

GUIA ANDROID JUNIOR

1. Introducción

- 1.1. Bienvenida
- 1.2. Historia
- 1.3. ¿Qué es Android?
- 1.4. ¿Qué necesito para programar en Android?

2. Arquitectura del Sistema

- 2.1. Arquitectura general
 - Licencia
- 2.2. Capas de Arquitectura
- 2.3. Componentes de una aplicación
 - Activities (actividades).
 - Intents (intenciones).
 - Content providers (proveedores de contenido).
 - Clases contract
 - Services (servicios).
 - AIDL
 - Broadcast receivers
- 2.4. Dalvik - Sobre librerías de núcleo
- 2.5. Procesos en Android

3. Entorno de Desarrollo

- 3.1. SDK (Instalación)
 - ADT Plugin
- 3.2. Instalar Plataformas (Comando android)
 - Instalar Plataformas
- 3.3. AVD Manager

4. Fundamentos de Aplicaciones Android

- 4.1. Desarrollo de Aplicaciones Android
- 4.2. Creación y estructura de un proyecto
- 4.3. Trabajando con el archivo AndroidManifest.xml
- 4.4. Creación y manejo de Activities
- 4.5. Uso de Intents explícitos
- 4.6. Uso de Intents implícitos
- 4.7. Creación y uso de recursos
- 4.8. Seguridad y permisos
- 4.9. Depuración de una App

5. Interfaz de Usuario y Controles

- 5.1. Unidades y Layout
- 5.2. Uso de Layout Managers
- 5.3. Controles de Texto
- 5.4. Controles de tipo Botón
 - Sobre eventos...

- [5.5. Controles de tipo Lista](#)
- [5.6. Layouts de Lista Personalizados](#)
- [5.7. Componentes más comunes](#)
- [6. Estilos y Elementos Gráficos](#)
 - [6.1. Creación y uso de Estilos](#)
 - [6.2. Creación y uso de Temas](#)
 - [6.3. Creación de Iconos](#)
 - [6.4. Creación de imágenes 9-Patch](#)
- [7. Soporte para Múltiples Pantallas](#)
 - [7.1. Comprensión de tamaño de pantalla y densidad](#)
 - [7.2. Incluir Layouts alternos](#)
- [8. Animación y Gráficos](#)
 - [8.1. Configuración de animación cuadro por cuadro](#)
 - [8.2. Mostrar animación sincronizada](#)
 - [8.3. Trabajar con Gráficos 2D](#)
- [9. Menús y Diálogos](#)
 - [9.1. Opciones de Menú](#)
 - [9.2. Menús Contextuales](#)
 - [9.3. Diálogos de Alerta](#)
 - [9.4. Diálogos de Progreso](#)
 - [9.5. Diálogos Personalizados](#)
- [10. Notificaciones y Toast](#)
 - [10.1. Desplegar notificaciones en barra de estatus](#)
 - [10.2. Desplegar notificaciones Toast](#)
- [11. Trabajar con Multimedia](#)
 - [11.1. Reproducción de audio](#)
 - [11.2. Reproducción de video](#)
 - [11.3. Acceso a la cámara](#)
- [12. Preferencias y Almacenamiento de Datos](#)
 - [12.1. Uso de Shared Preferences](#)
 - [12.2. Crear Activity de Preferences](#)
 - [12.3. Sistema de Archivos Interno y Externo](#)
 - [12.4. Uso de Base de Datos en SQLite](#)
 - [12.5. Acceso a la Red](#)
 - [12.6. Uso de Content Providers](#)
- [13. Geo localización y Mapas](#)
 - [13.1. Incorporación de Google Maps Android API v2](#)
 - [13.2. Uso de GPS para ubicar la posición actual](#)
- [14. Creación de Widgets](#)
 - [14.1. Widget simple home-screen](#)
 - [14.2. Activity de Configuración de Widget](#)
- [15. Publicación de Aplicación](#)
 - [15.1. Preparación para publicación](#)
 - [15.2. Firma y Construcción Lección](#)

[15.3. Preparación de los recursos gráficos](#)

[15.4. Publicación a Google Play \(ANTES Android Market !!!\)](#)

[16. Conclusiones](#)

[16.1. Uso de Ejemplos del SDK](#)

[16.2. Recursos de utilidad](#)

