear constructor, Getter y Setter, toString, Override Methods, ...
- Alt + Enter → Bombilla
- Ctrl + Space → InteliSense
- Ctrl + Alt + I → Identacion linea a linea
- Ctrl + Alt + O → Organizar Imports
- Alt + Ins → Abre menu de Generación de codigo, Getters, Setters, Constructor, ...
- Ctrl + H → Type Hierarchy
- Ctrl + Q → Documentación de un método.

6. Android Manifest

Fichero de configuracion de la aplicación Android.

El nodo principal de este XML, es el nodo <**Manifest**>, este permite definir los parámetros:

- xmlns:android → Esquema del documento.
- package → Paquete base a partir del cual se organizan los recursos compilados.
- android:versionCode → Entero que representa la versión de la aplicación (uso interno)
- android:versionName → String que representa la versión de la aplicación (uso externo)
- android:sharedUserId → El ID de usuario de Linux con el que correrá la aplicación. Por defecto, Android asigna a cada aplicación un ID de usuario único. Sin embargo, si este atributo se

establece con el mismo valor para dos o más aplicaciones, y están firmadas por el mismo certificado, se ejecutarán en el mismo proceso y pueden acceder a los datos de las demás.

- android:sharedUserLabel → Representa el Id de usuario de Linux de forma legible, ha de dársele valor a partir de un String definido como recurso y solo tiene sentido si se ha definido la propiedad android:sharedUserId.
- android:installLocation → Donde se instalará la aplicación, puede tener los valores:
  - internalOnly → Obliga a que se instale en la memoria interna, no se puede trasladar a la externa (Comportamiento por defecto).
  - auto → Por defecto se instala en la memoria interna, pero si esta esta llena se lleva a la externa, se puede mover una vez instalada.
  - preferExternal → Por defecto la instala en la externa, pero si esta no existe o esta llena la instala en la interna.

Cuando una aplicación se instala en el almacenamiento externo:

- El archivo apk se guarda en la memoria externa, pero los datos de la aplicación (por ejemplo, bases de datos) se guarda en la memoria interna del dispositivo.
- El archivo apk se guarda cifrado con una clave que permite que la aplicación solo funcione en el dispositivo que lo instaló. Aunque un dispositivo pueda tener varias tarjetas SD.

El nodo <**Manifest**>, puede contener los nodos:

- <application> → (Obligatorio) Define los componentes que forman la aplicación.
- <instrumentation> → Declara una clase de instrumentación que permite supervisar la interacción de una aplicación con el sistema. El objeto de instrumentación se instancia antes que cualquiera de los componentes de la aplicación.
- <permission> → Declara un permiso de seguridad que se puede utilizar para limitar el acceso a determinados componentes o características de esta u otras aplicaciones.
- <permission-group> → Declara una categoría en la que los permisos se pueden colocar.
- <permission-tree> → Declara un espacio de nombres en el que se pueden colocar permisos.
- <uses-configuration> → Indica qué características de hardware para la interacción con el usuario requiere la aplicación. Se utiliza para evitar la instalación de la aplicación en los dispositivos en los que no va a funcionar. Los atributos son:
  - reqFiveWayNav → Hardware de navegación (Trackball, …)
  - reqHardKeyboard –> Hardware de teclado.
  - reqKeyboardType → Tipo de teclado (nokeys, qwerty, twelvekey o undefined)
  - reqNavigation → Tipo de navegación (onav, dpad, trackball, wheel o undefined)
  - reqTouchScreen → Tipo de pantalla (notouch, stylus, finger o undefined)
- <uses-permission> → indica los permisos que necesita la aplicación para funcionar. Los permisos se conceden por el usuario cuando se instala la aplicación. Se puede encontrar una lista de los permisos definidos por la plataforma en la clase android.Manifest.permission.

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

- <permission> →Permite definir permisos personalizados, indicando el nivel del permiso (normal, dangerous, signature, signatureOrSystem)

```
<permission
    android:name="com.paad.DETONATE_DEVICE"
    android:protectionLevel="dangerous"
    android:label="Self Destruct"
    android:description="@string/detonate_description"/>
```

- <uses-feature> → Indica qué características de hardware requiere la aplicación. Se utiliza para evitar la instalación de la aplicación en los dispositivos en los que no va a funcionar.

```
<uses-feature android:name="android.hardware.nfc" />
```

Los posibles valores que puede tomar son: Audio, Bluetooth, Camera, Location, Microphone, NFC, Sensors, Telephony, Touchscreen, USB y Wi-Fi.

- <supports-screens> → Indica qué tipo de pantallas son soportadas por la aplicación.

```
<supports-screens
    android:smallScreens="false"
    android:normalScreens="true"
    android:largeScreens="true"
    android:xlargeScreens="true"
    android:requiresSmallestWidthDp="480"
    android:compatibleWidthLimitDp="600"
    android:largestWidthLimitDp="720"/>
```

- <uses-library> → Permite definir el uso de una librería compartida por parte de la aplicación, como por ejemplo la de Google.

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

- <activity> → Declara una actividad definida por una clase. Puede tener múltiples nodos hijos de tipo <intent-filter>, que permite definir cuando esta actividad se propondrá como capaz de actuar ante un evento.

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

- <service> → Declara un servicio definido por una clase. Puede tener múltiples nodos hijos de tipo <intent-filter>, que permite definir cuando este servicio se propondrá como capaz de actuar ante un evento.

```
<service android:name=".MyService"></service>
```

- <provider> → Declara un proveedor de contenido definido por una clase. Se emplea para compartir información con otras aplicaciones.

```
<provider
    android:name=".MyContentProvider"
    android:authorities="com.paad.myapp.MyContentProvider"/>
```

- <receiver> → Declara un listener definido por una clase.

```
<receiver android:name=".MyIntentReceiver">
    <intent-filter>
        <action android:name="com.paad.mybroadcastaction" />
    </intent-filter>
</receiver>
```

7. Aplication

Cada aplicación se ejecuta en su propio proceso linux con su VM Dalvik.

El runtime de Android gestiona el proceso de cada aplicación y por extensión de cada Actividad que contenga.

Las aplicaciones en Android no tienen control de su ciclo de vida.

Deben estar preparadas para su terminación en cualquier momento.

Con el pulsado prolongado sobre el botón físico de “home”, se accede a un gestor de Aplicaciones en ejecución, desde la cual se pueden detener las aplicaciones (no una actividad, sino la aplicación entera).

7.1. Tipos de aplicaciones

- Primer Plano (Foreground): Realiza su trabajo solo cuando es visible. Basados en Actividades. Ocupan toda la pantalla.
- Segundo Plano (Background): Realiza su trabajo sin ser visible. Basados en Servicios.
- Intermitente: Aúna los dos anteriores, realiza parte de su trabajo en segundo plano, pero permite al usuario interactuar.
- Widget: Aplicaciones que se muestran en el escritorio.

8. Contexto

Tanto las **Actividades** como los **Servicios**, extienden de **Context**, esta clase proporciona la interface de acceso desde estos componentes al contenedor de Android.

9. Actividades

Representa la interfaz de usuario (UI).

Se han de definir extendiendo la jerarquía de la clase **Activity**.

```
public class MainActivity extends Activity {}
```

Se han de declarar en el **AndroidManifest.xml**, ya que son **Beans manejados**.

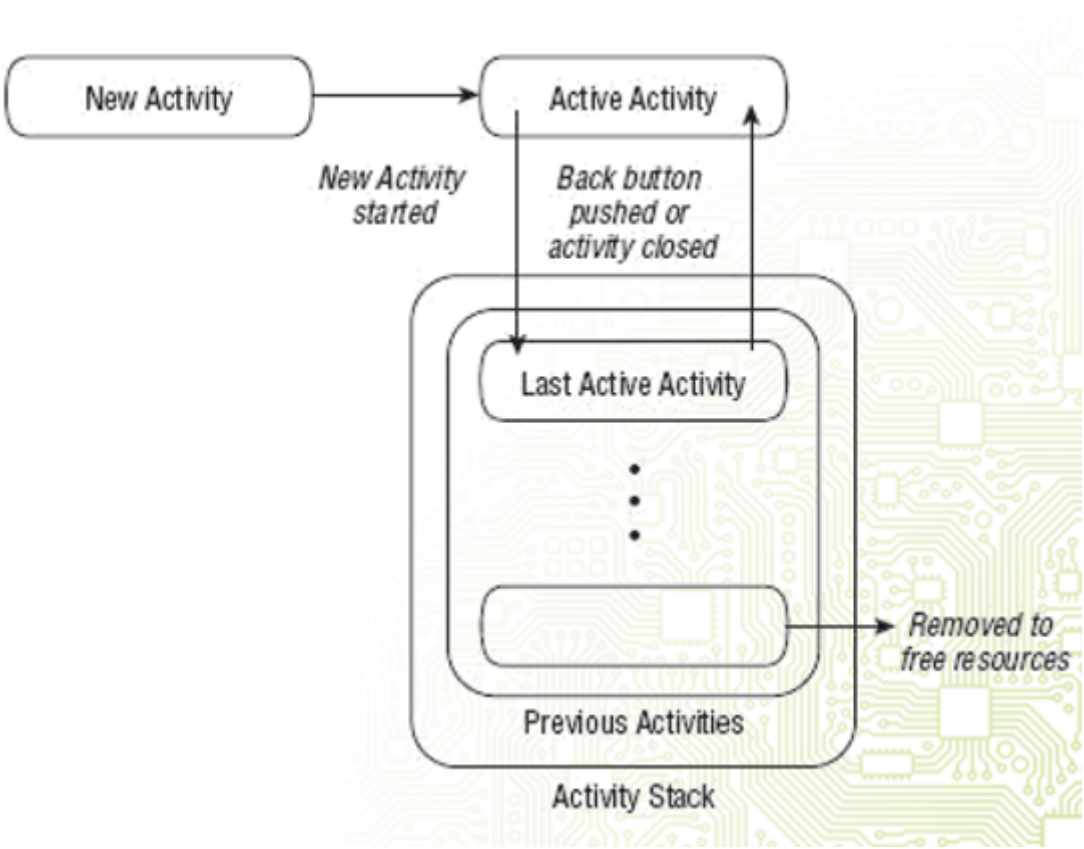
```
<activity android:name= "com.ejemplos.listadetareas.MainActivity"/>
```

Una de las actividades se ha de marcar como la **inicial**, que será la ejecutada cuando se arranque la aplicación desde el **Launcher**, para ello se declara un **Intent-Filter**.

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

9.1. Pila de Actividades

Consiste en una pila donde se van colocando las actividades que dejan de estar en primer plano y de la que se extraen actividades cuando se pulsa el botón **back**



Tiene una capacidad limitada y cuando esta se acaba **Android** evacua los objetos de ella, suponiendo esto la destrucción de la actividad.

9.2. Métodos

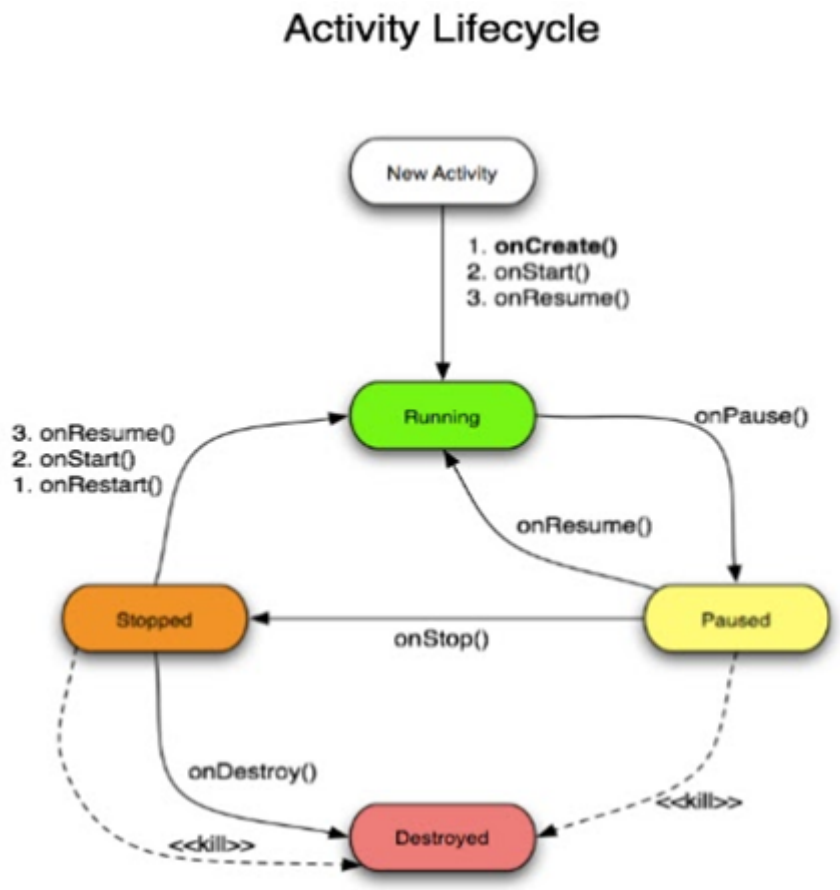
Algunos de los métodos más importantes de la clase Activity son

- **setContentView**: Permite la asignación del arbol de componentes **View** a la **Actividad**. Puede ser a partir de un **xml de layout** o de un objeto **View**.
- **getIntent**: Retorna la intención que dio origen a la **Actividad**.
- **getMenuInflater**: Retorna el objeto que se encarga de crear un objeto **Menu** a partir de un **xml**.
- **getPreferences**: Retorna el objeto **SharedPreferences** asociado a la **Actividad**.
- **getString**: Permite obtener una cadena de caracteres definida en un fichero de recursos.
- **onBackPressed**: Método que se ejecuta cuando se pulsa la tecla física **atrás**. Su funcionamiento por defecto es terminar la ejecución de la **Actividad**.
- **onCreateOptionsMenu**: Método invocado para crear el menú de opciones asociado a la actividad.
- **onOptionsItemSelected**: Método invocado cuando se selecciona una opción del menú de opciones asociado a la actividad.
- **onCreateContextMenu**: Método invocado para crear un menú contextual.
- **onContextItemSelected**: Método invocado cuando se selecciona una opción de un menú contextual.
- **setResult**: Establece el resultado de la ejecución de la Actividad, tanto el estado, como la

información retornada, se emplea cuando una **Actividad Secundaria**, retorna el control a la **Actividad Invocante** para dar feedback.

9.3. Ciclo de Vida

El siguiente diagrama muestra el ciclo de vida de una actividad



Para controlar el ciclo de vida, el API dispone de una serie de métodos de Listener, que van escuchando los eventos que el Contenedor va lanzando al cambiar el estado de la Actividad, estos son

- onCreate
- onStart
- onResume
- onPause
- onStop
- onDestroy
- onRestart

En el arranque de una actividad se pasa por unos estados transitorios, hasta llegar al estado Restaurada, estos son

- Creada (Se llega despues de ejecutar onCreate)
- Iniciada (Se llega despues de ejecutar onStart, se llega a este estado cuando se pulsa en el

launcher o se accede a las app recientes o se pulsa el boton atras y la actividad es la anterior en ejecucion o la activadad abierta durante la ejecución de esta actividad se termina (llamada de telefono))

- Reiniciada (Se llega despues de ejecutar onRestart, se llega a este estado cuando estando la Actividad en el Stack, se vuelve a abrir, se llega desde el estado Parada. Despues se ejecuta el onStart, se puede simular pasando al **Escritorio** con el boton físico y lanzandola desde el **Launcher**)

9.3.1. Estados estables

- Restaurada (Actividad responde a eventos del usuario, se llega despues de ejecutar onResume)
- Pausada (Actividad parcialmente oculta por una actividad mostrada como Dialogo, ojo que no es un dialogo, ya que los dialogos forman parte de la propia actividad, luego el estado de la actividad no cambia, no puede ejecutar codigo, se llega despues de ejecutar onPause, para poner una Actividad como un dialogo, hay que definir en el Manifest su Theme como

```
<activity android:theme="@style/Theme.AppCompat.Light.Dialog>"
```

- Parada (Actividad completamente oculta, la instancia existe, no puede ejecutar codigo)
- Destruida (se pierde la instancia, no es obligatorio la ejecución de onDestroy para llegar a este estado, por lo que es conveniente que los recursos abiertos, se liberen en onStop)

9.3.2. Navegaciones entre estados

Desde Restaurada se puede ir a Pausada, con onPause Desde Pausada se puede ir a Restaurada con onResume y a Parada con onStop Desde Parada se puede ir a Iniciada con onStart y a Destruida con onDestroy

9.3.3. Acciones a realizar en cada método

onPause → se deben parar funcionalidades que consuman recursos, como la ejecución de un video, guardar información en estado temporal, que sea interesante persistir si el usuario acaba abandonando la app (borrados de un mail) y deshabilitar escuchadores como broadcast

onRestart, se ejecuta antes que onStart, cuando el estado de partida en Parada.

9.4. Manteniendo el estado de la Actividad

9.5. Actividades del Sistema

El sistema ofrece unas funcionalidades de forma nativa, y estas pueden ser accedidas a traves de la invocación de intenciones.

9.5.1. Enviar mail con App del Sistema

```
Intent emailIntent = new Intent(android.content.Intent.ACTION_SEND);
emailIntent.setData(Uri.parse("mailto:"));
emailIntent.putExtra(android.content.Intent.EXTRA_EMAIL, to); //el parametro es
String[]
emailIntent.putExtra(android.content.Intent.EXTRA_CC, cc); //el parametro es String[]
emailIntent.putExtra(android.content.Intent.EXTRA_SUBJECT, asunto);
emailIntent.putExtra(android.content.Intent.EXTRA_TEXT, mensaje);
//Si se desea enviar como adjunto un icono local a la app.
emailIntent.putExtra(Intent.EXTRA_STREAM, Uri.parse("android.resource://" +
getPackageName() + "/" + R.drawable.icon));
emailIntent.setType("image/png");
```

9.6. Comunicacion entre Actividades

Para lanzar una nueva **Actividad**, el API proporciona una serie de métodos, principalmente

- **startActivity()** → Arranca una actividad a partir de una **Intent**.

```
Intent intent = new Intent(v.getContext(), ActividadSinResultadoActivity.class);
v.getContext().startActivity(intent);
```

- **startActivityForResult(Intent intent, int requestCode)** → Arranca una **Actividad**, de la cual espera un feedback, para lo cual la **Actividad Invocante** se define como **Listener** del final de la **Actividad Invocada**, esté **Listener** está representado por el método **onActivityResult(int requestCode, int resultCode, Intent result)**.

```
Intent intent = new Intent(v.getContext(),  ActividadConResultadoActivity.class);

((Activity)v.getContext()).startActivityForResult(intent,1);
```

El parámetro **requestCode**, permite a la actividad principal en su método **onActivityResult**, discriminar que Actividad secundaria ha respondido, y por tanto que ha de hacer con la respuesta.

El parámetro **resultCode**, permite a la actividad secundaria informar del estado de la petición.

Para intercambiar información entre las actividades, se empleará un **Intent**, principlamente empleando el **Bundle Extras** y el **URI Data**.

En los **Extras**, se podrán incluir datos que sean de los tipos primitivos, **Serializables** o **Parcelables**.

```
Intent intent = new Intent(v.getContext(),  ActividadConResultadoActivity.class);

intent.setData(Uri.parse("tel:555-2368"));

String[] datos =  {"1","2","3"};
intent.putExtra("datos",datos);
intent.putExtra("datosEjemplo", "Mensaje enviado a la actividad con resultados para
que lo trate");

((Activity)v.getContext()).startActivityForResult(intent,1);
```

10. Log

En construccion

11. Intents

Son mensajes asíncronos, que permiten la activación de componentes como Actividades o Servicios.

Las Intenciones, representan la intención de querer realizar algo (EJ: Seleccionar un contacto como remitente).

Las intenciones pueden ser de dos tipos:

- Implícita: Se la conoce como declarativa. No se conoce a priori quien atenderá la intención. Se pueden definir candidatos a traves de **intent-filter** en el AndroidManifest.
- Explícita: Se la conoce como programática. Se indica el tipo del objeto que ha de atender la intención.

11.1. Intent-Filter

Los intent-filter, permiten definir una actividad como candidata para procesar un tipo de intenciones, en este caso podemos ver como se establece una Actividad candidata para procesar la comparticion de texto y una imagen.

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
        <data android:mimeType="image/*"/>
    </intent-filter>
</activity>
```

Para invocar la anterior actividad, habria

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, "Este es mi texto a enviar.");
sendIntent.setType("text/plain");
startActivity(sendIntent);
```

11.2. Comprobar si la intencion será atendida

Antes de lanzar una intencion implicita, se puede comprobar si está será resuelta por algún componente declarado en el sistema.

```
PackageManager packageManager = getPackageManager();
List activities = packageManager.queryIntentActivities(intent, PackageManager
.MATCH_DEFAULT_ONLY);
boolean isIntentSafe = activities.size() > 0;
```

11.3. Selector de Actividades

Cuando se lanzan intenciones implicitas, pueden existir multiples candidatos para resolverla, en este caso se puede mostrar el selector de Actividades

En el siguiente ejemplo, se muestra en el selector todas las aplicaciones que pueden compartir un audio

```
Intent sendIntent = new Intent(Intent.ACTION_SEND);
sendIntent.setType("audio/*");
sendIntent.putExtra(Intent.EXTRA_SUBJECT, "SpyTools: AudioRecord");
sendIntent.putExtra(Intent.EXTRA_STREAM, Uri.parse("file://" + uri));

startActivity(Intent.createChooser(sendIntent, getString(R.string.compartir)));
```

12. Views

API que define los componentes visuales que se pueden añadir a las vistas.

Los principales son

- ImageView
- TextView
- ViewGroup

12.1. ViewGroup

Tipologia que cumple con el patrón **composite**, que permite construir objetos complejos a partir de otros más simples y similares entre sí formando una estructura de árbol.

En **Android** existen dos grandes grupos de objetos que cumplen con este patrón

- **Layout**
- **AdapterView**

Especial mencion a **RecyclerView**, nueva tipologia aparecida en **Lollipop** que persigue mejorar comportamientos de los **AdapterView**.

12.2. Layout

Especializados en organizar la distribución de los componentes en la pantalla.

Suelen ser el nodo principal de los XML de Layout.

12.2.1. Tipos

- **FrameLayout**
- **LinearLayout**
- **GridLAyout**
- **RelativeLayout**
- **TableLayout**

12.2.2. CoordinatorLayout

Layout que se encarga de gestionar las animaciones que se pueden definir en algunos componentes, por lo que normalmente figura como Layout principal.

Lo porporciona la libreria de diseño.

La idea una vez añadido el componente, es que todos los componentes dentro de el, a través de las propiedades

- **layout_behavior**. Aplicado sobre un componente dentro del **CoordinatorLayout**, permite definir el evento cuando el **CoordinatorLayout** tiene que aplicar alguna animación. Los valores posibles son
  - **@string/appbar_scrolling_view_behavior**. Evento de Scroll
- **layout_scrollFlags**. Aplicado sobre un componente dentro del **CoordinatorLayout**, permite definir que debe hacer con el componente el **CoordinatorLayout**. Los valores posibles son:
  - **scroll**. Indica que se ha de ocultar al hacer scroll hacia abajo.
  - **enterAlways**. Indica que el componente ocultado se mostrará inmediatamente al hace scroll

hacia arriba, aunque no le toque, de no aparecer solo se mostrará cuando se llegue arriba del todo.

- **exitUntilCollapsed**. Aplicable sobre **CollapsingToolbarLayout**, indica que el componente no desaparece.
- **enterAlwaysCollapsed**. Aplicable sobre **CollapsingToolbarLayout**, indica que se empieza a expandir cuando llega al top del scroll.

Otro efecto que permite, es el de anclar componentes a otros componentes, por ejemplo un boton flotante, anclado a un toolbar, esto se consigue con **app:layout_anchor="@id/appbarLayout"**

12.2.3. TabLayout

Componente que muestra un conjunto de pestañas, que permiten en conjunción con un **ViewPager** cambiar el contenido mostrado en la pantalla sin cambiar de actividad.

El componente viene incluido en la libreria de soporte de **design**

```
dependencies {
    compile 'com.android.support:design:+'
}
```

Para añadir el componente a un layout, se añade

```
<android.support.design.widget.TabLayout
    android:id="@+id/tabs"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

Para que el click sobre los **Tabs**, provoque un cambio de contenido, se ha de asociar el control de **TabLayout** al **ViewPager**

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.appbartabs);
tabLayout.setupWithViewPager(viewPager);
```

Adicionalmente, se puede definir el modo de representación de los **Tabs**, tipicamente cuando hay mas de 3, se emplea el modo **scrollable**, que permite deslizar los **tabs**, para acceder a los no visibles.

```
tabLayout.setTabMode(TabLayout.MODE_SCROLLABLE);
```

Por el momento no es posible codificar un color el subrayado que indica el tab seleccionado, ni para los divisores entre tabs.

12.3. AdapterView

Especializados en representar colecciones de objetos de la misma tipología de forma agrupada. Se basan en la creación de un objeto de tipo **BaseAdapter**, que se encarga de definir como se han de mostrar los datos en el **AdapterView**.

Lo normal, es que la estructura que tomen los datos en cada item del **AdapterView**, el **Adapter** los obtenga de un **XML de Layout**, el API de android, ofrece una serie de **XML de Layout** que pueden ser empleados, entre ellos:

- Simple_list_item_1
- Simple_list_item_2
- Simple_list_item_single_choice
- Simple_list_item_multiple_choice
- Simple_list_item_2_single_choice
- Simple_list_item_activated_1
- Simple_list_item_activated_2
- Simple_list_item_checked
- Simple_spinner_dropdown_item
- Simple_spinner_item

12.3.1. ¿Como funciona?

La idea es que el **AdapterView** es representable visualmente, pero no tiene la capacidad de representar los datos de una colección, para ello delega dicha funcionalidad en un **Adapter**, este recibira tantas llamadas desde el **AdapterView** al método **getView**, como elementos indique que hay el propio **Adapter** a través del método **getCount**. El **Adapter** en el método **getView** tendra que componer el arbol de objetos **View** que se deban representar en cada una de las posiciones del **AdapterView**, para lo cual necesitará tener acceso a la información, un **List** o un **Array** de objetos.

[adapterview como funciona] | *adapterview_como_funciona.png*

Captura 1: Funcionamiento del AdaoterView

Cuando el origen de datos (el **List** o **Array**), sea modificado, de forma automatica **NO** se actualizará el **AdapterView**, para coneguir dicho comportamiento, habrá que invocar **notifyDataSetChanged()**

```
adapter.notifyDataSetChanged();
```

12.3.2. BaseAdapter

Existen varias implmentaciones, las mas importantes son:

- ArrayAdapter

- CursorAdapter

ArrayAdapter, dispone de un método **setDropDownViewResource**, que permite definir como se muestran los elementos en el listado de seleccion en un **Spinner**, dado que el **Layout** que se indica con el constructor es el empleado por defecto para la representación del **Spinner** (cuando el dato esta selecionado).

```
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
```

Se puede crear un **ArrayAdapter** directamente desde un recurso de tipo **Array**

```
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,R.array
.colors,android.R.layout.simple_spinner_item);
```

12.3.3. Tipos

- **Spinner**. Representa una lista de seleccion simple.

```

```

- **ListView**. Representa una lista, permite la definición de una cabecera **addHeaderView** y un pie **addFooterView**.

```

```

- **GridView**. Representa una tabla. Aunque en apariencia es una tabla, en realidad, su comportamiento es muy similar al de un **ListView**, unicamente que antes de hacer el salto de linea representa varios componentes.

```
<GridView android:id="@+id/gvBanderas"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:numColumns="auto_fit"
    android:horizontalSpacing="2dp"
    android:verticalSpacing="5dp"
    android:stretchMode="spacingWidth"
    android:columnWidth="60px"/>
```

Las propiedades a destacar en **GridView** son:

- **numColumns**. Numero de columnas del grid, puede tomar el valor **auto_fit** si se ha de calcular con el tamaño de la pantalla y el resto de propiedades
- **columnWidth**. Ancho de cada columna del Grid.
- **horizontalSpacing**. Espacio horizontal entre elementos.

- **verticalSpacing**. Espacio vertical entre elementos.
- **stretchMode**. Procedimiento para repartir el espacio no ocupado por los elementos, puede ser
  - **columnWidth**. El espacio sobrante es repartido por las columnas de forma equitativa.
  - **spacingWidth**. El espacio sobrante es repartido por los espacios entre elementos de forma equitativa.
    - **Gallery**

12.3.4. Eventos

El principal evento para **Spinner** es **ItemSelected**.

```
spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    public void onItemSelected(AdapterView<?> parent, View v, int position, long id) {
        lblMensaje.setText("Seleccionado: " + parent.getItemAtPosition(position));
    }

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

El principal evento para **ListView** y para **GridView** es **ItemClick**.

```
lstOpciones.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> a, View v, int position, long id) {
        lblMensaje.setText("Seleccionado: " + a.getItemAtPosition(position));
    }
});
```

Es habitual emplear un menú contextual con los **ListView**, para mostrar opciones de borrado, edición, ... Para emplearlo primero habrá que registrar el componente para que tenga **Menu Contextual**, y en el **Listener** del evento **onContextItemSelected** obtener la posición del elemento pulsado.

```
@Override
public boolean onContextItemSelected(MenuItem item) {

    int position = ((AdapterView.AdapterContextMenuInfo) item.getMenuInfo()).position;

}
```

Tener en cuenta que la posición obtenida es en el **AdapterView**, que no tiene porque coincidir con la posición en la **Collection** que actua como origen de datos (puede haberse reordenado en el **Adapter**), por lo que las operaciones que se hagan, es mas recomendable que se realicen a través del **Adapter**, ya que este si tendra la colleción representada.

12.3.5. RecyclerView

Se trata de una implementación de **ViewGroup** que permite:

- Aplicar el patron **ViewHolder** mediante el API.
- Incluir efectos decorativos y de animacion.

La idea del componente es similar al de sus predecesores, definir un **Adaptador**, que establezca la relación entre los datos y el componente visual.

- **RecyclerView.Adapter<T extends RecyclerView.ViewHolder>**. Tiepologia que representa el adaptador. Debe implementar los siguientes métodos
  - **getItemCount**. Retorna el numero de elementos que se representan con el **RecyclerView**
  - **onCreateViewHolder**. Se invoca tantas veces como indique **getItemCount**, creando un **View** cada vez, encapsulado con un **RecyclerView.ViewHolder**. De momento el **View** no esta relleno con la información, unicamente se crea la estructura.
  - **onBindViewHolder**. Se invoca por cada **RecyclerView.ViewHolder** generado con el anterior método, indicando la posición que ocupa el **View** asociado al **RecyclerView.ViewHolder** en el **RecyclerView**, la idea es rellenar aqui la información del **View**, a través del **RecyclerView.ViewHolder**, que es quien tiene la referencia del **View**.

```
public class MiAdapter extends RecyclerView.Adapter<MiViewHolder> {

    private List<Dato> datos;

    public MiAdapter(List<Dato> datos){
        this.datos = datos;
    }


    @Override
    public MiViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        LayoutInflater inflater = LayoutInflater.from(parent.getContext());
        View view = inflater.inflate(android.R.layout.simple_list_item_1, parent,
false);

        //En este punto se tiene la referencia a cada uno de los View, por lo que es
el lugar
        //indicado para definir listener

        return new MiViewHolder(view);
    }

    @Override
    public void onBindViewHolder(MiViewHolder holder, int position) {
        Dato item = datos.get(position);
        holder.setData(item);
    }

    @Override
    public int getItemCount() {
        return datos.size();
    }
}
```

Como se aprecia, se aplica el patrón **ViewHolder** forzados por el API.

- **RecyclerView.ViewHolder**. Tipologia que representa el **ViewHolder**, debe de mantener principalmente la referencia a los componentes **View** de cada uno de los elementos que se representan en el **RecyclerView**, adicinalmente puede proporcionar funcionalidades para mantener los valores de dichas **View** o bien permitir el acceso a las mismas.

```
public class MiViewHolder extends RecyclerView.ViewHolder {

    private TextView tvDato;

    public MiViewHolder(View itemView) {
        super(itemView);

        tvDato = (TextView) itemView.findViewById(android.R.id.text1);
    }

    public void bindDato(Dato dato) {
        this.tvDato.setText(dato.getDato());
    }

    public Dato getDato(){
        return new Dato(this.tvDato.getText());
    }
}
```

RecyclerView necesita definir adicionalmente otro componente **LayoutManager**, que se encargará de establecer como se colocan los elementos en el **RecyclerView**, es precisamente en este componente donde reside la capacidad de sustituir tanto al **GridView** como al **ListView**, ya que el API proporciona implementaciones de **LayoutManager** precisamente para esas dos colocaciones.

- **LinearLayoutManager**. Muestra los elementos seguidos, como si fuese una lista, en vertical u horizontal, y permite mostrar los elementos en orden natural o inverso.

```
recView.setLayoutManager(new LinearLayoutManager(this,LinearLayoutManager.VERTICAL
,false));
```

- **GridLayoutManager**. Muestra los elementos como si fuera una tabla, indicando el numero de columnas que ha disponer.

```
recView.setLayoutManager(new GridLayoutManager(this,4));
```

Cuando se define un **RecyclerView**, se puede bloquear su tamaño con el método **setHasFixedSize()**, que no variará aunque se incluyan nuevos items.

Adicionalmente sobre los **RecyclerView**, se pueden definir **ItemDecoration**, que permiten incluir algun tipo de decoración visual a los item. Los definidos por el API son:

- **DividerItemDecoration**. Permite añadir decoraciones para dividir el espacio entre items (margenes). Tanto Vertical como Horizontal.

```
recView.addItemDecoration(new DividerItemDecoration(this,DividerItemDecoration
.VERTICAL_LIST));
recView.addItemDecoration(new DividerItemDecoration(this,DividerItemDecoration
.HORIZONTAL_LIST));
```

- **ItemTouchHelper**. Permite controlar eventos de **Swipe** y/o **Drag&Drop** sobre los item del **RecyclerView**. La idea de este componente es por un lado definir un **ItemTouchHelper**, que envuelve un **ItemTouchHelper.Callback**, que será el objeto encargado realmente de procesar los eventos de **swipe** y **drag&drop**.

La relación entre el **RecyclerView** y el **ItemTouchHelper** se realiza con **attachToRecyclerView**.

```
ItemTouchHelper itemTouchHelper = new ItemTouchHelper
(simpleItemTouchCallback);
itemTouchHelper.attachToRecyclerView(recyclerView);
```

- **ItemTouchHelper.Callback**. Define el comportamiento ante los eventos de **Swipe** o **Drag**. Lo primero que habrá que hacer es definir hacia que zona han de ser los movimientos para que sean procesados por el **Callback**, para ello hay constantes en la clase **ItemTouchHelper**.

Estas constantes representan **Flags**, luego son acumulativas.

Si se emplea la clase base **ItemTouchHelper.Callback** esta configuración se realiza en el método **getMovementFlags**.

```
public int getMovementFlags(RecyclerView recyclerView, RecyclerView.ViewHolder
viewHolder) {
    int dragFlags = ItemTouchHelper.UP | ItemTouchHelper.DOWN;
    int swipeFlags = ItemTouchHelper.START | ItemTouchHelper.END;
    return makeMovementFlags(dragFlags, swipeFlags);
}
```

Si la implementación elegida es **ItemTouchHelper.SimpleCallback**, se hará directamente en el constructor

```
new ItemTouchHelper.SimpleCallback(0, ItemTouchHelper.LEFT | ItemTouchHelper.RIGHT);
```

Se definirán los siguientes métodos.

- **getMovementFlags**. Permite definir que movimientos van a ser interpretados como **swipe**, y cuales como **drag&drop**
- **onMove**. Permite definir el comportamiento ante el drag & drop.

```
public boolean onMove(RecyclerView recyclerView, RecyclerView.ViewHolder from,
RecyclerView.ViewHolder to) {
    Collections.swap(countries,from.getAdapterPosition(), to.getAdapterPosition());
    adapter.notifyItemMoved(from.getAdapterPosition(), to.getAdapterPosition());
    return true;
}
```

- **onSwiped**. Permite definir el comportamiento ante el evento de swipe (deslizar).

```
public void onSwiped(RecyclerView.ViewHolder viewHolder, int direction) {
    int position = viewHolder.getAdapterPosition();

    if (direction == ItemTouchHelper.LEFT){
        countries.remove(position);
        adapter.notifyItemRemoved(position);
        adapter.notifyItemRangeChanged(position, countries.size());
    }
}
```

- **onChildDraw**. PErmite definir como se dibuja el **View** cuando se procesa el evento.

```
 public void onChildDraw(Canvas c, RecyclerView recyclerView, RecyclerView.ViewHolder
viewHolder, float dX, float dY, int actionState, boolean isCurrentlyActive) {
    if (actionState == ItemTouchHelper.ACTION_STATE_SWIPE) {
        // Fade out the view as it is swiped out of the parent's bounds
        final float alpha = ALPHA_FULL - Math.abs(dX) / (float) viewHolder.itemView
.getWidth();
        viewHolder.itemView.setAlpha(alpha);
        viewHolder.itemView.setTranslationX(dX);
    } else {
        super.onChildDraw(c, recyclerView, viewHolder, dX, dY, actionState,
isCurrentlyActive);
    }
}
```

12.4. Toolbar

Tambien llamado **ActionBar** o **AppBar**, es la barra superior (cabecera) que aparece en la mayoria de las aplicaciones Android. En ella se reflejan:

- Un titulo
- Un conjunto de iconos de acciones (Menu de Opciones)
- Un botón comodin (representado con tres puntos en vertical), que permite el acceso a las opciones que no caben en la barra.

El componente se obtiene de la libreria **appcompat**

```
dependencies {
    compile 'com.android.support:appcompat-v7:+'
}
```

12.4.1. Toolbar integrado en Activity

En las actividades de tipo **AppCompatActivity** viene integrado un componente **Toolbar**, para visualizarlo unicamente habrá que emplear un Tema para la actividad/aplicación que herede de **Theme.AppCompat**.

Si la configuracion de la aplicación en el **AndroidManifest** fuese

```
<application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
</application>
```

La configuración del tema en **/res/values/Styles.xml** podría ser

```
<resources>
     <style name="AppTheme" parent="Theme.AppCompat.Light">
        <item name="colorPrimary">@color/color_primary</item>
        <item name="colorPrimaryDark">@color/color_primary_dark</item>
        <item name="colorAccent">@color/color_accent</item>
    </style>
</resources>
```

En versiones anteriores de appcompat (anteriores a la 22), la clase base de las actividades era **ActionBarActivity**

Una vez definida la actividad y el tema, la implementación del **Toolbar**, será la del Menú de Opciones asociado a las actividades.

12.4.2. Toolbar como componente del Layout

Tambien se puede emplear el **Toolbar** como un componente mas de la vista de la actividad, añadiendola al **XML de layout**, lo cual permite mayores personalziaciones, para ello lo primero será deshabilitar el **Toolbar** enbebido en la actividad, para ello basta con cambiar el tema de la actividad/aplicación por uno de tipo **NoActionBar** (Theme.AppCompat.NoActionBar, Theme.AppCompat.Light.NoActionBar, ...)

```
<resources>
     <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <item name="colorPrimary">@color/color_primary</item>
        <item name="colorPrimaryDark">@color/color_primary_dark</item>
        <item name="colorAccent">@color/color_accent</item>
    </style>
</resources>
```

Una vez deshabilitado el **Toolbar** embebido, se debe añadir al layout el componente **Toolbar**

```
<android.support.v7.widget.Toolbar
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:id="@+id/appbar"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:minHeight="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        android:elevation="4dp"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
        app:popupTheme="@style/ThemeOverlay.AppCompat.Light" >
 </android.support.v7.widget.Toolbar>
```

Algunas de las propiedades del control

- **android:minHeight**. Se le suele asignar el valor **?attr/actionBarSize**, que permite que las **acciones** se muestren en la parte superior aunque el alto del control sea muy grande.
- **android:background**. Color de fondo, normalmente se esblece **?attr/colorPrimary**
- **android:elevation**. Elevación, lo recomendado 4dp (ver Material Design).
- **android:theme**. Tema para el control y todos sus hijos.
- **app:popupTheme**. Tema para el menu desplegable de opciones.

Una vez definido en el layout el componnete **Toolbar**, y dado que deshabilitamos el embebido, hay que registrar el nuevo **Toolbar**, como el menú de opciones de la **Actividad**, para poder emplear las funcionalidades del menu de opciones, para ello se emplea el método **setSupportActionBar**

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

    Toolbar toolbar = (Toolbar) findViewById(R.id.appbar);
    setSupportActionBar(toolbar);
}
```

Con esta configuración, conseguimos mayor control sobre el aspecto del **Toolbar**, puediendo incluir otros componentes dentro de el, ya que es un **AdapterView** y permite incluir componentes.

Tambien se puede incluir el **Toolbar** en otros componentes, puediendo así disponer de varios **Toolbar** en una misma **Actividad**.

12.4.3. Filtros

Se trata de añadir un **Spinner** al **Toolbar**, prestando especial atención a los estilos de pintado, dado que pertenecerá al **Toolbar**. Por lo que es interesante definir los layouts tanto para el elemento seleccionado, como para cada uno de los elementos candidatos a ser seleccionados, para ello:

- En el layout que represente el seleccionado, emplear los estilos del tipo **@style/TextAppearance.AppCompat.Widget.ActionBar.Title**

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    style="@style/TextAppearance.AppCompat.Widget.ActionBar.Title"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

- En el layout que representa a cada item candidato, emplear los estilos del tipo **?android:attr/spinnerDropDownItemStyle**.

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    style="?android:attr/spinnerDropDownItemStyle"
    android:layout_width="match_parent"
    android:layout_height="48dp" />
```

Al emplear los filtros, es habitual que no se muestre el titulo

```
getSupportActionBar().setDisplayShowTitleEnabled(false);
```

12.4.4. Tabs

Se activará como modo

```
actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
```

12.4.5. CollapsingToolbarLayout

Permite que **Toolbar** cambie de tamaño con el gesto de **scroll**

```
<android.support.design.widget.AppBarLayout
    android:id="@+id/appbarLayout"
    android:layout_width="match_parent"
    android:layout_height="192dp"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar" >

    <android.support.design.widget.CollapsingToolbarLayout
        android:id="@+id/ctlLayout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_scrollFlags="scroll|exitUntilCollapsed">

        <android.support.v7.widget.Toolbar
            xmlns:android="http://schemas.android.com/apk/res/android"
            xmlns:app="http://schemas.android.com/apk/res-auto"
            android:id="@+id/appbar"
            android:layout_height="?attr/actionBarSize"
            android:layout_width="match_parent"
            android:minHeight="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
            app:popupTheme="@style/ThemeOverlay.AppCompat.Light"
            app:layout_collapseMode="pin" >

        </android.support.v7.widget.Toolbar>

    </android.support.design.widget.CollapsingToolbarLayout>

</android.support.design.widget.AppBarLayout>
```

Si una actividad se abre como parte de otra, es decir como **detalle**, se puede incluir en el **Toolbar** el icono de volver indicando en la declaracion de la actividad de detalle, que la actividad invocante es su **padre**.

```
android:parentActivityName=".MainActivity"
```

Empleando la plantilla que ofrece Android Studio, se puede mostrar una imagen en el **Toolbar** unicamente haciendo lo siguiente

Añadir la imagen al layout

```
<android.support.design.widget.AppBarLayout
        android:id="@+id/app_bar"
        android:layout_width="match_parent"
        android:layout_height="@dimen/app_bar_height" <!-- Importante
esta dimension, ya que dependen de la imagen-->
        android:fitsSystemWindows="true"
        android:theme="@style/AppTheme.AppBarOverlay">

        <android.support.design.widget.CollapsingToolbarLayout
            android:id="@+id/toolbar_layout"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:fitsSystemWindows="true"
            app:contentScrim="?attr/colorPrimary"
            app:layout_scrollFlags="scroll|exitUntilCollapsed">

            <ImageView
                android:id="@+id/image_paralax"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:fitsSystemWindows="true"
                android:scaleType="centerInside" <!-- Importante esta
propiedad, ya que hace que la imagen se expanda, pero nunca se salga de
los limites -->
                app:layout_collapseMode="parallax"
                android:src="@drawable/imagen"/>

            <android.support.v7.widget.Toolbar
                android:id="@+id/toolbar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                app:layout_collapseMode="pin"
                app:popupTheme="@style/AppTheme.PopupOverlay" />

        </android.support.design.widget.CollapsingToolbarLayout>
    </android.support.design.widget.AppBarLayout>
```

La propiedad **app:layout_collapseMode**, tiene como valores posibles

- **pin**. La imagen desapare como empujada hacia arriba
- **parallax**. La imagen desaparece en paralelo de arriba y de abajo.

Tener en cuenta que un tamaño adecuado para la imagen a representar en un toolbar es de 640x420

12.5. NavigationView

Antes llamado **NavigationDrawer**, es el componente que comienza oculto y aparece deslizandose desde la izquierda al interaccionar con el botón situado en la parte izquierda del **Toolbar**.

La estructura de la pagina contendrá los siguientes componentes

- DrawerLayout
- NavigationView
- Layout de contenido

Estos componentes se obtienen de la libreria de soporte.

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context=".MainActivity">

    <!-- Layout real de la actividad -->
    <include layout="@layout/content_layout" />

    <!-- Layout del menú lateral (Navigation View) -->
    <android.support.design.widget.NavigationView
        android:id="@+id/navview"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:fitsSystemWindows="true"
        android:layout_gravity="start"
        app:headerLayout="@layout/header_navview"
        app:menu="@menu/menu_navview" />

</android.support.v4.widget.DrawerLayout>
```

La propiedad **android:fitsSystemWindows** del **DrawerLayout** y de **NavigationView**, permite mantener el componente deslizante debajo de la barra de notificaciones.

La propiedad **android:layout_gravity** de **NavigationView** indica el lado por el que saldrá el componente, **start** por la izquierda y **end** por la derecha.

La propiedad **app:headerLayout** de **navigationView**, permite configurar la cabecera del menu deslizante, será un **View**.

La porpiedad **app:menu** de **NavigationView**, permite configurar las opciones del menú, que será el menú tradicional de Android.

Para que se pueda activar el menu deslizante, se ha de configurar un **ActionBarDrawerToggle** asociado al **DrawerLayout** y al **Toolbar**.

```
DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
        this, drawer, toolbar, R.string.navigation_drawer_open, R.string
.navigation_drawer_close);
drawer.addDrawerListener(toggle);
toggle.syncState();
```

En control sobre las acciones seleccionadas del **NavigationView**, se realizará con un **OnNavigationItemSelectedListener**.

```
drawerLayout = (DrawerLayout)findViewById(R.id.drawer_layout);
navView = (NavigationView)findViewById(R.id.navview);

navView.setNavigationItemSelectedListener(
    new NavigationView.OnNavigationItemSelectedListener() {
        @Override
        public boolean onNavigationItemSelected(MenuItem menuItem) {

            boolean fragmentTransaction = false;
            Fragment fragment = null;

            switch (menuItem.getItemId()) {
                case R.id.menu_seccion_1:
                    fragment = new Fragment1();
                    fragmentTransaction = true;
                    break;
                case R.id.menu_seccion_2:
                    fragment = new Fragment2();
                    fragmentTransaction = true;
                    break;
                case R.id.menu_seccion_3:
                    fragment = new Fragment3();
                    fragmentTransaction = true;
                    break;
                case R.id.menu_opcion_1:
                    Log.i("NavigationView", "Pulsada opción 1");
                    break;
                case R.id.menu_opcion_2:
                    Log.i("NavigationView", "Pulsada opción 2");
                    break;
            }

            if(fragmentTransaction) {
                getSupportFragmentManager().beginTransaction()
                    .replace(R.id.content_frame, fragment)
                    .commit();

                menuItem.setChecked(true);
                getSupportActionBar().setTitle(menuItem.getTitle());
            }

            drawerLayout.closeDrawers();

            return true;
        }
    }
);
```

12.6. CardView

Componente que permite representar los View en forma de **Post-It**, con esquinas redondeadas y sombras.

Habrá que añadir una nueva dependencia del API de soporte, ya que es un componente que aparece en **Lollipop**

```
dependencies {
    compile 'com.android.support:cardview-v7:21.0.+'
}
```

12.7. ViewPager

Componente que permite alternar el contenido mostrado por el, con eventos de **swipe** a derecha e izquierda, entre distintos **Fragmentos**, para ello hace uso de un **FragmentPagerAdapter**.

Para asociar el **ViewPager** con un **FragmentPagerAdapter**, se realiza de forma similar a los **ViewAdapter**.

```
ViewPager viewPager = (ViewPager) findViewById(R.id.viewpager);
viewPager.setAdapter(new ChatFragmentPagerAdapter(getSupportFragmentManager()));
```

12.7.1. FragmentPagerAdapter

Adaptador que de forma analoga a lo que sucede con los **AdapterView**, proporciona los item a mostrar por el **ViewPager**, la caracteritica principal, es que los item en este caso serán **Fragments**. Se deben implementar:

- Constructor
- getCount()
- getItem(int position)

Adicionalmente se puede implementar **getPageTitle** para cambiar el titulo.

```
public class ChatFragmentPagerAdapter extends FragmentPagerAdapter {

    private String tabs[] = new String[] { "Llamadas", "Chats", "Contactos"};

    public ChatFragmentPagerAdapter(FragmentManager fm) {
        super(fm);
    }

    @Override
    public int getCount() {
        return tabs.lenght;
    }

    @Override
    public Fragment getItem(int position) {

        Fragment f = null;

        switch(position) {
            case 0:
                f = new LlamadasFragment();
                break;
            case 1:
                f = new ChatsFragment();
                break;
            case 2:
                f = new Contactos();
                break;
        }

        return f;
    }

    @Override
    public CharSequence getPageTitle(int position) {
        return tabs[position];
    }
}
```

13. Menús

Permiten ofrecer operaciones al usuario, sin ocupar espacio permanente en la pantalla.

Hay dos tipos

- Opciones. Asociado a la ventana.
- Contextual. Asociado a un componente.

13.1. Menú de Opciones

En versiones antiguas de Android, estaba asociado a un botón físico del terminal, actualmente esta asociado a un componente visual **Toolbar**.

Para definir un **menú de opciones** hay que definir los **item del menú**, para lo cual hay dos opciones:

- Definir un **XML de menú**
- Crear un objeto de tipo **Menu**, con **MenuItem** para cada opción del menú.

Lo más habitual es el **XML**.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools" tools:context=".MainActivity">
    <item android:id="@+id/action_settings" android:title="@string/action_settings"
        android:orderInCategory="100" app:showAsAction="always"
        android:icon="@android:drawable/ic_menu_manage"/>
    <item android:id="@+id/action_help" android:title="@string/action_help"
        android:orderInCategory="200" app:showAsAction="always"
        android:icon="@android:drawable/ic_menu_help"/>
</menu>
```

Este **xml** se debe situar en la carpeta **/res/menu**, tendra´un elemento principal **menu** y subelementos **item** que definirán cada una de las opciones, estas opciones pueden tener las siguientes propiedades

- **id**. Identificador.
- **title**. El texto que se visualizará para la opción.
- **icon**. Icono mostrado en **toolbar** cuando se muestra la **opción** como **acción**.
- **showAsAction**. Indica cuando la **opción** se muestra como **acción**. Puede combinar varios valores con |. Los valores posibles son:
  - **never**. Nunca se muestra como **acción**, siempre como **opción**
  - **always**. Siempre se muestra como **acción**, nunca como **opción**.
  - **ifRoom**. Se muestra como **acción** si cabe en la **toolbar**.
  - **withText**. Se muestra junto con el **title** en el **toolbar**.

Una vez definido el **XML** habrá que asociarlo con el toolbar de la actividad, para ello la actividad dispone de un método **onCreateOptionsMenu**.

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

Este método básicamente debe contruir el arbol de objetos asociado al menú, es aquí donde se deben construir los objetos de tipo **MenuItem**, y retornar el estado de la creación.

Una vez creado el menú de opciones, el siguiente paso sería escuchar el evento de selección sobre los item del menú, para ello de nuevo la actividad proporciona un listener **onOptionsItemSelected**, donde a través del identificador del item, se discrimina la tarea.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case  R.id.action_settings:
            startActivity(new Intent(this, SettingsActivity.class));
            break;
        case R.id.action_help:
            startActivity(new Intent(this, HelpActivity.class));
            break;
    }
    return super.onOptionsItemSelected(item);
}
```

13.1.1. Submenús

Como su nombre indica, son menús dentro de una opción de menú. Se definen así

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/configuracion" android:title="Opcion3"
          android:icon="@android:drawableic_menu_manage/">
        <menu>
            <item android:id="@+id/prefijo"
                  android:title="Prefijo" />
            <item android:id="@+id/sufijo"
                  android:title="Sufijo" />
        </menu>
    </item>
</menu>
```

El procesamiento de la pulsación de las opciones del **submenú**, se igual que con las opciones principales, se hace a través del **onOptionsItemSelected**, empleando el identificador del item.

13.2. Menú Contextual

Se asocian a los componentes visuales **View**, aunque los gestiona la Actividad.

- Se crea con **onCreateContextMenu**
- Se procesan las selecciones con **onContextItemSelected**

El menu contextual aparecerá con el evento de click prolongado.

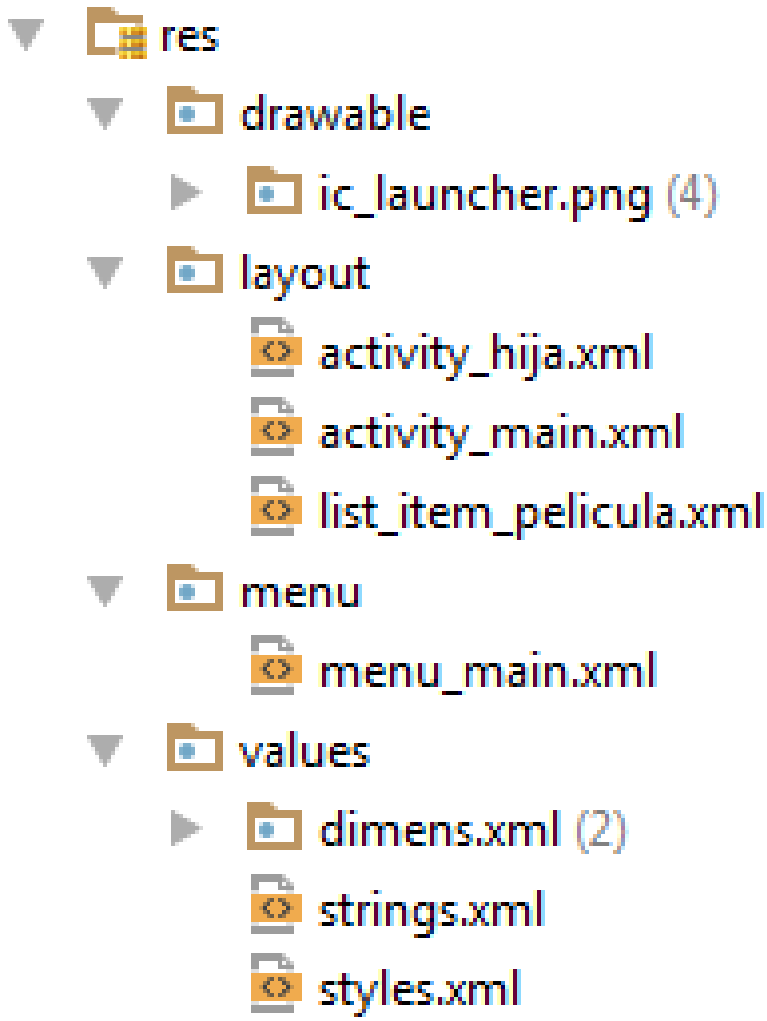
Se ha de registrar que componente visual tiene menu contextual, para ello se invoca **registerForContextMenu**

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    ListView lista = (ListView) findViewById(android.R.id.list);
    registerForContextMenu(lista);
    ...
}

@Override
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu
.ContextMenuInfo menuInfo) {
        super.onCreateContextMenu(menu, v, menuInfo);

        switch (v.getId()){
            case R.id.listView:
                //Queremos escribir en la cabecera algo que represente al item.
                int position = ((AdapterView.AdapterContextMenuInfo) menuInfo)
.position;
                Adapter adapter = ((AdapterView) v).getAdapter();
                menu.setHeaderTitle(adapter.getItem(position).toString());
                getMenuInflater().inflate(R.menu.menu_listview, menu);
                break;
        }
    }

    @Override
    public boolean onContextItemSelected(MenuItem item) {

        int position = ((AdapterView.AdapterContextMenuInfo) item.getMenuInfo())
.position;

        View targetView = ((AdapterView.AdapterContextMenuInfo) item.getMenuInfo())
.targetView;

        switch (item.getItemId()){
            case R.id.action_add:
                //Logica que da de alta un nuevo elemento
                Intent intent = new Intent(this, NuevaPeliculaActivity.class);
```

```
                startActivityForResult(intent,1);
                break;
            case R.id.action_edit:
                //Logica que edita
                Intent intent1 = new Intent(this, NuevaPeliculaActivity.class);
                intent1.putExtra("position",position);//Para que mandar el position??
si la actividad NuevaPeliculaActivity no tiene acceso al
                //adapter ni a la coleccion, y la es imposible obtenerlo, habra que
enviar el objeto asociado con el item pulsado en la lista
                startActivityForResult(intent1, 2);
                break;
            case R.id.action_remove:
                adapter.removePelicula(position);
                break;
        }

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

14. Recursos

En Android hablar de recursos, es hablar de la carpeta **res**, que es donde se englobaran todos los recursos internos del proyecto (apk).



La carpeta **res** esta intimamente ligada con la clase autogenerada **R**, que contendra una eferencia en forma de variable entera estatica a cada uno de los recursos incluidos en la carpeta **res**.

```
public final class R {
    public static final class anim {
        public static final int abc_fade_in=0x7f040000;
        public static final int abc_fade_out=0x7f040001;
        public static final int abc_slide_in_bottom=0x7f040002;
        public static final int abc_slide_in_top=0x7f040003;
        public static final int abc_slide_out_bottom=0x7f040004;
        public static final int abc_slide_out_top=0x7f040005;
    }
```

La clase **R** en Android Studio, se genera en la ruta **<modulo>/build/generated/source/r/debug/<paquete>**

14.1. Tipos de recursos

Los tipos de recursos que se pueden encontrar en la carpeta res son * **drawable**. Contendra eleemntos dibujables (imagenes, vectores, ...) * **layout**. Contendrá xml que representan las vistas de las actividades y fragmentos. * **menu**. Contendra xml que representan los elementos que componen los menus de opcionesy contextuales. * **values**. Contendra constantes y estilos de diversos tipos: String, string-array, int, ... * **xml**. Contendra ficheros de configuración de preferencias. * **anim**.

Contiene animaciones aplicables a los **View**. * **raw**. Contendrá ficheros no catalogables en las otras carpetas (.txt, .mp3, .png, .mov, .pdf, ...).

```
InputStream is = getResources().openRawResource(R.raw.test);
```

Los recursos de la carpeta res, se clasificarán dentro de la clase R, siguiendo una estructura similar a la de la carpeta res, pero no igual.

Las entradas que se crearán en la clase R, estarán organizadas en

- **id**
- **drawable**
- **layout**
- **menu**
- **raw**
- **string**
- **stringArray**
- **integer**
- **style**
- **xml**

14.2. Acceso a recursos

Para acceder a un recurso desde un XML, por ejemplo, acceder a un recurso de tipo **style** desde un XML de **layout**, se emplea la siguiente sintaxis

```
@[packagename:]resourcetype/resourceidentifier
```

El **packagename**, hace referencia al paquete de la **Application** que contiene los recursos, tipicamente se trabajan con dos paquetes, el de la propia aplicación, que puede ser obviado, y el del API de Android, que es **android**.

Para acceder a un recurso desde codigo java, se necesita el contexto y mas concretamente el método **getResources**, que proporciona un objeto **Resources**, que tiene un método para adquirir cada uno de los tipos de recursos, layout, string, drawable,...

Obtención de un String definido en el fichero **values/strings.xml** *desde una* **Ativity**

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

Se puede obtener el identificador generado en la clase R, para un recurso concreto

```
int imageId = getResources().getIdentifier("nombre del recurso", "drawable",
getActivity().getPackageName());
```

14.3. Assets

Tipo de recurso, que no es incluido en la clase **R**, para su acceso se emplea el **AssetManager**

Apertura de fichero en **Assets** *para lectura desde una* **Activity**.

```
InputStream is = getAssets().open("test.txt");
```

14.4. Memoria Interna

Se puede Leer y Escribir, las **Activities**, proporcionan los métodos **openFileInput** y **openFileOuput**.

Hay que recordar llamar al close.

La ruta donde se almacena el fichero será.

```
/data/data/<paquete_de_aplicacíon>/files/<nombre_fichero>
```

Aunque los ficheros aquí escritos pudieran ser accedidos por otras aplicaciones, esta posibilidad esta **deprecated**, por lo que esta memoría debera emplearse unicamente para ficheros de uso interno.

14.4.1. Escritura

El método **openFileOuput**, recibe como parámetros: * **nombre del fichero** * **modo de acceso**. Este modo de acceso puede ser:

- **MODE_PRIVATE** (por defecto) 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. (DEPRECATED)
- **MODE_WORLD_WRITABLE** para permitir a otras aplicaciones escribir sobre el fichero. (DEPRECATED)

Escritura en fichero de la memoria interna desde una **Activity**

```
OutputStreamWriter fout = new OutputStreamWriter(openFileOutput("prueba_int.txt"
,Context.MODE_PRIVATE));
fout.write("Texto de prueba.");
fout.close();
```

14.4.2. Lectura

openFileInput únicamente recibe como parámetro el nombre del fichero. El nombre del archivo no puede contener separadores de path

Lectura de fichero en la memoria interna desde un **Activity**

```
BufferedReader fin = new BufferedReader(new InputStreamReader(openFileInput(
"prueba_int.txt")));
String cadena = fin.readLine();
fin.close();
```

14.5. Memoria Externa (SD)

La tarjeta de memoria no tiene por qué estar presente en el dispositivo, e incluso estándolo puede no estar reconocida por el sistema, por tanto el primer paso recomendado a la hora de trabajar con ficheros en memoria externa es asegurarnos de que dicha memoria está presente y disponible para leer y/o escribir en ella.

El método estático **getExternalStorageStatus** de la clase **Environment** nos devuelve entre otros, los siguientes valores.

- **Environment.MEDIA_MOUNTED**, que indica que la memoria externa está disponible para leer y escribir.
- **Environment.MEDIA_MOUNTED_READ_ONLY**, memoria externa disponible sólo para leer.
- **Environment.MEDIA_UNMOUNTED**, se ha desmontado la tarjeta SD.
- **Environment.MEDIA_REMOVED**, se ha eliminado la tarjeta SD.

El método **getExternalStorageDirectory** de la clase **Eviroment** nos devolverá un objeto File con la ruta de dicho directorio.

Lectura de un fichero en la tarjeta SD

```
File ruta_sd = Environment.getExternalStorageDirectory();
File f = new File(ruta_sd.getAbsolutePath(), "prueba_sd.txt");
BufferedReader brin = new BufferedReader(new InputStreamReader(new FileInputStream(f)
));
String temp = brin.readLine();
brin.close();
```

Escritura de un fichero en la tarjeta SD

```
File ruta_sd = Environment.getExternalStorageDirectory();
File f = new File(ruta_sd.getAbsolutePath(), "prueba_sd.txt");
OutputStreamWriter fout = new OutputStreamWriter(new FileOutputStream(f));
fout.write("Texto de prueba.");
fout.close();
```

15. Estilos

Es un conjunto de propiedades graficas que permiten definir la forma como se pintan los componentes **View**, tamaño de fuente, color de fondo, elevacion, pading, ...

Los estilos en Android, siguen la misma filosofia que en CSS, la de separar el diseño, de la estructura de la vista.

En un Estilo se pueden definir todas las caracteristicas que pueden tener cada uno de los elementos de tipo View existentes en Android, aplicandose unicamente aquellas caracteristicas propias del View al que se aplique el Estilo.

GridView tiene una propiedad layout_gravity, aplicable a cada elemento dentro del layout que permite a los elementos en la celda expandirse para ocupar todo el espacio, especialmente util con los columnspan y rowspan.

Los estilos se definen como recursos xml en la carpeta res/values, tipicamente el un fichero styles.xml, aunque esto ultimo no es obligatorio.

Para la definición de un estilo, se ha de definir un nodo principal <resources> y dentro de el un nodo <style> indicando un name, que será unico, y que permitirá hacer referencia al estilo desde el resto de recursos Android.

Dentro de cada nodo <style> se definiran cada una de las caracteristicas que se engloban en el estilo empleando nodos <item> indicando con name la caracteristica y con el contenido del nodo el valor de la caracteristica.

Ejemplo de definición de un estilo en Android

```
<resources>
    <style name="MiEstilo">
        <item name="titleTextAppearance">@style/TextAppearance.Widget
.AppCompat.Toolbar.Title</item>
        <item name="subtitleTextAppearance">@style/TextAppearance.Widget
.AppCompat.Toolbar.Subtitle</item>
        <item name="android:minHeight">?attr/actionBarSize</item>
        <item name="titleMargins">4dp</item>
        <item name="maxButtonHeight">56dp</item>
        <item name="collapseContentDescription">@string
/abc_toolbar_collapse_description</item>
        <item name="contentInsetStart">16dp</item>
    </style>
</resources>
```

Una vez definido un estilo, este se puede referenciar desde los componentes View, tipicamente desde los ficheros xml de layout

Ejemplo de referencia a un estilo en Android

```
<TextView
    style="@style/MiEstilo"
    android:text="@string/hello" />
```

15.1. Herencia

La Herencia entre estilos, permite que un estilo tome como base para su definición otro estilo, puediendo sobreescribir aquellas caracteristicas que desee.

La herencia se establece definiendo en el atributo parent del nodo <style> el name del estilo base.

Ejemplo de herencia en un estilo en Android

```
<resources>
    <style name="MiEstilo" parent="EstiloBase">
    </style>
</resources>
```

Se puede crear la estructura de herencia que se desee, no hay limite de herencia.

16. Temas

Los temas, no son mas que estilos, que se aplican sobre una aplicación o una actividad, de forma que todos los componentes View dentro de esa aplicación o actividad, se ven afectados por el estilo.

Este es un comportamiento que solo se da con los temas, dado que aplicar un estilo a un componente View de tipo Composite, por ejemplo un Layout, no hará que se aplique a todos los elementos dentro del Layout dicho estilo.

Para aplicar un tema, se ha de hacer referencia al estilo a aplicar desde el atributo theme de los nodos <activity> y <application> del AndroidManifest.xml

Ejemplo del uso de un estilo como tema en Android

```
<application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
</application>
```

Existen una serie de atributos solo aplicables a los estilos de tipo tema, que configuran el estilo aplicado a los tipos de componentes dentro del tema

16.1. Colores Principales

```
A traves de cualquier recurso de la carpeta */res/values*, tipicamente */styles.xml*
se pueden configurar los tres colores principales:
```

- **colorPrimary**. Es el color principal de la aplicación, y se utilizará entre otras cosas como color de fondo de la action bar.
- **colorPrimaryDark**. Es una variante más oscura del color principal, que por ejemplo en Android 5.0 se utilizará como color de la barra de estado (la barra superior que contiene los iconos de notificación, batería, reloj, ...). Nota: en Android 4.x e inferiores la barra de estado permanecerá negra.
- **colorAccent**. Suele ser el color utilizado para destacar el botón de acción proncipal de la aplicación (por ejemplo un botón flotante como el que vimos en el artículo sobre botones, y para otros pequeños detalles de la interfaz, como por ejemplo algunos controles, cuadros de texto o checkboxes.

Existe un generador de colores online para **MaterialDesign**, a la que podemos acceder desde aquí.

16.2. Atributos graficos de Layout

- **android:layout_width**. Ancho del componente, es obligatorio, de tener 0dp, será el atributo **layout_weight** el que decida su anchura.
- **android:layout_height**. Alto del componente.
- **android:layout_weight**. Peso del componente dentro del layout, indica los que ocupa con respecto al resto de componentes dentro del mismo layout. Cuanto mayor, mas peso y mas ocupa dentro del layout, es una proporción, por lo que si uno tiene 1 y otro tiene 2, el de 2 ocupa el doble que el de 1.
- **android:layout_marginLeft**. Margen que se separa el componente de la zona izquierda.
- **android:layout_marginTop**. Margen que se separa el componente de la zona superior.
- **android:layout_marginBottom**. Permite indicar el espacio hasta el margen inferior que deja el componente.
- **android:layout_marginEnd**. Permite indicar el espacio hasta el margen derecho que deja el componente.

16.3. Atributos graficos de Textos

- **android:text**. Texto que visualizará el componente. Debe ser un **TextView**.
- **android:textStyle**. Estilo del texto que se visualiza en el componente. Puede ser: **bold**, **italic**, ... El componente debe ser un **TextView**.
- **android:textSize**. Tamaño del texto que se visualiza en el componente. Debe ser un **TextView**.
- **android:textColor**. Color del texto.

- **android:textAppearance**. Atributo que permite referenciar a un estilo que defina todos los atributos de tipo texto.

Referencia a un textAppearance

```
<TextView
        android:textAppearance="?android:textAppearanceLarge"/>
```

Definicion de Attr **android:textAppearanceLarge** *asociado al estilo empleado por la App*

```
<style name="Platform.AppCompat" parent="android:Theme">
    <item name="android:textAppearanceLarge">@style/TextAppearance.AppCompat.Large<
/item>
</style>
```

Definicion del estilo **TextAppearance.AppCompat.Large** *asociado con el Attr* **android:textAppearanceLarge**

```
<style name="TextAppearance.AppCompat.Large">
    <item name="android:textSize">@dimen/abc_text_size_large_material</item>
    <item name="android:textColor">?android:attr/textColorPrimary</item>
</style>
```

16.4. Otros atributos graficos

- **android:windowBackground**. Background de la **Application** o **Activity**, no es aplicable a **Views**. Puede ser un **Color** o un **Drawable**.
- **android:gravity**. Forma en la que se justifica un componente. Puede tener como valores un conjunto de: **Left**, **Right**, **Top**, **Bottom** y **Center**.
- **android:background**. Permite definir un color de fondo o imagen de un **View**.
- **android:scrollbars**. Permite definir en que sentido se quieren tener las barras de desplazamiento. Los posibles valores son **horizontal** y **vertical**. (Se puede aplicar en el control RecyclerView)
- **android:alignParentBottom**. Parametro **boolean** que permite indicar que el **FloatingActionButton** se situa en la parte inferior del contenedor que lo alberga.
- **android:alignParentEnd**. Parametro **boolean** que permite indicar que el **FloatingActionButton** se situa en la parte derecha del contenedor que lo alberga.
- **android:elevation**. Indica la altura del componente con respecto a la elevación base. Provoca sombras.s
- **android:transitionName**. Permite indicar el tipo de transición que provoca un evento sobre un componente. (Aplicable a FloatingActionButton). Los valores posibles son: **fab_transition**
- **android:clicKable**
- **android:foreground**

- **card_view:cardCornerRadius**
- **android:padding**. Distancia entre todos los margenes del **View** y el contenido que contiene el **View**.

16.5. Unidades de medida

- **dp**: Unidad independiente de la densidad de la pantalla.
- **sp**: Como **dp**, pero para los textos.

16.6. Atributos

Los atriutos de los Temas, son caracteristicas que se declaran en un fichero **attrs.xml** y que establecen su valor en un fichero de **styles.xml** asoiados a un tema

Declaracion de atributo

```
<resources>
    <declare-styleable name="AppTheme">
        <attr name="menuIconCamera" format="reference" />
    </declare-styleable>
</resources>
```

Definición de atributo para un tema dado

```
<style name="AppTheme.Light" parent="@android:style/Theme.Holo.Light">
    <item name="menuIconCamera">@drawable/ic_menu_camera_holo_light</item>
</style>
```

17. Material Design

Nueva vision de los estilos aparecido en 2014, que pretende sustituir la anterior filosofia **Holo**.

Persigue dar una directiva de diseño homogenea para aplicaciones **Android**, **Web**, **TV**, ...

17.1. Principios

- Se crean **capas** de **Papel digital** superpuestas.
- Lo que se pinta en las capas se hace con **Tinta digital**.
- Se permite definir **animaciones** que aporten valor al contenido, que resalten los contenidos y ayuden al usuario a centrarse en la parte importante de la vista.
- Se ha de aplicar diseño adaptativo **Responsive desgin**.

17.2. Papel digital

Se define un tercer eje, el Z, que define la profundidad, cada **capa**, ocupa 1dp de grueso, no puede tener grosor de 0dp.

Las capas tiene capacidades:

- Cambiar de forma (estirarse, reducirse, rotar en el mismo plano, ...)
- Proyectan sombras sobre las capas que están debajo.
- Pueden conectarse/unirse, formando una nueva capa.
- Pueden moverse en los tres ejes.

Aun teniendo estas capacidades, las capas no pueden atravesarse.

Hay dos tipos de sombras * Ambiente * Directa

17.3. Tinta Digital

Se emplea una sola fuente **Roboto**, esto se debe a que una unica fuente permite realizar resaltado basandose por tamaño o color.

La paleta de colores estandar se basa en los colores primarios, dividiendo entre colores principales y acentuados.

Se proporciona una libreria **Palette**, que permite extraer los colores predominantes de imagenes de forma dinámica, con lo que la UI podría adaptar sus colores a los predominantes en la imagen.

Material es solido, los eventos de entrada, es decir las interacciones del usuario con la pantalla, no atraviesan los componentes

Los componentes de Material pueden cambiar su forma, siempre que sea en su plano, no se permiten pliegues o dobleces.

Se pueden juntar componentes de distintos niveles y se pueden partir componentes, de ocurrir esto segundo, los trozos que queden en la misma capa, se pueden unir (cicatrizar)

Los componentes Material se pueden crear o destruir dinamicamente.

El movimiento en Z, es el resultado, de forma general, de la interaccion del usuario con la app.

Todos los componentes o capas tienen una elevacion en reposo AppBar 4dp Floating Action Button 8dp Card 2dp

Si un componente cambia su elevación, deberia volver a ella lo antes posible.

Los diferentes estados de los componentes (normal, enfocado, presionado), pueden cambiar la elevacion

Las sombras son coherentes con la distancia entre capas, mas distancia, sombra mas larga y difusa,

menos distancia, sobra mas corta y fuerte.

Los componentes, tienen relaciones padre-hijo, donde las propiedades como transformacion, rotacion, escala y elevacion son heredadas, luego si el padre ve afectada alguna propiedad, los hijos tambien.

La separacion en Z de padres e hijos es minima y no puede haber nada entre medias (como si se pegasen fotos (hijos) sobre una hoja (padre)).

Algunos componentes van por su cuenta, como menus laterales, actionbars dialogos.

Aceleraciones rapidas al principio que se frenan al final, cuando un componente entra/salga en escena, que lo haga ya acelerado

17.4. Aplicar Tema Material

Crear dos ficheros **style.xml** en la carpeta **/res/values**, uno por defecto para los API anteriores al 21, con el siguiente contenido

```
<resources>
    <style name="BaseTheme" parent="Theme.AppCompat.Light.DarkActionBar"></style>
    <style name="AppTheme" parent="BaseTheme"></style>
</resources>
```

Y otro para los API iguales o mayores que el 21 con el siguiente contenido

```
<resources>
    <style name="AppTheme" parent="BaseTheme"></style>
</resources>
```

17.5. Animaciones

Con **Material Design** se pueden definir transiciones asociadas a los **View**, actualmente y desde el API 21, se soporta la animación circular **createCircularReveal**.

```
//Componente sobre el que se aplica la animación
final View myView = findViewById(R.id.textView);

//Se oculta el componente, para que sea visible cuando se active la animación
myView.setVisibility(View.INVISIBLE);

//Establecer quien lanza la animación
findViewById(R.id.button).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        //La animación, que mostrara un circulo creciendo desde un radio 0, hasta un
radio definido,
        //centrara el circulo en la parte superior izquierda del componente.
        int cx = myView.getLeft();
        int cy = myView.getTop();

        //El radio final, es el maximo de los dos lados.
        int finalRadius = Math.max(myView.getWidth(), myView.getHeight());

        //Se crea la animación de tipo circular.
        Animator anim =
                ViewAnimationUtils.createCircularReveal(myView, cx, cy, 0,
finalRadius);

        //Se establce un escuchador para volver a ocultar el componente una vez
termina la animación
        anim.addListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                super.onAnimationEnd(animation);
                //Se oculta el componente
                myView.setVisibility(View.INVISIBLE);
            }
        });

        //Se hace visible el componente justo antes de empezar la animacion
        myView.setVisibility(View.VISIBLE);
        //Se establece la duracion de la animación en 2 segundos
        anim.setDuration(2000);
        //Se inicia la animacion.
        anim.start();
    }
});
```

17.6. Transiciones entre Actividades

Proporcionan fluided entre los distintos estados por lo que atraviesa la vista, mediante el movimiento y las transformaciones de elementos comunes.

Se pueden definir animaciones al entrar/salir de las actividades, en las que pueden participar elementos comunes.

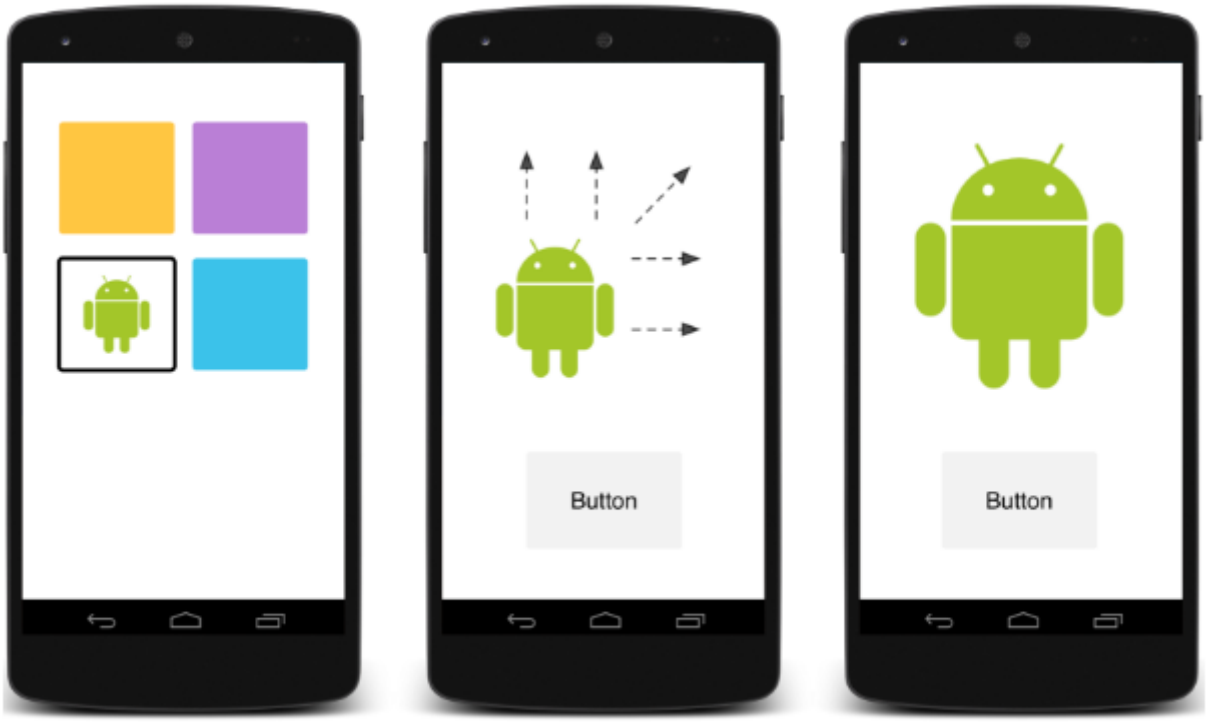
Desde el API 21, se permiten estas transiciones de entrada y salida:

- **expandir**: desplaza vistas hacia adentro o hacia afuera del centro de la escena.
- **deslizar**: desplaza vistas hacia adentro o hacia afuera de uno de los bordes de la escena.
- **difuminar**: agrega o quita una vista de la escena al cambiar su opacidad.

A partir de aqui, toda extension de clase **Visibility** se admite como una transición de entrada o salida.

Desde el API 21, se permiten estas transiciones de elementos compartidos:

- **changeBounds**: anima los cambios en los límites de las vistas de destino.
- **changeClipBounds**: anima los cambios en los límites de recorte de las vistas de destino.
- **changeTransform**: anima los cambios en escala y rotación de las vistas de destino.
- **changeImageTransform**: anima los cambios de tamaño y escala de imágenes de destino.



Para emplear las transiones entre **Activities** se han de habilitar, para ello se ha de definir el siguiente parametro en el estilo empleado

```
<!-- Habilitar transiones entre actividades -->
<item name="android:windowContentTransitions">true</item>
```

Se pueden personalizar que transiciones se han de emplear en cada uno de los casos, para ello

```
<!-- Definir el tipo de transicion de entrada -->
<item name="android:windowEnterTransition">@transition/explode</item>

<!-- Definir el tipo de transicion de salida -->
<item name="android:windowExitTransition">@transition/explode</item>

<!-- Definir el tipo de transicion de entrada con componentes compartidos -->
<item name="android:windowSharedElementEnterTransition"
>@transition/change_image_transform</item>

<!-- Definir el tipo de transicion de salida con componentes compartidos -->
<item name="android:windowSharedElementExitTransition"
>@transition/change_image_transform</item>
```

Estas transiciones se pueden definir en la carpeta **res/transition**

fichero res/transition/change_image_transform.xml

```
<transitionSet xmlns:android="http://schemas.android.com/apk/res/android">
    <changeImageTransform/>
</transitionSet>
```

fichero res/transition/explode.xml

```
<transitionSet xmlns:android="http://schemas.android.com/apk/res/android">
    <explode android:duration="2000"></explode>
</transitionSet>
```

Una vez habilitadas las transiciones, habrá que arrancar la actividad indicando que va a haber una transicion, esto es posible desde el API 16.

```
Intent intent = new Intent(MainActivity.this, DetalleActivity.class);
startActivity(intent, ActivityOptions.makeSceneTransitionAnimation(MainActivity.this)
.toBundle());
```

Si se desea que se comparta una vista en concreto, habrá que indicar el mismo identificador de componente compartido al **View** en las dos actividades

```
<View android:transitionName="view_compartida"/>
```

Y arrancar la actividad con

```
Intent intent = new Intent(MainActivity.this, Activity2.class);
ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation(MainActivity
.this, viewShared, "view_compartida");
startActivity(intent, options.toBundle());
```

Y si es una actividad iniciada con **startActivityForResult**, para finalizarla emplear

```
Activity.finishAfterTransition();
```

La transición en estos casos será habitualmente una que escale los componentes (tipicamente una imagen) y que haga que el escalado empiece desde donde esta la imagen de partida en la pantalla llegando a donde esta la imagen final en la pantalla, para ello

fichero res/transition/changeImageTransform.xml

```
<transitionSet xmlns:android="http://schemas.android.com/apk/res/android">
    <changeBounds/>
    <changeImageTransform/>
</transitionSet>
```

De quererse incluir la transición al incluir un nuevo fragmento, se ha de incluir el elemento compartido en la transaccion de fragmentos

```
getFragmentManager().beginTransaction()
    .replace(R.id.fragment_container, fragment2)
    .addSharedElement(view, "shared_view")
    .addToBackStack("fragment1")
    .commit();
```

18. Fragmentos

Surgen de la necesidad de adaptar las aplicaciones a las pantallas grandes de las **tablets**, para aprovechar las nuevas dimensiones, se busca reutilizar las vistas.

Aparece a partir del API 3.0

Un **Fragment** no es un **View** ni un **Activity**, se parece más a una **Activity**, ya que es una porción de la UI, vista y controlador, pero no es autonoma, puede añadirse o eliminarse de forma independiente al resto de elementos de una **Activity**.

Las **Activity** por tanto pueden estar formadas por varios **Fragment**, teniendo a su vez cada **Fragment** un conjunto de **Views**.

Los **Fragment** pueden ser añadidos a una **Activity** de dos formas

- **Estatica**. Indicando en el **Layout** que se incluye un *Fragment de un tipo concreto.

```
<fragment
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:name="com.example.tarde.fragmentos.ListadoFragment"
    android:id="@+id/fragmentoListado"
    android:layout_alignParentTop="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"
    android:layout_alignParentBottom="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />
```

- **Dinámica**. Empleando el **FragmentManager** para incluir un **Fragment** en un determinado **hueco** de la UI

Los **Fragment** añadidos de forma estatica en el **xml de layout**, no son gestionables de forma dinámica.

18.1. FragmentManager

Para la gestión de los **Fragment**, se incorpora el servicio **FragmentManager**, que permite definir que **Fragments** forman parte de una **Activity** en cada momento.

Las clases **Activity** incorporan un método que obtiene la referencia al servicio **FragmentManager**

- **getSupportFragmentManager()**. En **Actividades** del API de soporte.
- **getFragmentManager()**. En **Actividades** del API básico.

FragmentManager permite

- Buscar **Fragment** definidos dentro del **Layout** de la actividad con **findFragmentById()**. Hay que tener en cuenta que como no son **View**, que si una **Activity** tiene **Fragment**, estos no pueden ser localizados con **findViewById()**.

```
FragmentManager fragmentManager = getFragmentManager();

ListadoFragment fragmentoListado = (ListadoFragment) fragmentManager.findFragmentById
(R.id.fragmentoListado);
```

- Controlar que **Fragment** se asocian al **Activity**. Este comportamiento es transaccional, es decir se pueden realizar varias operaciones de adición, borrado o actualización sobre **Fragments** en un mismo **Activity**, siendo consideradas todas como una unidad (**Atomica**), de tal forma que si en alguna de las operaciones falla, ninguna de las que formaba parte de la transacción, se llevará a cabo.

```
FragmentTransaction tx = getFragmentManager().beginTransaction();

tx.add(R.id.contenedor, new BuscadorFragment());

tx.commit();
```

Los métodos de **FragmentTransacion** son:

- add
- replace
- remove
- commit
- rollback

Para poder añadir un **Fragment** a una **Activity**, debe existir un componente de tipo **Layout** que haga de contenedor, donde se incluira la porción de UI, que proporciona el **Fragment**, tipicamente se recurre a un **FrameLayout**, por ser el Layout mas simple.

```
<FrameLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_below="@+id/textView"
    android:layout_toRightOf="@+id/textView"
    android:layout_toEndOf="@+id/textView"
    android:layout_marginTop="52dp"
    android:id="@+id/contenedor"></FrameLayout>
```

18.2. Implementación de Fragment

Un **Fragment**, está representado por una clase que herede de la jerarquia de **android.app.Fragment**, cuando se emplea el API de soporte, será de **android.support.v4.app.Fragment**.

Esta clase proporciona los siguientes métodos:

- **onCreateView()**. Debe retornar el arbol de componentes **View** que representa el **Fragment**.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
    return inflater.inflate(R.layout.fragment_detalle, container);
}
```

- **onActivityCreated()**. Se ejecutará una vez se haya construido toda la **Activity**, en el se inicializarán los valores de los **View** y se registrarán los **Listener**.

```
@Override
public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);

    View guardar = getActivity().findViewById(R.id.btGuardar);

    View cancelar = getActivity().findViewById(R.id.btCancelar);

    guardar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // TODO implementar cuando se expliquen los Adapter
        }
    });

    cancelar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // TODO implementar cuando se expliquen los Adapter
        }
    });
}
```

- **getActivity()**. Funcionalidad que permite obtener la referencia de la **Activity** contenedora. Será interesante para establecer la comunicacion entre **Fragment**, ya que es la **Activity** la que deberá de realizar tareas de mediación, ya que los **Fragment** dentro de una misma **Activity** no se ven.

Se puede incluir las transacciones con **Fragments**, dentro del **BackStack**, con el método **addToBackStack()**, de tal forma que el botón atrás incluye no solo **Activity**, sino tambien cambios de **Fragment**.

19. Dialogos

Son ventanas modales asociadas a las **Activities**. Al contrario que los **Toast**, interrumpen el uso normal de una **Activity**, ya que precisan de la atención inmediata del usuario, quedando la ventana bloqueada por el **Dialogo**. Se suelen emplear para

- Mostrar un mensaje.
- Pedir una confirmación rápida.
- Solicitar al usuario una elección.

Los **Dialogos** se crean empleando **Fragment**, mas concretamente **DialogFragment**.

Fragmento que implementa un Dialogo

```
public class AdvertenciaDialogFragment extends DialogFragment {}
```

Aunque en las **Activities** existen los métodos **onCreateDialog** y **showDialog**, estan **deprecated**, por lo que no esta recomendado su uso, y se han de emplear los analogos de la clase **DialogFragment**

La implementación del **Fragment**, debe implementar el método **onCreateDialog**

```
private AlertDialog.Builder builder;

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    return builder.create();
}
```

Como se puede apreciar, se hace referencia a un **Builder**, que es el que se encarga de crear el **Dialog**. Lo mas habitual, es que este **Builder** se configure en la creación del **Fragment**, para que así este disponible siempre que se necesite mostrar el **Dialog**.

Inicialización del AlertDialog.Builder

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    builder = new AlertDialog.Builder(getActivity());

    builder.setTitle("Advertencia")
            .setIcon(android.R.drawable.ic_dialog_alert)
            .setNeutralButton("Aceptar",new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    Toast.makeText(getActivity(), "Se acepto", Toast.LENGTH_LONG).
show();
                    dialog.dismiss();
                }
            });
}
```

En este **Builder**, se pueden definir todos los aspectos necesarios para representar el **Dialog**.

- **setTitle**. Titulo del **Dialog**
- **setIcon**. Icono del **Dialog**
- **setMessage**. Texto descriptivo de la pregunta o mensaje que se hace al usuario.
- **setNeutralButton**. Texto del botón y **Listener** de **OnClick** para el bóton central.

- **setPositiveButton**. Texto del botón y **Listener** de **OnClick** para el bóton derecho
- **setNegativeButton**. Texto del botón y **Listener** de **OnClick** para un bóton izquierdo
- **setView**. Objeto de tipo View, que representa el **Dialog**.
- **setAdapter**.
- **setItems**.
- **setSingleChoiceItems**. Listado de elementos, junto con el **Listener** a ejecutar cuando se pulse uno de los elementos.
- **MultiChoiceItems**. Listado de elementos, junto con el **Listener** a ejecutar cuando se pulse uno de los elementos.

Una vez configurado, unicamente habrá que ejecutar el método **show** sobre una instancia de un **DialogFragment** para mostrar el **Dialog**.

```
AdvertenciaDialogFragment dialogFragment = new AdvertenciaDialogFragment();
dialogFragment.show(getSupportFragmentManager(),"Dialogo");
```

Si se desea eliminar el **Dialog**, se ha de invocar **dismiss()**.

Si se desea mostrar el **Dialog**, se ha de invocar **show()**.

Si se desea ocultar el **Dialog**, se ha de invocar **hide()**.

Existen tres implementaciones funcionales de la clase **AlertDialog**

- **TimePickerDialog**

```

```

- **DatePickerDialog**

```
Calendar c = Calendar.getInstance();
int year = c.get(Calendar.YEAR);
int month = c.get(Calendar.MONTH);
int day = c.get(Calendar.DAY_OF_MONTH);

DatePickerDialog datePickerDialog = new DatePickerDialog(
    this,
    new DatePickerDialog.OnDateSetListener() {
        @Override
        public void onDateSet(DatePicker view, int year, int monthOfYear, int
dayOfMonth) {
        }
    },
    year, month, day);

datePickerDialog.show();
```

- **ProgressDialog**

```
ProgressDialog progreso;

if(progreso == null) {
    progreso = new ProgressDialog(this);
    progreso.setMax(100);
    progreso.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
    progreso.setProgress(0);
    progreso.show();
} else if(progreso.getProgress() == 100){
    progreso.hide();
    progreso.setProgress(0);
} else if(progreso.getProgress() == 0) {
    progreso.show();
} else {
    progreso.setProgress(progreso.getProgress() + 10);
}
```

20. Notificaciones

Sistema de aviso al usuario persistente, no bloqueante, es decir no exige el tratamiento inmediato del aviso.

Se representan como mensajes mostrados en la barra de notificaciones.

Para construir una **Notification**, se emplea un **builder** que permite configurar las notificaciones antes de construirlas y guardar dicha configuración para su reutilización.

Creación del Builder

```
Builder builder = new Notification.Builder(this);

builder.setSmallIcon(android.R.drawable.stat_sys_warning);
builder.setTicker("Alerta!");
builder.setAutoCancel(true);
builder.setContentText("Texto");
builder.setContentTitle("Titulo");
builder.setContentInfo("Texto de Info");
builder.setWhen(System.currentTimeMillis());
```

Las opciones que se pueden definir en el **Builder** son

- **setSmallIcon**. Define el Icono que se muestra en la barra de notificaciones.
- **setTicker**. Define un texto que desaparece de la barra de notificaciones.
- **setAutoCancel**. Indica si la Notificacion desaparece al pulsar sobre ella.
- **setContentText**. Define el texto que aparece en la Notificacion.
- **setContentTitle**. Define el titulo de la Notificacion.
- **setContentInfo**. Define información que aparece en la Notificacion.
- **setWhen**. Define una fecha asociada a la notificacion.
- **setShowWhen**. Define si se muestra la fecha asociada a la notificacion
- **setLargeIcon**. Define un Icono que se muestra en la Notificacion.
- **setStyle**. Define el estilo de la Notificación, existen varias opciones:
  - **Notification.BigPictureStyle**
  - **Notification.BigTextStyle**
  - **Notification.InboxStyle**
  - **Notification.MediaStyle**
- **setContentIntent**. Define un **PendingIntent**, que encapsula una **Intent** que se lanzará cuando se pulse sobre la **Notificación**.
- **setLights**. Define el color con el que se enciende el Led cuando hay una notificación de este tipo pendiente.
- **setVibrate**. Define un patron de vibración, basado en un array de long, donde cada long representa alteernativamente el tiempo en milisegundos que esta encendido o apagado el vibrador.
- **addAction**. Permite añadir un botón asociado con un **PendingIntent** en la parte inferior de la **Notificacion**.
- **setProgress**. Permite añadir una barra de progreso, esta puede ser de dos tipos:
  - **indeterminada**, unicamnete se mueve.
  - **determinada**, permite indicar la cantidad progresada.

- **setSound**. Permite indicar que sonido se escuchara cuando se notifique, indicando este con una URI.

Una vez configurado el **builder**, se proceden a crear las notificaciones

Instanciación de las Notificaciones a traves del Builder

```
//API 11
Notification notification = builder.getNotification();
//API 16
Notification notification = builder.build();
```

Una vez creada la **Notification**, se ha publicar, para ello se emplea en **NotificationManager**

Obtencion del **NotificationManager**

```
NotificationManager notificationManager = (NotificationManager) getSystemService
(NOTIFICATION_SERVICE);
//o
NotificationManagerCompat notificationManager = NotificationManagerCompat.from(this);
```

Para publicar las **Notification**, se ha de indicar un identificador para el tipo de notificaciones, que permite al **NotificationManager** sustituir una notificacion mas antigua por una mas nueva, sin tener que mostrar todas las notificaciones.

Publicacion de notificacion

```
notificationManager.notify(NOTIF_ALERTA_ID2, notification);
```

21. PendingIntent

Representa la posibilidad que tienen algunos servicios del sistema, de lanzar una **Intent** en diferido, empleando los permisos de la **Application** que genera el **PendingIntent**.

Algunos de estos servicios del Sistema son **NotificationManager**, **AlarmManager**, ...

Para instancíar el **PendingIntent**, se emplean los métodos estaticos de la clase

- **PendingIntent.getBroadcast**
- **PendingIntent.getActivity**
- **PendingIntent.getService**

```
Intent intent = new Intent(this, NotificationReceiver.class);

PendingIntent pIntent=PendingIntent.getActivity(this,0,intent,0);
```

Los parametros que pide en **PendingIntent**, son * **RequestCode**. * **Flags**. Puede tomar los siguientes valores

- **FLAG_ONE_SHOT**
- **FLAG_NO_CREATE**
- **FLAG_CANCEL_CURRENT**
- **FLAG_UPDATE_CURRENT**

22. Shared Preferences

Las preferencias son una forma que proporciona **Android** para persistir información de forma local.

Se definen a nivel de la aplicación, luego todas las actividades, servicios, ... de una aplicación tienen acceso a las mismas preferencias.

La persistencia esta basada en XML, los ficheros generados se pueden localizar en la carpeta

```
/data/data/<paquete>/shared_prefs/<nombre>.xml
```

Siendo el contenido del XML

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

Los datos que se deberian almacenar el las **Shared Preferences** son aquellos que permiten personalizar la experiencia del usuario con la aplicación, por ejemplo información personal, opciones de presentación, etc...

Cada preferencia se almacenará en forma de **clave-valor**, donde la clave es un identificador único (p.e. **email**) y el valor, un valor asociado a dicho identificador (p.e. **prueba@email.com**).

El manejo de este API, se centra en la clase **SharedPrefences**, que representa una colección de preferencias.

Una aplicación puede manejar varias colecciones de preferencias, que se diferenciaran por un identificador único.

Para obtener la referencia a la colección de preferencias se emplea el método **getSharedPrefences** de **Activity**.

```
SharedPreferences prefs = getSharedPreferences("MisPreferencias", Context.
MODE_PRIVATE);
```

Donde se pide como parametros, el identificador de la colección de preferencias y el modo de acceso al XML donde se almacenan las preferencias, este modo puede tomar los siguientes valores:

- **MODE_PRIVATE**. Sólo nuestra aplicación tiene acceso a estas preferencias.
- **MODE_WORLD_READABLE**. Todas las aplicaciones pueden leer estas preferencias, pero sólo la nuestra puede modificarlas. (DEPRECATED)
- **MODE_WORLD_WRITABLE**. Todas las aplicaciones pueden leer y modificar estas preferencias.(DEPRECATED)

El la clase **Context**, existen constantes que representan los modos.

Tambien se definen unas preferencias por defecto, que no tienen un identificador configurable, que se obtienen con

```
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
```

22.1. Lectura

Una vez se tiene la referencia a las preferencias, se pueden obtener los valores con métodos get.

- getInt
- getFloat
- getLong
- getBoolean
- getString
- getStringSet

```
String correo = prefs.getString("email", "por_defecto@email.com");
```

22.2. Escritura

Para el establecimiento de las propiedades, se emplea el objeto **Editor**.

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

22.3. Implementación

Para hacer accesibles las preferencias al usuario de la aplicación, se precisa de un formulario que permita el establecimiento/modificacion de las mismas, esto se puede lograr de varias maneras:

- **Manualmente**, generando una nueva actividad o fragmento, que represente los datos a perssitir en las preferencias y empleando el objeto **SharedPReferences** se codifique la interacción con las preferencias.
- **PreferenceActivity**, Actividad que permite asociar de forma sencilla un XML donde se definen los datos a persistir con la actividad, generando de forma automatica el formulario necesario para el establecimiento/modificacion de las preferencias

```
public class PreferenciasActivity extends PreferenceActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState)  addPreferencesFromResource(R.xml
.preferences);
    }
}
```

- **PreferencesFragment**, similar a la anterior, pero el formulario de establecimiento/modificacion de preferencias no tiene porque ser lo único que aparecezca en la actividad. (> API 11)

```
public class PrefsFragment extends PreferenceFragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(
                    R.xml.preferences);
    }
}
```

En los dos últimos casos, se hace referencia a un XML, que define las preferencias, este XML se ha de situar en

```
/res/xml
```

Los elementos que pueden aparecer en este XML son

- **PreferenceScreen**. Nodo principal.
- **PreferenceCategory**. Nodo para agrupar las propiedades por categorías.
- **CheckBoxPreference**. Marca seleccionable.

```
<CheckBoxPreference
    android:key="opcion1"
    android:title="Preferencia 1"
    android:summary="Descripción de la preferencia 1" />
```

- **EditTextPreference**. Cadena simple de texto.

```
<EditTextPreference
   android:key="opcion2"
   android:title="Preferencia 2"
   android:summary=
        "Descripción de la preferencia 2"
   android:dialogTitle="Introduce valor" />
```

- **ListPreference**. Lista de valores seleccionables.

```
<ListPreference
   android:key="opcion3"
   android:title="Preferencia 3"
   android:summary=
        "Descripción de la preferencia 3"
   android:dialogTitle="Indicar Pais"
   android:entries="@array/pais"
   android:entryValues="@array/codigopais" />
```

- **MultiSelectListPreference** (> API 11). Lista de valores seleccionables múltiple.

```
<MultiSelectListPreference
   android:key="opcion4"
   android:title="Preferencia 4"
   android:summary=
        "Descripción de la preferencia 4"
   android:dialogTitle="Indicar Pais"
   android:entries="@array/pais"
   android:entryValues="@array/codigopais" />
```

23. Bases de Datos SQLite

SQLite es un motor de bases de datos que ofrece características tan interesantes como:

- Pequeño tamaño.
- No necesita servidor.
- Poca configuración.
- Transaccional.

- Código libre.

Es el motor de base de datos embebido en las plataformas Mac y Android.

Una de las herramientas para administrar la base de datos, es sqliteadmin, que se puede descargar desde aquí

Para conectarnos a la base de datos, tendremos que copiar a nuestro equipo local el fichero y acceder a el a través de sqliteadmin.

Por defecto las bases de datos en Android son privadas a la aplicación, si se quiere compartir información entra aplicaciones se emplearán los **ContentProviders**.

Todas las bases de datos se crean en la carpeta.

```
/data/data/<paquete>/databases
```

Para las consultas de Base de Datos con Android incluidas en los XML, se han de poner los Varchar entre comillas dobles escapadas con \, es decir: \"Dato\"

Acepta campos autoincrementales

```
CREATE TABLE Noticias (_id INTEGER PRIMARY KEY AUTOINCREMENT)
```

23.1. Tipos de datos soportados

- INTEGER
- FLOAT
- REAL
- NUMERIC
- BOOLEAN
- TIME (hh:mm:ss)
- DATE (Formato dd/MM/yyyy)
- TIMESTAMP (Formato dd/MM/yyyy hh:mm:ss)
- VARCHAR
- NVARCHAR
- TEXT (Equivale a CLOB)
- BLOB

23.2. SQLiteOpenHelper

La interfaz con la Base de datos nos la proporciona la clase abstracta **SQLiteOpenHelper**, al extender dicha clase, habrá que implementar dos métodos:

- **onCreate** (invocado al crear la BBDD nueva)
- **onUpgrade** (invocado al emplear una base de datos existente, con versiones distintas, es decir, se ha modificado el esquema de la base de datos, en ocasiones se borra y se crea nuevo, pero pueden producirse perdida de datos).

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE Noticias (_id INTEGER PRIMARY KEY AUTOINCREMENT, titulo
TEXT, link TEXT)");
}
@Override
public void onUpgrade(SQLiteDatabase db, int versionAnterior, int versionNueva) {
    db.execSQL("DROP TABLE IF EXISTS Noticias");
    db.execSQL(sqlCreate);
}
```

La clase **SQLiteOpenHelper**, tiene un método **getWritableDatabase()** que nos devolverá un objeto de tipo **SQLiteDatabase**.

23.3. ContentValues

Es una tipologia que permite crear objetos que llevan embebido un **Map**, donde las claves son los nombre de los campos y los valores los valores del registro.

Tiene métodos put, que aceptan:

- Boolean
- Byte
- Double
- Float
- Integer
- Long
- Short
- String

Y un método putNull, que permite establecer null en un campo.

23.4. SQLiteDatabase

Esta clase provee los métodos para interaccionar con la base de datos.

- **execSQL()** → Ejecuta una sentencia SQL que se le pase como parámetro. Es parametrizable.

```
db.execSQL("CREATE TABLE Noticias (_id INTEGER PRIMARY KEY AUTOINCREMENT, titulo TEXT,
link TEXT)");
```

- **insert()** → Inserta un registro en una tabla. Retorna el Id (long) asignado al registro. Se pasan por parámetro:
  - Table: Nombre de la tabla.
  - NullColumnHack: En caso de insertar todos los campos null, el nombre de un campo en el que explícitamente se inserte null.
  - Values: Valores de los campos en forma de **ContentValues**.

```
long id = db.insert(table, null, values);
```

- **delete()** → Borra uno o varios registros de una tabla. Retorna el numero de registros borrados. Se pasan por parámetros:
  - Table: Nombre de la tabla.
  - WhereClause: String que defina la clausula where, es parametrizable con ?.
  - WhereArgs: Array de String con los valores con los que se han de sustituir las ? de la clausula where.

```
String[] args = new String[] { id + "" };
int registrosBorrados = db.delete("table", "_id = ?", args);
```

OJO con las comillas!! Los parámetros son siempre String

- **update()** → Actualiza un registro de una tabla. Retorna el numero de registros afectados. Se pasan por parámetros:
  - Tabla: Nombre de la tabla.
  - Values: Valores de los campos en forma de ContentValues.
  - WhereClause: String que defina la clausula where, es parametrizable con ?.
  - WhereArgs: Array de String con los valores con los que se han de sustituir las ? de la clausula where.

```
String[] args = new String[] { id + "" };
db.update("table", ContentValues , "_id = ?", args);
```

- **rawQuery()** → rawQuery(). Ejecuta una sentencia de consulta sobre la base de datos, devolviendo un cursor. Recibe como parámetros.
  - Query: La consulta donde pueden aparecer ?.
  - QueryArgs: String[] con los valores con los que se han de sustituir las ?.

```
String query = "SELECT _id, titulo, link, descripcion, fecha FROM Noticias WHERE _id =
?";
String[] args = new String[] { id + "" };
Cursor cursor = db.rawQuery(query, args);
```

- **query()** → Consulta una tabla, devolviendo un cursor. Recibe como parámetros.
  - Table: Nombre de la tabla.
  - Columns: String[] con los nombres de las columnas a retornar, si se pone null, retorna todas.
  - WhereClause: String que defina la clausula where, es parametrizable con ?.
  - WhereArgs: Array de String con los valores con los que se han de sustituir las ? de la clausula where.
  - GroupBy: Clausula para agrupar registros.
  - Having: Clausula where sobre grupos.
  - OrderBy: Clausula para definir el orden de los registros retornados.

```
String[] args = new String[] { id + "" };
db.query(table, columns, "_id = ?", args, null, null, null, null);
```

23.5. Cursor

Es un puntero a un conjunto de datos.

Dispone de los siguientes métodos para moverse por los datos:

- move(posicionRelativa).
- moveToPosition(posicion).
- moveToFirst.
- moveToLast.
- moveToPrevious.
- moveToNext.
- isFirst.
- isLast.
- isAfterLast.
- isBeforeFirst.

- getPosition.

Dispone de los siguientes métodos para obtener los valores de los registros:

- getShort.
- getInt.
- getLong.
- getFloat.
- getDouble.
- getExtras.
- getString.
- getBlob.
- getType.

Otros métodos:

- Close. Cierra el cursor liberando completamente los recursos tomados
- Deactivate. Desactiva el cursor haciendo que fallen todas las llamadas al cursor hasta que se llame a requery. (Libera recursos) (DEPRECATED)
- Requery. Lanza de nuevo la consulta que creó el cursor, refrescando su contenido. (DEPRECATED)

23.6. Transacciones

SQLite, es una base de datos Transaccional, se pueden crear transacciones a partir del objeto **SQLiteDatabase**, que ofrece los siguientes métodos.

- beginTransaction(): Inicia una transacción.
- setTransactionSuccessful(): Marca la transacción para realizar el commit al finalizar.
- endTransaction(): Realiza el commit si se ejecuto setTransactionSuccessful, sino rollback en caso contrario.

```
db.beginTransaction();
try {
    db.setTransactionSuccessful();
} finally {
    db.endTransaction();
}
```

24. Content Providers

Representan una interface de acceso a un servicio de persistencia.

Especialemnte util si se tiene en cuenta las caracteristica privada de la base de datos de android.

Conceptualmente se comportan de la misma forma que los servicios RESTful.

El Sistema los emplea para proporcionar acceso a distintas entidades persistentes que se encuentran de forma nativa en los terminales android

- El historial de llamadas.
- La agenda de contactos y teléfonos.
- Las bibliotecas multimedia (audio y video).
- El historial y la lista de favoritos del navegador.

En el paquete **android.provider** se pueden encontrar clases de ayuda para acceder a los distintos Providers proporcionados por el sistema.

El metodo onCreate del Content Provider, se ejecuta antes, que el método onCreate del Application, por lo que ojo con definir referencias al Application en el onCreate de un Provider. Esto no es así en las Actividades donde el onCreate de las Actividades se ejecuta despues del onCreate del Application

25. Threads

Todos los componentes de una aplicación Android, tanto las Activity, los services o los BroadcastReceivers se ejecutan en el mismo hilo de ejecución Este hilo, también es el hilo donde se ejecutan todas las operaciones que gestionan la interfaz de usuario de la aplicación. Es llamado hilo principal, main thread o GUI thread.

Cualquier operación larga o costosa que realicemos en este hilo va a bloquear la ejecución del resto de componentes de la aplicación y por supuesto también la interfaz. Android monitoriza las operaciones realizadas en el hilo principal y detecta aquellas que superen los 5 segundos, en cuyo caso muestra el mensaje de “Application Not Responding” (ANR) y el usuario debe decidir entre forzar el cierre de la aplicación o esperar a que termine.

En Android hay dos formas de generar un hilo paralelo de ejecución donde se deberán ejecutar las tareas de larga duración:

- Thread.
- AsyncTask.

Para ejecutar un bloque de código en el hilo principal, estando ordenada dicha ejecución desde un hilo secundario, se cuenta en el API con **runOnUiThread**, que ejecuta el codigo del **runnable** que recibe por parametro en el **MainThread** o **UIThread**.

```
runOnUiThread(new Runnable() {
        public void run() {
    //Codigo a ejecutar en el hilo principal.
        }
});
```

Esta caracteristica es necesaria, ya que los componentes graficos, solo pueden ser accedidos por el **UIThread**.

25.1. AsyncTask

Dado que el manejo de la **UI** desde la ejecución de código en otros **Threads** no resulta muy comoda, el API, proporciona una clase **AsyncTask**, que facilita este trabajo.

Esta clase generica, obliga a definir 3 tipos cuando se la extiende.

```
public class MiTarea extends AsyncTask<Params, Progress, Result> {}
```

Donde:

- **Params**: Tipo para los parámetros de inicio
- **Progress**: Tipo para el parámetro que permite actualización de la UI según se progrese en la tarea que representa el **Thread**.
- **Result**: Tipo del dato a retornar al finalizar la tarea y que se deberá trasladar a la **UI**.

Al definir estos tipos, se puede emplear el tipo especial **Void** sino se va a emplear la funcionalidad a la que esta asociada el tipo.

Se ha de implmentar obligatoriamente el método **doInBackground**, que es el que ejecuta la tarea de larga duración, siendo el único método que se ejecutará en el nuevo **Thread**, por lo que no puede interacionar con los **View**.

```
@Override
protected Bitmap doInBackground(String... params) {
    //Realizar la tarea de larga duracion
    try {
        URL url = new URL(params[0]);

        HttpURLConnection con = (HttpURLConnection) url.openConnection();

        int tamañoImagen = con.getContentLength();
        byte[] imagen = new byte[tamañoImagen];
        byte[] buffer = new byte[1024];

        InputStream is = con.getInputStream();

        for(int bytesTotalesLeidos = 0; bytesTotalesLeidos < tamañoImagen; ){
            int bytesLeidos = is.read(buffer);

            System.arraycopy(buffer, 0, imagen, bytesTotalesLeidos, bytesLeidos);

            bytesTotalesLeidos += bytesLeidos;

            publishProgress(bytesTotalesLeidos*100/tamañoImagen);
        }

        return BitmapFactory.decodeByteArray(imagen, 0, tamañoImagen);

    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    return null;
}
```

Adicionalmente se pueden implementar

- **onPreExecute**. Se ejecuta antes de **doInBackground**.
- **onProgressUpdate**. Responde a invocaciones de **publishProgress** desde **doInBackground**.
- **onPostExecute**. Se ejecuta al terminar **doInBackground**.
- **onCancelled**. Se ejecutará cuando se cancela la tarea antes de su finalización normal, esto se consigue invocando el método **cancel** del objeto **AsyncTask**. También se ejecuta en el **MainThread**.

25.1.1. Inicialización

La tarea a realizar quedará representada por una instancia de la clase que extienda a **AsyncTask**, donde normalmente el objeto recibirá en los parametros del constructor aquellos **View** sobre los

que se quiera actual.

```
TareaAsincrona tareaAsincrona = new TareaAsincrona(dialog, imageView);
```

Donde para iniciar la tarea, habrá que ejecutar el método **execute**:

```
tareaAsincrona.execute("http://www.dieselstation.com/wallpapers/albums/McLaren/P1-Concept/mclaren-p1-concept-2012-widescreen-5.jpg");
```

Si se quiere realizar de nuevo la tarea, habra que volver a crear una instancia, dado que no se puede volver a ejecutar una ya finalizada.

Se puede conocer el estado en el que está la tarea

```
tareaAsincrona.getStatus().equals(AsyncTask.Status.FINISHED)
```

26. Conexiones Remotas

En la movilidad las conexiones remotas, son muy importantes, dado que la logica de negocio de las aplicaciones, no reside habitualmente en el terminal, sino en un servidor remoto, a traves del cual se tiene la posibilidad de compartir la información con otros terminales.

Para poder realizar conexiones a través de internet, se necesita dar permisos a la aplicación, una vez la aplicación tiene estos permisos podrá realizar peticiones a URL externas.

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

Desde la versión 3.0, no se permiten realizar conexiones en el hilo principal.

Si se trabaja con el emulador y se desea conectar con la maquina física, se ha de hacer referencia a la IP

```
10.0.2.2
```

26.1. API URLConnection

Se basa en los objetos

- **java.net.URL**. Que define un **endpoint** donde escucha un servicio.
- **java.net.URLConnection**. Que define cada una de las conexiones hacia el **endpoint**.

```
URL url = new URL(direccion);

URLConnection connection = url.openConnection();
```

26.1.1. HttpURLConnection

Sub API de **URLConnection**, que permite manejar peticiones empleando el protocolo de transporte HTTP.

```
URL url = new URL(direccion);

HttpURLConnection connection = (HttpURLConnection)url.openConnection();
```

Permite obtener el codigo HTTP de respuesta de la petición

```
int responseCode = connection.getResponseCode();
```

Teniendo definidas en la propia clase las siguientes constantes con los codigos posibles

- HttpURLConnection.HTTP_OK
- HttpURLConnection.HTTP_FORBIDDEN
- HttpURLConnection.HTTP_INTERNAL_ERROR
- HttpURLConnection.HTTP_NOT_FOUND
- HttpURLConnection.HTTP_UNAUTHORIZED
- HttpURLConnection.HTTP_UNAVAILABLE
- HttpURLConnection.HTTP_PROXY_AUTH

Permite establecer el **METHOD** a emplear

```
connection.setRequestMethod("POST");
```

26.1.2. Definición de un Proxy

Es posible que en el desarrollo con el emulador, al emplear una red empresarial, se necesite configurar la salida a traves de un proxy.

```
SocketAddress socketAddress = new InetSocketAddress(“host", port);

Proxy proxy = new Proxy(Type.HTTP,socketAddress);

URL url = new URL(direccion);

URLConnection connection = url.openConnection(proxy);

InputStream is = connection.getInputStream();
```

26.1.3. Descarga

La información que retorna el **endpoint** llega a la aplicación como un **java.io.InputStream**.

```
InputStream is = connection.getInputStream();
```

Sera importante cerrar tanto los Streams, como las conexiones.

26.1.4. Subida

La subida de datos se realiza a traves de un **java.io.OutputStream**.

```
//Se indica que se va a subir un recurso al endpoint
connection.setDoOutput(true);

//Mejora de rendimiento cuando se conoce el tamaño del recurso a subir
connection.setFixedLengthStreamingMode(int); //Tamaño conocido

//Mejora de rendimineto cuando no se conoce el tamaño total del recurso a subir
connection.setChunkedStreamingMode(0); //Tamaño desconocido

connection.getOutputStream();
```

26.2. Streams

Repasemos como manejar los Stream, veamos como realizar una lectura progresiva del Stream que retorna un **endpoint**.

```
URL url = new URL("http://www.google.es");

HttpURLConnection con = null;

try{
    //Se abre la conexion
    con = (HttpURLConnection) url.openConnection();

    //Se obtiene el tamaño en bytes del recurso
    int tamaño = con.getContentLength();

    //Se crea un Array de Bytes, que contendrá el resultado de la transferencia
    byte[] resultado = new byte[tamaño];

    //Se crea un Array de Bytes, que representa el buffer de lectura, es decir, no se
lee el fichero entero, sino que
    //se lee a trozos de forma progresiva
    byte[] buffer = new byte[1024];

    //Se abre el Stream sobre la conexion
    InputStream is = con.getInputStream();

    //Variable temporal, para conocer mientras se lee el recurso, el numero de bytes
leidos en total
    int bytesTotalesLeidos = 0;

    //Lectura del Stream empleando el buffer, se leeran de 1024 en 1024 los bytes,
hasta que no queden mas, momento en el
    //que el método read, retornará -1, pudiendo retornar en la anterior lectura un
numero de butes leidos inferior a 1024
    while ((int bytesLeidosIteracion = input.read(buffer)) != -1) {

        //Copia de los bytes leidos en la iteracion, que se encuentra en buffer, a
resultado, empezando a copiar los bytes de
        //buffer en 0, y poniendoles en resultado a partir de bytesTotalesLeidos,
copiando un numero de bytes en total igual a
        //bytesLeidosIteracion.
        System.arraycopy(buffer, 0, resultado, bytesTotalesLeidos,
bytesLeidosIteracion);

        //Actualización del numero de bytes leidos
        bytesTotalesLeidos += bytesLeidosIteracion;

        //Publicación del grado de progreso en %
        publishProgress(bytesTotalesLeidos*100/tamañoImagen);
    }
} finally {
    con.disconnect();
}
```

En el trabajo con Arrays, se puede hace uso de las herramientas del sistema, como las clase Arrays o como en este caso la clase System.

26.3. API de HttpClient

API de la **Apache Software Fundation**, que permite facilitar el trabajo con las conexiones.

No se encuentra en la JRE de java, es un añadido en el SDK de Android.

No es el API recomendada.

Se basa en los objetos

- **org.apache.http.client.HttpClient**. La plataforma para realizar la petición.
- **org.apache.http.client.methods.HttpGet**. La petición a realizar con el Method GET de HTTP.
- **org.apache.http.client.methods.HttpPost**. La petición a realizar con el Method POST de HTTP.
- **org.apache.http.HttpResponse**. La respuesta.
- **org.apache.http.HttpEntity**. El cuerpo de la respuesta.

```
HttpClient cliente = new DefaultHttpClient();

HttpPost post = new HttpPost("http://www.google.es");

HttpResponse respuesta = cliente.execute(post);

String cuerpo = EntityUtils.toString(respuesta.getEntity());
```

26.3.1. Proxy

```
DefaultHttpClient httpclient = new DefaultHttpClient();

HttpHost proxy = new HttpHost("someproxy", 8080);

httpclient.getParams().setParameter(ConnRoutePNames.DEFAULT_PROXY, proxy);
```

26.4. Proxy del Sistema

Se puede emplear el proxy del sistema, que se obtiene desde código como

```
System.getProperty("http.proxyHost");
System.getProperty("http.proxyPort");

System.getProperty("https.proxyHost");
System.getProperty("https.proxyPort");
```

Y se establece con los métodos set correspondientes.

27. JSON

Para trabajar con JSON en Android, se ofrece un API nativo basado en objetos de tipo **JSONObject** y **JSONArray**.

La idea es construir a partir de un string que representa un JSON

```
{
   "personas": [
      { "nombre":"Juan" , "edad":31 },
      { "nombre":"Maria" , "edad":22 },
      { "nombre":"Pedro" , "edad":53 }
    ]
}
```

Un **JSONObject** o un **JSONArray**, esto el api lo hace directamente.

```
String json = "{'personas': [..........]}";

JSONObject object = new JSONObject(json); //Objeto JSON con una unica caracteristica
"personas"

JSONArray json_array = object.optJSONArray("personas");//El Array de personas en
formato JSONArray

JSONObject item = json_array.getJSONObject(0);//La primera persona en formato
JSONObject

String valor = item.getString("nombre");//El nombre de la persona
```

Estos objetos son genericos y no se adaptan al modelo empleado por las aplicaciones, por lo que habrá que transformarlos

Para la clase de modelo siguiente

```
public class Persona {

    public String nombre;

    public int edad;

    public Persona (String nombre, int edad){
        this.nombre = nombre;
        this.edad = edad;
    }

    // getter y setter
}
```

```
public class JSONUtils {
    public static Persona createPersonaFromJson(JSONObject json){
        return new Persona(
            json.getString("nombre"),
            json.getInt("edad")
        );
    }
}
```

27.1. GSON

API que se puede obtener desde los repositorio de Maven

```
compile 'com.google.code.gson:gson:2.3.1'
```

Facilita el proceso de transformación de JSON a objetos del modelo

```
String json = "{'personas': [..........]}";

Gson gson = new Gson();

List<Persona> personas = gson.fromJson(json, Persona.class);
```

28. Parseo de XML

Existen en el SDK varios APIs para el procesamiento de XML

- **SAX** (Simple Api for Xml)
- **DOM** (Document Object Model)

- **StAX** (Streaming Api for Xml)
- **XmlPull** (Analoga a StAX)

El empleo de un API u otro, dpenderá de lo que se quiera hacer con el XML, si se pretende transformar en objetos de dominio, se deberan emplear SAX, StAX o XmlPull, si en cambio se desean hacer repetidas busquedas de información puntual sobre el XML, es decir no importa toda la información del XML, sino solo partes, es recomendable emplear DOM.

28.1. SAX

El API de **SAX**, se basa en dos componentes

- **Parser**. Se encarga de ir leyendo el XML de forma secuencial, y va generando los siguientes eventos
  - **startDocument**. comienza el documento.
  - **endDocument**. termina el documento.
  - **startElement**. comienza una etiqueta.
  - **endElement**. termina una etiqueta.
  - **characters**. se encuentra una linea de texto entre etiquetas.

```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser parser = factory.newSAXParser();
```

- **Handler**. Es el que procesa los eventos que genera el **Parser**, es donde reside la logica de parseo, donde se decide que hacer con la información encontrada en el XML.

Habrá que asociar una instancia al **Parser**

```
RssHandler handler = new RssHandler();
parser.parse(this.getInputStream(), handler);
```

Y habrá que definir el procedimiento de parseo en una clase que herede de **DefaultHandler**.

```
public class RssHandler extends DefaultHandler {}
```

Es habitual que se incluya en esta nueva clase un atributo de clase y su correspondiente getter con el resultado del procesamiento.

```
public class RssHandler extends DefaultHandler {
    private List<Noticia> noticias;
    public List<Noticia> getNoticias(){
        return noticias;
    }
}
```

Se deberan implementar los manejadores de los eventos que lanza el **Parser**

- **startDocument**. Permite inicializar atributos de clase, para que en el nuevo parseo este todo en estado inicial, y no se encuentren referencias a null o datos de antiguos procesamientos.

Se suelen inicializar el resultado del procesamiento y un **StringBuilder** que se emplea para acumular los caracteres leidos entre etiquetas.

```
@Override
public void startDocument() throws SAXException {
    super.startDocument();
    noticias = new ArrayList<Noticia>();
    sbTexto = new StringBuilder();
}
```

- **startElement**. Permite inicializar/resetear el estado de atributos temporales reutilizadas en varias partes del parseo.

Se suelen resetear atributos asociados a los item repetidos en el XML

```
private Noticia noticiaActual = null;

@Override
public void startElement(String uri, String localName, String name, Attributes
attributes) throws SAXException {
    super.startElement(uri, localName, name, attributes);
    if (localName.equals("item")) {
        noticiaActual = new Noticia();
    }
}
```

- **characters**. Permite acumular los caracteres entre etiquetas. Se hace uso del **StringBuilder** para acumular caracteres.

A tener en cuenta que no solo se acumulan los caracteres entre etiquetas de inicio y fin, que son los que representan en un XML la información, sino que tambien se acumulan los que aparecen entre etiquetas de inicio consecutivas y entre etiquetas de fin y de inicio, normalmente en estos casos serán caracteres de espaciado o saltos de linea.

```
@Override
public void characters(char[] ch, int start, int length) throws SAXException {
    super.characters(ch, start, length);
    if (this.notciaActual != null) {
        builder.append(ch, start, length);
    }
}
```

- **endElement**. Permite asignar la información acumulada en **StringBuilder** al objeto correspondiente. Para conocer cual es la información que contiene el **StringBuilder**, se acude a interpretar la etiqueta, para ello se emplea el atributo **localName**.

```
@Override
public void endElement(String uri, String localName,
            String name) throws SAXException {
    super.endElement(uri, localName, name);
    if (this.noticiaActual != null) {
        if (localName.equals("title")) {
                        noticiaActual
                .setTitulo(sbTexto.toString());
        } else if (localName.equals("item")) {
                        noticias.add(noticiaActual);
                }
        sbTexto.setLength(0);
    }
}
```

Una vez que se ha extraido la información del **StringBuilder**, este se suele vaciar, para ello

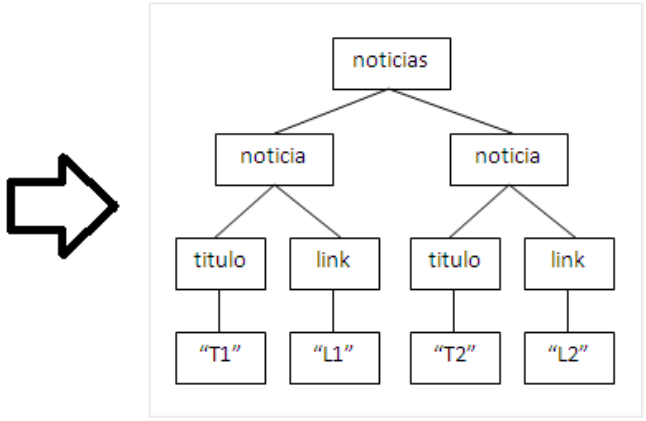
```
sbTexto.setLength(0);
```

28.2. DOM

A diferencia que con **SAX**, con **DOM**, el documento XML se lee completamente antes de poder realizar ninguna acción con su contenido.

Se crea una estructura de árbol, donde los distintos elementos del XML se representa en forma de **nodos** y su jerarquía padre-hijo se establece mediante relaciones entre dichos nodos.

```
<noticias>
    <noticia>
        <titulo>T1</titulo>
        <link>L1</link>
    </noticia>
    <noticia>
        <titulo>T2</titulo>
        <link>L2</link>
    </noticia>
<noticias>
```



DOM, se basa en el Parseo total del fichero, obteniendo un objeto **Document**.

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();

Document dom = builder.parse(this.getInputStream());
```

Una vez parseado el documento, se puede acceder a la informacióncon métodos como.

- **getDocumentElement**. Retorna el nodo principal.
- **getElementsByTagName**. Retorna un **NodeList** con los nodos que tengan como nombre el parámetro pasado.
- **getChildNodes**. Retorna un NodeList con todos los nodos hijos.
- **getFirstChild**. Retorna el primer nodo hijo.

El texto contenido por una etiqueta, se almacena en el árbol como un nodo hijo del nodo que representa la etiqueta, por lo que para accederlo se hará

```
String texto = nodo.getFirstChild().getNodeValue();
```

28.2.1. XPath

API que a partir de objetos **Document**, permite realizar búsquedas de nodos a traves de expresiones.

Se basa en un evaluador de expresiones

```
XPath xPath = XPathFactory.newInstance().newXPath();
```

Al cual se le va pidiendo que evalue las expresiones en formato **String**

```
NodeList nodes = (NodeList)xPath.evaluate("/feed/entry", doc.getDocumentElement(),
XPathConstants.NODESET);

Node id = (Node)xPath.evaluate("id", element, XPathConstants.NODE);
```

Donde las expresiones estarán formadas por los siguientes elementos

Expression	Description
nodename	Selecciona todos los nodos con nombre de etiqueta “nodename” a partir del nodo actual
/	Selecciona a partir del nodo principal, hay que indicar toda la ruta
//	Selecciona los nodos sin importar donde se ubican en el documento
.	Selecciona el nodo actual
..	Selecciona el padre del nodo actual
@	Selecciona en funcion de los atributos
/feed/entry[1]	Selecciona el primer nodo "entry" que sea hijo del nodo principal "feed"
/feed/entry[last()]	Selecciona el ultimo nodo "entry" que sea hijo del nodo principal "feed"
/feed/entry[last()-1]	Selecciona el penultimo nodo "entry" que sea hijo del nodo principal "feed"
//link[@rel='self']	Selecciona el nodo "link", este donde este, que tenga como valor "self" en el atributo "rel"

Tomando el siguiente XML como ejemplo

```
<feed xmlns:georss="http://www.georss.org/georss" xmlns="http://www.w3.org/2005/Atom">
    <title>USGS All Earthquakes, Past Hour</title>
    <updated>2015-02-06T18:14:55Z</updated>
    <author>
        <name>U.S. Geological Survey</name>
        <uri>http://earthquake.usgs.gov/</uri>
    </author>
    <id>http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_hour.atom</id>
    <link href="
http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_hour.atom"
        rel="self" />
    <icon>http://earthquake.usgs.gov/favicon.ico</icon>
    <entry>
        <id>urn:earthquake-usgs-gov:ci:37313544</id>
        <title>M 1.8 - 6km NW of Mira Loma, California</title>
        <updated>2015-02-06T18:14:28.710Z</updated>
        <link href="http://earthquake.usgs.gov/earthquakes/eventpage/ci37313544" rel=
"alternate"
            type="text/html" />
        <summary type="html">
            <![CDATA[<dl><dt>Time</dt><dd>2015-02-06 18:03:39 UTC</dd><dd>2015-02-06
10:03:39 -08:00 at epicenter</dd><dt>Location</dt><dd>34.033&deg;N
117.563&deg;W</dd><dt>Depth</dt><dd>9.03 km (5.61 mi)</dd></dl>]]></summary>
        <georss:point>34.033 -117.5626667</georss:point>
        <georss:elev>-9030</georss:elev>
        <category label="Age" term="Past Hour" />
        <category label="Magnitude" term="Magnitude 1" />
    </entry>
</feed>
```

Aqui vemos algunos ejemplos de expresiones

```
//Seleccion de todos los nodos "entry"
"entry"

//Seleccion de todos los nodos "entry" dentro de un nodo "feed"
"feed/entry"

//Seleccion de todos los nodos entry, sin importar su ubicacion en el documento
"//entry"
```

28.3. XMLPullParser

Framework similar a **SAX**, como diferencia esta que permite controlar cuando se pasa al siguiente estado, e incluso terminar el parseo de forma controlada.

Otra diferencia con SAX, es que desde la etiqueta de apertura, se puede obtener el texto continuado dentro de la etiqueta, ya no hace falta acumular y recoger en la etiqueta de finalizacion.

Se basa en un **Parser**, que permite el cambio de estado

```
XmlPullParser parser = Xml.newPullParser();
```

Para obtener el estado inicial, se hará

```
int evento = parser.getEventType();
```

Para continuar con el parseo, moviendo el procesamineto al siguiente estado, se hará

```
event = xmlPullParser.next();
```

Los distintos estados por lo que puede pasar son

- **XmlPullParser.START_DOCUMENT**. Estado inicial, indica que se esta procesando la primera etiqueta del XML, la etiqueta de inicio.
- **XmlPullParser.END_DOCUMENT**. Indica que se ha llegado al final del documento.
- **XmlPullParser.START_TAG**. Indica que se ha llegado a una etiqueta de inicio.
- **XmlPullParser.END_TAG**. Indica que se ha llegado a una etiqueta de fin.

El procedimineto por tanto cambia con respecto a SAX, ya que ahora hay que controlar cuando se pasa al siguiente estado y no hará falta acumular caracteres.

```
LinkedList<Terremoto> resultado = new LinkedList<>();

XmlPullParser xmlPullParser = Xml.newPullParser();

xmlPullParser.setInput(is,"UTF-8");

int event = xmlPullParser.getEventType();

Terremoto terremoto = null;

while(event != XmlPullParser.END_DOCUMENT){
    String tag = xmlPullParser.getName();

    if (event == XmlPullParser.START_TAG) {
        if (tag.equals("entry")) {
            terremoto = new Terremoto();
        } else if (tag.equals("id") && terremoto != null) {
            terremoto.setId(xmlPullParser.nextText());
        } else if (tag.equals("title") && terremoto != null) {
            String titulo = xmlPullParser.nextText();
            terremoto.setTitulo(titulo);
            terremoto.setMagnitud(Float.valueOf(titulo.split(" ")[1]));
        } else if (tag.equals("updated") && terremoto != null) {
            DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");
            Date date = dateFormat.parse(xmlPullParser.nextText());
            terremoto.setFecha(date);
        } else if (tag.equals("link") && terremoto != null) {
            terremoto.setLink(xmlPullParser.getAttributeValue(null, "href"));
        } else if (tag.equals("point") && terremoto != null) {
            String[] latlon = xmlPullParser.nextText().split(" ");
            terremoto.setLatitud(Float.valueOf(latlon[0]));
            terremoto.setLongitud(Float.valueOf(latlon[1]));
        }
    } else if (event == XmlPullParser.END_TAG){
        if (tag.equals("entry")) {
            resultado.add(terremoto);
            terremoto = null;
        }
    }
    event = xmlPullParser.next();
}

return resultado;
```

29. Broadcast Receiver

Se pueden considerar los Broadcast, como eventos tratados por contexto.

Y los Broadcast Receiver como sus Listener.

Pensados para poder escuchar eventos del sistema de forma desacoplada, a traves de las intenciones.

Por ejemplo se puede configurar que se abra una Actividad propia ante el evento del sistema de llamada de teléfono entrante.

Como son objetos manejados por el contexto, es necesario que sean declarados, esta declaración que para el resto de elementos manejados, se realiza en el AndroidManifest, en el caso de Broadcast, se puede realizar así, o de forma programática.

Registro de un Broadcast Receiver de forma programática.

```
context.registerReceiver(finalizar, new IntentFilter("paquete.ACCION"));
```

Registro de un Broadcast Receiver de forma declarativa en XML.

```
<receiver android:name="MiReceiver" >
    <intent-filter>
                <action android:name="paquete.ACCION" />
    </intent-filter>
</receiver>
```

A partir de la versión Android 3.1 el sistema excluye todo BroadcastReceiver de recibir Intent si la aplicación nunca ha sido iniciada por el usuario o si el usuario explícitamente detuvo la aplicación.

Aunque principalmente se emplearán para escuchar eventos del sistema, se pueden lanzar dichos eventos a traves de intenciones

Invocación de un Broadcast.

```
Intent cerrar = new Intent("paquete.ACCION");
sendBroadcast(cerrar);
```

La clase BroadCastReceiver es Abstracta, obligando a sus descendientes a implementar el método

- onReceive

Registro de un Broadcast Receiver de forma programática.

```
public class MyReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context,
                    Intent intent) {
        Toast.makeText(context, "Evento recogido 2",                Toast.
LENGTH_SHORT).show();
    }
}
```

29.1. Evento del Sistema: Arranque completado

Uno de los eventos del sistema que mas habitualmente se desea escuchar es el del arranque del terminal.

Este evento se puede escuchar con un BrodcastReceiver, que procese Intents con Action **android.intent.action.BOOT_COMPLETED**.

Declaración del Receiver.

```
<receiver android:name=".BootCompletedIntentReceiver">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
```

Para que el Receiver responda a los eventos del sistema de BOOT_COMPLETED, se ha de iniciar al menos una vez la aplicación, hasta que esto no sucede el Receiver no estará activo

Este evento del sistema necesitará los permisos de

Declaración de permiso necesario.

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

29.2. Sticky Broadcast Intents

Cuando se envía una intención a un BroadCastReceiver, con el método sendBroadCast(), la intención desaparece en cuanto se termina de procesar.

En cambio si se hace con sendStickyBroadcast(), la intención perdura y es accesible a través de la referencia que retorna el método registerReceiver().

Un ejemplo de uso de estos tipos de Intents, es la consulta del estado de la batería

Consulta del estado de la batería.

```
IntentFilter filter = new                          IntentFilter(Intent
.ACTION_BATTERY_CHANGED);
Intent batteryStatus = this.registerReceiver(null, filter);
int status = batteryStatus.getIntExtra(
            BatteryManager.EXTRA_STATUS, -1);
boolean isCharging = status ==                     BatteryManager
.BATTERY_STATUS_CHARGING ;
boolean isFull = status ==                         BatteryManager.
BATTERY_STATUS_FULL;
int chargePlug = batteryStatus.getIntentExtra(
            BatteryManager.EXTRA_PLUGGED, -1);
boolean usbCharge = chargePlug ==                      BatteryManager
.BATTERY_PLUGGED_USB;
boolean acCharge = chargePlug ==                       BatteryManager
.BATTERY_PLUGGED_AC;
```

30. Google Play Service

Es un API de Google, en el que se reunen todos los APIs que Google proporciona para el desarrollo en Android.

Se encuentra instalado de forma nativa en los dispositivos a partir de la version 2.2

En el emulador, hay que incluirlo, usando el runtime con las Google APIs.

Para el desarrollo habra que incluir la libreria correspondiente desde el menú **proyectoectStructure/<modulo>/Dependencies**

En este caso el modulo a añadir es

```
dependencies {
    compile 'com.google.android.gms:play-services:+'
}
```

Una vez añadida la libreria, en el fichero **AndroidManifest**, se deberá añadir la declaración de la versión empleada de Google Play Service. Se hará dentro del nodo Application

Declaración de la version de Google Play Service

```
<application>
    .
    .
    .
    <meta-data
        android:name="com.google.android.gms.version"
        android:value="@integer/google_play_services_version"/>
    .
    .
    .
</application>
```

31. Google Maps

Es uno de los APIs que proporciona Google dentro de las **Google Play Services**, por tanto para poder emplearlo, será necesario incluir las **Google Play Services** al proyecto.

Para su uso, es necesario obtener una clave de la consola de de Google APIs.

31.1. Obtención del API Key

Acceder a la consola de APIs de Google aquí

Allí se deberá crear un proyecto nuevo que asociaremos con la aplicación Android.

[formulario creacion proyecto consola google] | *formulario_creacion_proyecto_consola_google.png*

En la ventana del proyecto que se abre a continuación, habra que seleccionar y activar el API, para ello acceder al menú **APIS y autenticación/APIs/Google Maps Android API**

[apis y autenticacion] | *apis_y_autenticacion.png*

Una vez activada el API, se procede a registrar la aplicación, para ello seleccionar el menú **APIS y autenticación/Credenciales/Añadir Credenciales/Clave del API/Clave de Android**

En el formulario para **crear clave de API de Android**, se pedirá un nombre de paquete de la aplicación, que es el definido en el **AndroidManifest**.

Paquete en AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package=
"com.curso.android.terremotos" >
```

Y tambien la **huella digital SHA1** asociada con el **certificado digital** empleado para firmar el APK de la aplicación Android.

Este certificado digital, se encuentra configurado en **Android Studio** en el menú de **Project**

Structure/app/Signing

De no haberse definido ninguno, se estará empleando el por defecto de **Android Studio**

El certificado de **Android Studio**, no será aceptado por el **Market** cuando se publique la aplicación, asi que puede ser buen momento para cambiarlo por uno nuevo.

31.2. Certificado Digital

Para crear un **certificado digital**, se puede emplear el **Android Studio**, en el menú **Build/Generated Signed APK**

Se puede seleccionar la creación de uno nuevo, para lo cual habra que indicar * **Ruta** (path) * **Key Store password** * **Key password** * **Key validity** * **Nombre** * **Organización** * **Localidad** * **Provincia/Estado** * **Codigo Pais**

Una vez generado, se puede añadir a la configuración en el menú anterior **Project Structure/app/Signing**

Para obtener la huella digital SHA1, se puede emplear un comando que se incluye con la DJK de java.

```
keytool -list -v -keystore <path del fichero del certificado>
```

El comando pedirá el password y mostrara las huellas, entre ellas la SHA1 que nos pide la consola de Google.

31.3. Configuración del proyecto

En el **AndroidManifest.xml** habra que incluir el **API key** obtenido anteriormente. Se hará dentro del nodo **Application**.

Declaración del API KEY de Google.

```
<application>
    .
    .
    .
    <meta-data
        android:name="com.google.android.maps.v2.API_KEY"
        android:value="valor del API KEY"/>
    .
    .
    .
</application>
```

Se han de incluir adicionalmente una serie de permisos

Declaración de los permisos de Google.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission    android:name=
"com.google.android.providers.gsf.permission.READ_GSERVICES"/>
```

La version 2 del API de Google Maps, emplea OpenGL v2, lo cual habrá que indicarlo

Declaración de OpenGL v2.

```
<uses-feature
    android:glEsVersion="0x00020000"
    android:required="true"/>
```

31.4. Creación del Mapa

Para incluir un mapa en el proyecto, se hará a traves de un fragmento de tipo **MapFragment**, el cual se puede extender o insertar directamente en la Actividad que represente el mapa

Declaración en un layout de actividad del fragmento de tipo MapFragment

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:name="com.curso.android.terremotos.TerremotoMapFragment"
        android:id="@+id/map" />
```

Para dar soporte a versiones inferiores a la 2.2, se ha de emplear el API de soporte, siendo la clase a emplear com.google.android.gms.maps.SupportMapFragment

Extension de la clase MapFragment

```
public class TerremotoMapFragment extends MapFragment {...}
```

31.4.1. El Mapa

Una vez incluido el Fragmento del mapa, este tiene un metodo que da acceso al objeto GoogleMap

Obtención del objeto GoogleMap del fragmento

```
GoogleMap mapa = ((MapFragment) getFragmentManager().findFragmentById(R.id.map))
.getMap();
```

Una de las primeras tareas de confguración, será establecer el tipo de mapa a visualizar. Teniendo las siguientes opciones

- **MAP_TYPE_NORMAL**
- **MAP_TYPE_HYBRID**
- **MAP_TYPE_SATELLITE**
- **MAP_TYPE_TERRAIN**

Establecimineto del tipo de mapa

```
mapa.setMapType(GoogleMap.MAP_TYPE_NORMAL);
```

Repositorio de GitHub, con ejemplo de Google Maps para Android aquí

31.4.2. Movimientos en el Mapa

Se puede mover el mapa empleando los objetos

- **CameraUpdate**
- **CameraPosition**

El primero **CameraUpdate**, permite modificar los siguientes parametros del punto de vista actual del mapa, con los siguientes métodos

- **newLatLong()**. Nuevo centro
- **newLatLongBounds()**. Nuevo centro, indicando el recuadro a mostrar.
- **newLatLongZoom**. Nuevo centro indicando el nivel de zoom.
- **zoomIn**. Acercar un nivel de zoom.
- **zoomOut**. Alejar un nivel de zoom.
- **zoomTo**. Establecer un nuevo nivel de zoom.
- **newCameraPosition**. Actualziación del punto de vista empleando un objeto **CameraPosition**.

La creación de estos objetos de hace a traves de **CameraUpdateFactory**

Establecimiento de atributos principales

```
CameraUpdate camUpd1 = CameraUpdateFactory.zoomIn();
camUpd1 = CameraUpdateFactory.zoomOut();
camUpd1 = CameraUpdateFactory.zoomTo(nivel);
camUpd1 = CameraUpdateFactory.newLatLng(lat, long);
camUpd1 = CameraUpdateFactory.newLatLngZoom(lat, long, zoom);
camUpd1 = CameraUpdateFactory.scrollBy(scrollHorizontal, scrollVertical);
```

El objeto **CameraPosition**, permite indicar los siguientes parametros al punto de vista del mapa.

- **latitud-longitud**.
- **zoom**.
- **bearing**. Orientación (N-S-E-O)
- **tilt**. Angulo de visión o inclinación.

Establecimiento de atributos principales y secundarios

```
LatLng madrid = new LatLng(40.417325, -3.683081);
CameraPosition camPos = new CameraPosition.Builder()
    .target(madrid)
    .zoom(19)
    .bearing(45)
    .tilt(70)
    .build();
 CameraUpdate camUpd3 = CameraUpdateFactory.newCameraPosition(camPos);
```

La aplicación de un **CameraPosition** al mapa, se hace a traves del **CameraUpdate**, puediendo aplicarse de dos formas

- Animando
- Moviendo

Aplicar atributos al mapa

```
mapa.animateCamera(camUp1);
mapa.moveCamera(camUp1);
```

31.4.3. Eventos en el Mapa

Se definen una serie de eventos que pueden producirse sobre el mapa y sus correspondientes Listener

- **onMapReady** ⇒ **OnMapReadyCallback** ⇒ Permite escuchar cuando el mapa esta disponible para el uso.

Registro de actividad como listener

```
SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
.findFragmentById(R.id.map);
mapFragment.getMapAsync(this);
```

Implementación de Listener

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;

    // Add a marker in Sydney and move the camera
    LatLng sydney = new LatLng(-34, 151);
    mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));
    mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
}
```

- **onMapClick** ⇒ **onMapClickListener** ⇒ Permite escuchar cuando se hace click sobre el mapa

Definición de Listener

```
mapa.setOnMapClickListener(new OnMapClickListener() {
    public void onMapClick(LatLng point) {
        Projection proj = mapa.getProjection();
        Point coord = proj.toScreenLocation(point);

        Toast.makeText(
            MainActivity.this,
            "Click\n" +
            "Lat: " + point.latitude + "\n" +
            "Lng: " + point.longitude + "\n" +
            "X: " + coord.x + " - Y: " + coord.y,
            Toast.LENGTH_SHORT).show();
    }
});
```

- **onMapLongClick** ⇒ **onMapLongClickListener**
- **onCameraChange** ⇒ **onCameraChangeListener**
- **onMarkerClick** ⇒ **onMarkerClickListener**

Se definen dos tipos de datos que representan las posiciones en el mapa **LatLong** y **Point**, proporcionando la clase **Projection** algunas herramientas para la traducción

Conversion de LatLong a Point

```
LatLng point = .....;
Projection proj = mapa.getProjection();
Point coord = proj.toScreenLocation(point);
```

31.4.4. Herramientas de navegacion

El mapa dispone de una serie de herramientas de navegacion uqe pueden ser habilitadas, para ello se accede a un objeto **UiSettings**.

```
---
UiSettings uiSettings = mMap.getUiSettings();
---
```

El objeto **UiSettings** dispone de varios métodos para la habilitación de los controles de navegación, como son:

- **setZoomControlsEnabled()**
- **setCompassEnabled()**
- **setMapTollbarEnabled()**
- **setMyLocationEnabled()**
- **setAllGesturesEnabled()**
- **setRateGesturesEnabled()**
- **setScrollGesturesEnabled()**
- **setTiltGesturesEnabled()**
- **setZoomGesturesEnabled()**

31.4.5. POIs Mapa

Los puntos de información (POI) se representan con la clase **MarkerOptions**, a la que se le puede definir

- **position**
- **icon**
- **title**
- **anchor**

Creación y adición de una marca al mapa

```
mapa.addMarker(new MarkerOptions()
        .position(new LatLng(lat, lng))
        .title("Pais: España"));
```

Las marcas se pueden mover, para ello, se ha de activar la caracteristica **draggable**

```
centerMarker = mMap.addMarker(new MarkerOptions().position(center).draggable(true));
```

Los iconos se crearán a partir de la clase **BitmapDescriptorFactory**.

```
radiusMarker = mMap.addMarker(new MarkerOptions().position(toRadiusLatLng(center,
radius)).icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_AZURE)
));
```

31.4.6. GroundOverlay

Son marcas, que se ven afectadas por la orientación, iclinacion y rotación del mapa, estan fijas sobre la superficie.

```
LatLng NEWARK = new LatLng(40.714086, -74.228697);

GroundOverlayOptions newarkMap = new GroundOverlayOptions()
        .image(BitmapDescriptorFactory.fromResource(R.drawable.newark_nj_1922))
        .position(NEWARK, 8600f, 6500f);
map.addGroundOverlay(newarkMap);
```

31.4.7. Figuras geometricas

Se pueden dibujar figuras geometricas en el mapa empleando las clases **PolygonOptions**, **PolylineOptions**, **CircleOptions**

Definición de una linea

```
PolylineOptions lineas = new PolylineOptions()
    .add(new LatLng(45.0, -12.0))
    .add(new LatLng(45.0, 5.0))
    .add(new LatLng(34.5, 5.0))
    .add(new LatLng(34.5, -12.0))
    .add(new LatLng(45.0, -12.0));

lineas.width(8);
lineas.color(Color.RED);

mapa.addPolyline(lineas);
```

Definición de un poligono

```
PolygonOptions rectangulo = new PolygonOptions().add(
        new LatLng(45.0, -12.0),
        new LatLng(45.0, 5.0),
        new LatLng(34.5, 5.0),
        new LatLng(34.5, -12.0),
        new LatLng(45.0, -12.0));

rectangulo.strokeWidth(8);
rectangulo.strokeColor(Color.RED);

mapa.addPolygon(rectangulo);
```

Definicion de un circulo

```
Circle circle = mMap.addCircle(new CircleOptions().center(center).radius(radius)
.strokeWidth(mWidthBar.getProgress()).strokeColor(mStrokeColor).fillColor(mFillColor))
;
```

32. Servicios REST

Los servicios REST son servicios basados en recursos, montados sobre HTTP, donde se da significado al Method HTTP.

La palabra REST viene de

- **Representacion**: Permite representar los recursos en multiples formatos, aunque el mas habitual es JSON.
- **Estado**: Se centra en el estado del recurso y no en las operaciones que se pueden realizar con el.
- **Transferencia**: Transfiere los recursos al cliente.

Los significados que se dan a los Method HTTP son:

- **POST**: Permite crear un nuevo recurso.
- **GET**: Permite leer/obtener un recurso existente.
- **PUT** o **PATCH**: Permiten actualizar un recurso existente.
- **DELETE**: Permite borrar un recurso.

Existen varios apis que permiten simplificar el desarrollo de clientes de servicios REST para Android

- **Retrofit**
- **Spring Android Rest Template**

32.1. Spring Andrid REST Template

Para poder emplear el API, hay que añadir las dependencias

```
compile 'org.springframework.android:spring-android-rest-template:1.0.1.RELEASE'
compile 'com.fasterxml.jackson.core:jackson-databind:2.3.2'
```

El uso del API se basa en la clase **RestTemplate**

```
RestTemplate restTemplate = new RestTemplate();
```

Será necesario declarar que **HttpMessageConverter** se desean emplear.

```
restTemplate.getMessageConverters().add(new MappingJackson2HttpMessageConverter());
```

Una vez configurado como se ha de realizar el intercambio de información, basta con hacer uso del API de **RestTemplate**

32.1.1. HttpMessageConverter

Son los encargados de procesar objetos a su representacion en JSON, XML, etc...

Tienen en cuenta la cabecera **Accept** de la petición HTTP para la respuesta y la cabecera **Content-Type** para los datos que llegan en el cuerpo de la petición.

Por defecto se tienen:

- **ByteArrayHttpMessageConverter** → convierte los arrays de bytes
- **StringHttpMessageConverter** → convierte las cadenas de caracteres
- **ResourceHttpMessageConverter** → convierte a objetos **org.springframework.core.io.Resource** desde y hacia cualquier **Stream**.
- **SourceHttpMessageConverter** → convierte a **javax.xml.transform.Source**
- **FormHttpMessageConverter** → convierte datos de formulario (application/x-www-form-urlencoded) desde y hacia un **MultiValueMap<String, String>**.
- **Jaxb2RootElementHttpMessageConverter** → convierte objetos Java desde y hacia XML, con media type **text/xml** o **application/xml** (solo sí la librería de JAXB2 está presente en el classpath).

```
compile 'com.fasterxml.jackson.dataformat:jackson-dataformat-xml:2.5.3'
```

- **MappingJackson2HttpMessageConverter** → convierte objetos Java desde y hacia JSON (solo sí la librería de Jackson2 está presente en el classpath).

```
compile 'com.fasterxml.jackson.core:jackson-databind:2.5.3'
```

- **MappingJacksonHttpMessageConverter** → convierte objetos Java desde y hacia JSON (sólo si la librería de Jackson está presente en el classpath).
- **AtomFeedHttpMessageConverter** → convierte objetos Java del tipo **Feed** que proporciona la librería Rome desde y hacia feeds Atom, media type **application/atom+xml** (solo si si la librería Roma está presente en el classpath).
- **RssChannelHttpMessageConverter** → convierte objetos Java del tipo **Channel** que proporciona la librería Rome desde y hacia feeds RSS (sólo si si la librería Roma está presente en el classpath).

32.1.2. Personalizar el Mapping de la entidad

En transformaciones a XML o JSON, de querer personalizar el Mapping de la entidad retornada, se puede hacer empleando las anotaciones de JAXB, como son **@XmlRootElement**, **@XmlElement** o **@XmlAttribute**.

32.1.3. RestTemplate

Las operaciones que se pueden realizar con RestTemplate son

- **Delete** → Realiza una petición DELETE HTTP en un recurso en una URL especificada

```
public void deleteSpittle(long id) {
    RestTemplate rest = new RestTemplate();
    rest.delete(URI.create("http://localhost:8080/spittr-api/spittles/" + id));
}
```

- **Exchange** → Ejecuta un método HTTP especificado contra una URL, devolviendo un ResponseEntity que contiene un objeto mapeado del cuerpo de respuesta
- **Execute** → Ejecuta un método HTTP especificado contra una URL, devolviendo un objeto mapeado en el cuerpo de la respuesta.
- **GetForEntity** → Envía una solicitud HTTP GET, devolviendo un ResponseEntity que contiene un objeto mapeado del cuerpo de respuesta

```
public Spittle fetchSpittle(long id) {
    RestTemplate rest = new RestTemplate();
    ResponseEntity<Spittle> response = rest.getForEntity(
"http://localhost:8080/spittr-api/spittles/{id}", Spittle.class, id);
    if(response.getStatusCode() == HttpStatus.NOT_MODIFIED) {
        throw new NotModifiedException();
    }
        return response.getBody();
    }
}
```

- **GetForObject** → Envía una solicitud HTTP GET, devolviendo un objeto asignado desde un cuerpo de respuesta

```
public Spittle[] fetchFacebookProfile(String id) {
    Map<String, String> urlVariables = new HashMap<String, String>();
    urlVariables.put("id", id);
    RestTemplate rest = new RestTemplate();
    return rest.getForObject("http://graph.facebook.com/{spitter}", Profile.class,
urlVariables);
}
```

- **HeadForHeaders** → Envía una solicitud HTTP HEAD, devolviendo los encabezados HTTP para los URL de recursos
- **OptionsForAllow** → Envía una solicitud HTTP OPTIONS, devolviendo el encabezado Allow URL especificada
- **PostForEntity** → Envia datos en el cuerpo de una URL, devolviendo una ResponseEntity que contiene un objeto en el cuerpo de respuesta

```
    RestTemplate rest = new RestTemplate();
    ResponseEntity<Spitter> response = rest.postForEntity(
"http://localhost:8080/spittr-api/spitters", spitter, Spitter.class);
    Spitter spitter = response.getBody();
    URI url = response.getHeaders().getLocation();
}
```

- **PostForLocation** → POSTA datos en una URL, devolviendo la URL del recurso recién creado

```
public String postSpitter(Spitter spitter) {
    RestTemplate rest = new RestTemplate();
    return rest.postForLocation("http://localhost:8080/spittr-api/spitters", spitter)
.toString();
}
```

- **PostForObject** → POSTA datos en una URL, devolviendo un objeto mapeado de la respuesta

cuerpo

```
public Spitter postSpitterForObject(Spitter spitter) {
    RestTemplate rest = new RestTemplate();
    return rest.postForObject("http://localhost:8080/spittr-api/spitters", spitter,
Spitter.class);
}
```

- **Put** → PUT pone los datos del recurso en la URL especificada

```
public void updateSpittle(Spittle spittle) throws SpitterException {
    RestTemplate rest = new RestTemplate();
    String url = "http://localhost:8080/spittr-api/spittles/" + spittle.getId();
    rest.put(URI.create(url), spittle);
}
```

33. Servicios

Un Servicio es un proceso que se ejecuta en segundo plano sin interacción con el usuario.

Un Servicio no tiene UI.

Los servicios por defecto corren en el hilo principal (Main Thread o Thread UI)

Algunos ejemplos de Servicios pueden ser

- La recuperación de datos con una conexión a internet cada cierto tiempo mientras el usuario usa el teléfono.
- El reproductor de música que reproduce una canción mientras el usuario navega por internet.

Los Servicios se pueden implementar de tres formas distintas

- Servicios Locales
- IntentService
- Servicios Remotos

Son elementos manejados, por lo que se han de declarar en el AndroidManifest

Declaración de Servicio en el AndroidManifest.

```
<service android:name="com.ejemplo.MiServicio">
</service>
```

Los Servicios se ejecutan en el mismo proceso de la aplicación a la que pertenecen, pero se pueden independizar incluyendo el atributo android:process en su declaración

Declaración de Servicio en otro proceso.

```
<service android:name="com.ejemplo.MiServicio" android:process=":nombre_proceso">
</service>
```

Los : en el nombre del proceso, indica que el proceso es privado para la aplicación, sino fuese así, indicaria que otra aplicación, podria emplear dicho proceso para lanzar sus propios servicios.

Los Servicios tienen menos prioridad que la Actividad en primer plano.

Se puede hacer que un Servicio tenga la misma prioridad que la Actividad en primer plano, pero para ello hay que mostrar una notificación.

Desde la versión 3.0, los Service pueden ser lanzados en el arranque siempre que se haya arrancado al menos una vez la aplicación que lo contiene en el terminal. Esto implica que no pueden existir Aplicaciones con unicamente Service, han de tener al menos una Activity.

33.1. Servicios Locales

Se basan en la clase abstracta Service, de la cual habrá que heredar.

Implementación de Servicio.

```
public class ActualizacionesService extends Service {
    private LocalBinder binder;

    public ActualizacionesService() {
    }

    @Override
    public void onCreate() {
        super.onCreate();
        binder = new LocalBinder();

    }

    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }
}
```

Esta clase nos obligará a implmentar el método

- onBind

Que deberá retornar un objeto de tipo IBinder, o mediador, que permitirá interaccionar al cliente con el servicio. Digamos que proporciona la interface de uso del Servicio.

Implementación de IBinder.

```
public class LocalBinder extends Binder {
    ActualizacionesService getService() {
        return ActualizacionesService.this;
    }
}
```

El método onBind, respondera a eventos de tipo

Interacción con el Service.

```
context.bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
```

Siendo **mConnection** un objeto **ServiceConnection** que hará de Callback, obteniendo el control cuando el cliente y el servicio esten enlazados y se tenga por tanto acceso al objeto mediador IBinder.

Implementacion de ServiceConnection.

```
ServiceConnection() {
    public void onServiceConnected(ComponentName className, IBinder service) {
        mBoundService = ((LocalService.LocalBinder)service).getService();
    }

    public void onServiceDisconnected(ComponentName className) {
        mBoundService = null;
    }
}
```

La interface ServiceConnection, obliga a implementar los métodos

- onServiceConnected ⇒ Callback cuando se ha producido la conexion con el servicio, que proporciona el objeto mediador IBinder
- onServiceDisconnected ⇒ Callback cuando se pierde la conexión con el servicio por que el proceso donde corre ha sido destruido, esto en los casos de servicios que corren en el mismo proceso que el cliente no va a suceder nunca, ya que en ese caso tambien el cliente habria sido destruido.

Tambien se ha de indicar un Flag, que podra ser de los tipos

- Context.BIND_AUTO_CREATE *

El servicio además podrá redefinir métodos como

- onStartCommand

- onUnbind
- onDestroy

El método onStartCommand responde a la invocación de startService desde el cliente, indicando con un Flag el modo en el que se arranca en servicio, pudiendo este ser de tres tipos

- START_STICKY ⇒ Servicios que han de ser explicitamente arrancados y parados. Si el sistema lo para, este se vuelve a iniciar pero con un intent nulo.
- START_NOT_STICKY ⇒ Si el sistema para el servicio, no se inicia automáticamente, excepto que haya intenciones pendientes.
- START_REDELIVER_INTENT ⇒ Si el sistema lo para, se vuelve a iniciar con la ultima intención recibida.

Los Servicios se pueden parar invocando stopService() o desde el propio servicio con stopSelf()

34. Hardware

Los dispositivos **Android** disponen de una serie de dispositivos **Hardware**, sobre los cuales Android proporciona un API para su manejo.

Los APIs proporcionados para manejar los dispositivos hardware se dividen en

- Camara de fotos.
- Sensores.
- Sensores de ubicación.
- Vibracion.
- Bluetooth.
- NFC.

34.1. Cámara de Fotos

La **Cámara de fotos** es uno de los dispositivos mas empleados en Android, teniendo un API propio para su manejo.

Dentro del API se proporciona una aplicación que permite el manejo de la cámara, no siendo necesario escribir el código para su manejo, únicamente requiriendo la invocación de dicha aplicación.

A la hora de invocar la aplicación de la cámara de fotos, habrá que tener en cuenta, el tipo de resultado deseado, pudiendo ser de dos tipos.

- Thumbnail.
- URI al fichero local con la fotografía a la resolución con la que ha sido tomada.

En ambos dos casos se ha de invocar la misma Actividad, aquella que responda a la siguiente

intención

Intencion para abrir la cámara de fotos.

```
Intent intent =  new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
```

Por regla general, la tarea que invoca esta Actividad querrá trabajar con el resultado de la operación, la fotografía, independientemente de si el resultado es un thumbnail o una imagen de gran tamaño, por lo que, de nuevo en ambos casos, habrá que invocarla con

Invocación de actividad de cámara de fotos.

```
startActivityForResult(intent, code);
```

La diferencia en el retorno se marcará, enviando o no, un parámetro en los Extras de la intención que invoca la cámara.

Paso de parámetro **EXTRA_OUPUT** *con la URI donde se guardará la foto tomada.*

```
intent.putExtra(MediaStore.EXTRA_OUTPUT, output);
```

Este parámetro marcará, que el resultado de la fotografía sea almacenado en la tarjeta SD. Siendo la URI de la ubicación donde se guardará la fotografía tomada.

Ejemplo de obtencion de un URI a partir de una ubicación en la tarjeta SD.

```
String name = Environment.getExternalStoragePublicDirectory(
                                    Environment.DIRECTORY_PICTURES).getPath() + "/" +
                                    nombreFichero.getText();
Uri output = Uri.fromFile(new File(name));
```

Para recoger el Thumbnail, se ha de obtener un parámetro en los Extras de tipo Parcelable, con clave **data**, que será el Bitmap del Thumbnail.

Obtención del Thumbnail.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    .
    .
    .
    Bitmap thumbnail = (Bitmap) data.getParcelableExtra("data");
    .
    .
    .
}
```

Para procesar la imagen a tamaño completo, se deberá hace uso de la misma URI que se envió a la

actividad para acceder al recurso.

Obtención del Thumbnail.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    .
    .
    .
    ImageView iv = (ImageView)findViewById(R.id.imgView);
    iv.setImageBitmap(BitmapFactory.decodeFile(name));
    .
    .
    .
}
```

Si se desea se dispone de un API para el manejo directo del Hardware de la camara, pudiendo manipular el Stream proporcionado por la misma y acceder a caracteristicas como detección de rostros, ... Si bien es un API que solo será necesario para manejos avanzados del Hardware, siendo en la mayor parte de las ocasiones suficiente con lo visto.

Para mas información soncultar http://developer.android.com/guide/topics/media/camera.html

34.2. Galería

Es una aplicación preinstalada en los terminales Android, que permite un acceso rapido a los contenidos de tipo Media, como Videos e Imagenes.

Para que esta aplicación proporcione visualización de dichos contenidos, se han de registrar primero, es decir no accede directamente a los directorios del sistema, buscando los recursos, sino que se han de dar de alta.

Para ello, se ha de recurrir a un Content Provider, aquel que responde a la URI **MediaStore.Images.Media.EXTERNAL_CONTENT_URI**, enviando como parametro con clave **MediaStore.Images.ImageColumns.DATA** el **Path** del recurso

Registro de un recurso en la galeria.

```
ContentValues values = new ContentValues();

values.put( MediaStore.Images.ImageColumns.DATA, name);

Uri result = getContentResolver().insert(
                        MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
                        values);
```

Es posible que las siguientes soluciones tambien sean validas

```
sendBroadcast(new Intent(
    Intent.ACTION_MEDIA_MOUNTED,
    Uri.parse("file://"+ Environment.getExternalStorageDirectory())));
```

```
sendBroadcast(new Intent(
    Intent.ACTION_MEDIA_SCANNER_SCAN_FILE,
    Uri.parse("file://" + path)));
```

```
MediaStore.Images.Media.insertImage(
    ContentResolver cr,
    String imagePath,
    String name,
    String description);
```

34.2.1. Acceso a la Galería en busca de un recurso

El API pone a disposición una actividad que permite la selección de recursos, esta actividad responde al **Intent.ACTION_PICK**, un Action generico, que determinará la actividad a invocar a partir de **Content Uri**, en este caso **MediaStore.Images.Media.EXTERNAL_CONTENT_URI**

Intención para abrir la actividad de galeria.

```
Intent intent = new Intent(Intent.ACTION_PICK, MediaStore.Images.Media
.EXTERNAL_CONTENT_URI);
```

Dado que se querrá trabajar con el resultado de la operación, esta actividad se invocará con

Invocación al contexto de la actividad de Galeria.

```
startActivityForResult(intent, code);
```

Dado que el registro de los recursos se realiza a traves de un **Content Provider**, el resultado de la operación, será un URI de un **Content Provider**

Obtención del resultado de la invocación de la actividad de Galeria.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    .
    .
    .
    Uri selectedImage = data.getData();
    InputStream is;
    try {
        is = getContentResolver().openInputStream(selectedImage);
        BufferedInputStream bis = new BufferedInputStream(is);
        Bitmap bitmap = BitmapFactory.decodeStream(bis);
        ImageView iv = (ImageView)findViewById(R.id.imgView);
        iv.setImageBitmap(bitmap);
    } catch (FileNotFoundException e) {}
    .
    .
    .
}
```

34.3. Sensores

Cada dispositivo tendrá unos sensores distintos, pudiendo ser estos de los siguientes tipos:

- Proximidad.
- Presión atmosférica.
- Temperatura interna.
- Gravedad.
- Acelerómetro lineal.
- Vector de rotación.
- Temperatura ambiental.
- Humedad relativa.

Todos estos sensores se procesan con el mismo API.

El eje central del API es el SensorManager, accesible desde el contexto

Obtención del SensorManager.

```
SensorManager sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE
);
```

A partir de este objeto se puede obtener el listado de sensores disponibles en general, o de una tipologia concreta

Listado de sensores.

```
List<Sensor> deviceSensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
```

Obteniendo referencias a objetos Sensor, que representarán el sensor.

Existen unas constates definidas en la clase Sensor, para acceder a los distintos sensores

- Sensor.TYPE_GYROSCOPE
- Sensor.TYPE_MAGNETIC_FIELD
- Sensor.TYPE_ORIENTATION (deprecated en API 8 en favor de SensorManager.getOrientation())
- Sensor.TYPE_ACCELEROMETER
- Sensor.TYPE_LIGHT
- Sensor.TYPE_PRESSURE
- Sensor.TYPE_TEMPERATURE
- Sensor.TYPE_PROXIMITY
- Sensor.TYPE_GRAVITY
- Sensor.TYPE_LINEAR_ACCELERATION
- Sensor.TYPE_ROTATION_VECTOR
- Sensor.TYPE_RELATIVE_HUMIDITY
- Sensor.TYPE_AMBIENT_TEMPERATURE
- Sensor.TYPE_ALL

Otra forma de obtener la referencia al sensor es como sensor por defecto

Sensor por defecto.

```
Sensor defaultSensorAcelerometro = sensorManager.getDefaultSensor(Sensor
.TYPE_ACCELEROMETER);
```

Una vez obtenida la referencia al Sensor, las lecturas se harán de forma asincrona, es decir, no se puede controlar el momento exacto de la lectura, sino que dependerá del sistema, pudiendo unicamente registrar un listener a la lectura con una determinada frecuencia.

El Listener debera implementar la interface **SensorEventListener**

Registro de Listener de Sensor.

```
sensorManager.registerListener(this, defaultSensorAcelerometro, SensorManager
.SENSOR_DELAY_NORMAL);
```

Las frecuencias posibles son

- SensorManager.SENSOR_DELAY_NORMAL. Velocidad adecuada para los cambios de orientación

de pantalla

- SensorManager.SENSOR_DELAY_FASTEST. Velocidad mas rápida posible.
- SensorManager.SENSOR_DELAY_GAME. Velocidad adecuada para los juegos
- SensorManager.SENSOR_DELAY_UI. Velocidad adecuada para la interface de usuario.

El Listener implementará dos métodos

- onSensorChanged
- onAccuracyChanged

El primero de ello, recibira en forma de float[] como parametro del evento, las lecturas del sensor.

Implementación del Listener.

```
@Override
public void onSensorChanged(SensorEvent event) {
    float[] values = event.values;
    if(event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        Toast.makeText(this, "Acelerometro: " + values[0] + ":" + values[1] + ":" +
values[2], Toast.LENGTH_LONG).show();
    } else if(event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD){
        Toast.makeText(this, "Campo Magnetico: " + values[0] + ":" + values[1] + ":" +
values[2], Toast.LENGTH_LONG).show();
    }
}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {

}
```

Dependiendo del Sensor, estas lecturas corresponderan a unos valores u otros, para mas información

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

Es importante remarcar que los sensores consumen bateria cuando estan enviado muestras, por lo que es conveniente, en caso de emplear desde un Actividad, desregistrar los Listener cuando la actividad no este en primer plano, y volver a registrarlos cuando la Actividad vuelva al primer plano.

Para ello, serán utiles los métodos del ciclo de vida de onPause y onResume

Para poder probar los sensores, se puede emplear una consola del terminal, a la que se puede acceder con

```
telnet localhost <puerto de la consola, por defecto 5554>
```

Realizando peticiones del estilo

```
sensor status
sensor get <nombre sensor>
sensor set <nombre sensor> <valor1:valor2:valor3>
```

34.4. Sensores de ubicación

Existen varias formas de obtener la localización en un terminal Android.

- GPS
- Antenas de telefonía móvil.
- Puntos de acceso WIFI.

Cada mecanismo, tiene una precisión, velocidad y consumo de recursos distinto.

Estos mecanismos tienen su representación en Android como **LocationProviders**.

El eje central del API es el LocationManager, accesible desde el contexto

Obtención del LocationManager.

```
LocationManager locationManager = (LocationManager) getSystemService(Context
.LOCATION_SERVICE);
```

A partir de este objeto se puede obtener el listado de los nombres de los proveedores de localización disponibles

Listado de sensores de localización.

```
List<String> listaProviders = locManager.getAllProviders();
```

O de un proveedor de localización concreto aplicando unos determinados criterios de seleccion, representados por la clase Criteria

Obtención del proveedor con Criteria.

```
Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_FINE);
criteria.setAltitudeRequired(true);
String bestProvider = locationManager.getBestProvider(criteria, false);
```

Ademas de no estar habilitado el sensor, se puede pedir al usuario que lo active.

Invocación de activación de sensores de localización.

```
if (!locationManager.isProviderEnabled(bestProvider)){
    Intent gpsOptionsIntent = new Intent(android.provider.Settings
.ACTION_LOCATION_SOURCE_SETTINGS);
    startActivity(gpsOptionsIntent);
}
```

Una vez obtenida la referencia a proveedor de Localización, se necesita registrar un Listener para procesar las lecturas, para ello

Registro de Listener para Localización.

```
locationManager.requestLocationUpdates(bestProvider, 1 * 60 * 1000, 100, this);
```

Los parámetros que recibe el método requestLocationUpdates son

- String ⇒ Nombre del Proveedor de Localización.
- long ⇒ Mínimo intervalo de tiempo entre muestras en milisegundos.
- float ⇒ Mínima distancia entre muestras en metros.
- LocationListener ⇒ Objeto que será el que reciba las muestras.

El Listener implementará la interface LocationListener, debiendo implementar los métodos

- onLocationChanged
- onStatusChanged
- onProviderEnabled
- onProviderDisabled

El el método onLocationChanged, se recibe un objeto de tipo Location, que permite acceder a las caracteristicas de localización.

Ejemplo de implementación de onLocationChange.

```
public void onLocationChanged(Location location) {
    Toast.makeText(this,
            "Altitud: " + location.getAltitude() +
                    "; Longitud: " + location.getLatitude() +
                    ":Latitud: " + location.getLongitude(),
            Toast.LENGTH_SHORT).show();
}
```

34.4.1. Permisos

Será necesario definir permisos en el AndroidManifest, para poder acceder al API de Localización

Ejemplo de implementación de onLocationChange.

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

34.5. Vibración

Para controlar la vibración del terminal, el API proporciona un Servicio del Sistema Vibrator.

Obtención de Vibrator.

```
Vibrator vibrator = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
```

A partir del objeto Vibrator se pueden enviar patrones de vibración al servicio con el método vibrate

Definición y ejecución de patrón de vibración.

```
vibrator.vibrate(new long[]{1000l, 2000l, 1000l, 3000l}, 4);
```

El método **vibrate**, acepta como parámetros

- long[] ⇒ que define el patron en milisegundos encendido/apagado
- int ⇒ que define las veces que se repite el patrón

34.5.1. Permisos

Será necesario definir permisos para acceder al API de vibración

Ejemplo de implementación de onLocationChange.

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

35. Timmer

En construccion

36. AlarmManager

En construccion

37. Seguridad

En construccion

38. Imagenes 2D

En construccion

39. Imagenes 3D

En construccion

40. Gestos

En construccion

41. Pruebas

En construccion

42. Widget

En construccion

43. Proguard

En construccion

44. Publicacion en Market

El registro como desarrollador Android en Google Play tiene un coste de 25$ en un sólo pago, no existe cuota anual, para registrarse pulsar aquí

Se necesita una cuenta de GMAIL.

Se ha de proceder al registro en

```
https://play.google.com/apps/publish/signup/
```

Con los pasos indicados en el enlace, se consigue ser desarrollador de aplicaciones gratuitas.

Para publicar aplicaciones de pago, hay que completar un nuevo paso, darse de alta en Google Merchant, pero este ya se ha de realizar sobre la consola de desarrollador.