1. Introducción

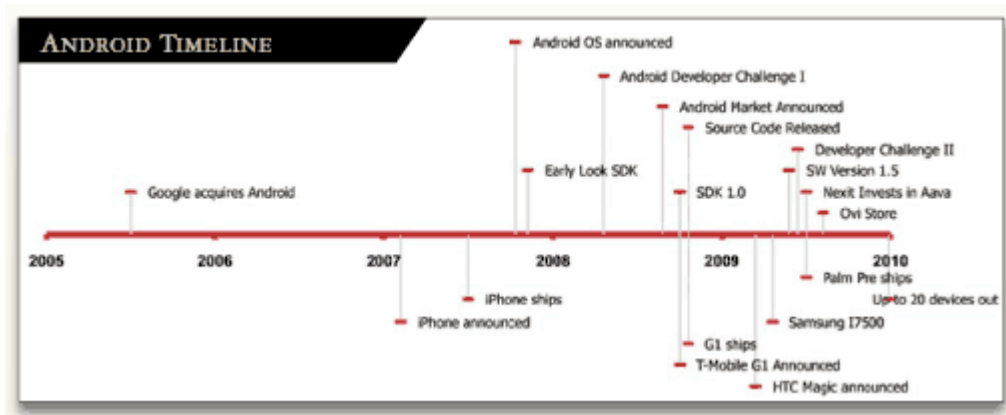
1.1. Bienvenida

1.2. Historia

Android era un sistema operativo para móviles prácticamente desconocido propiedad de una empresa llamada **Android Inc.** hasta que en 2005 Google lo compró. Actualmente [Andy Rubin](#), el creador de Android, trabaja como vicepresidente de ingeniería de Google y tiene bajo su mando este proyecto.

En noviembre de 2007 sólo circulaban rumores de que el gigante de Internet tenía en planes lanzar un proyecto para móviles, y precisamente por esas fechas se lanzó la [Open Handset Alliance](#), que agrupaba a muchos fabricantes de teléfonos móviles, chipsets y Google y se proporcionó la primera versión de Android, junto con el SDK para que los programadores empezaran a crear sus aplicaciones para este sistema.

Aunque los inicios fueran un poco lentos, debido a que se lanzó el sistema operativo antes que el primer móvil, rápidamente se ha colocado como una plataforma que ya se ha ganado muchos adeptos y que ha demostrado una velocidad de madurez importante debido a los diferentes fabricantes que han adaptado interesantes piezas de hardware para acompañar las diferentes funciones de Android.



1.3. ¿Qué es Android?

Android es una pila de **software** pensada inicialmente para teléfonos móviles (**smartphones**) que incluye un **sistema operativo**, **middleware** y una capa aplicaciones para que el teléfono pueda realizar funciones más allá de los dispositivos que se usaban antaño.

Dejando un poco de lado los tecnicismos, Android es otra de las opciones de interfaces y características que podemos encontrar en teléfonos móviles; así como podemos identificar aspectos particulares de un Nokia cuyo sistema operativo es Symbian, o un iPhone con iOS o incluso los Blackberry, también existen cosas muy específicas en los teléfonos que funcionan con Android.

Ahora volvámonos más técnicos. Android es un sistema operativo basado en el kernel de Linux, por esa razón tiene inmersas las características de ser libre, gratuito y multiplataforma. Cualquier desarrollador puede crear aplicaciones para Android sin la necesidad de pagar membresías anuales para obtener el kit de desarrollo (SDK).

Android utiliza una variación del lenguaje de programación **Java** que es diferente a la Java ME. Si has desarrollado en Java, seguramente conocerás lo que es trabajar con una **máquina virtual** que sirve para interpretar todo ese código que genera nuestro programa (bytecode) y pueda ejecutarse. Pues bien, Android, tiene una adaptación de esta máquina virtual y se llama **Dalvik**. Distinta de la virtual machine por default de Java, **Dalvik** es una excelente versión que optimiza muchas cosas en la plataforma y te sorprenderá saber que la rapidez de Android y de algunas aplicaciones se debe precisamente a esto.

Cabe mencionar que el sistema operativo proporciona todas las interfaces necesarias para desarrollar aplicaciones que accedan a las funciones del teléfono (como el GPS, las llamadas, la agenda, etc.) de una forma muy sencilla utilizando Java.



1.4. ¿Qué necesito para programar en Android?



Herramientas de Software

- **Kit de desarrollo.** La versión más nueva del SDK de Android que puedes encontrar en [este link](#).
- **Sistema operativo.** Una computadora que corra con Windows, Linux o Mac OS. En Windows y en Linux no importa la arquitectura que tengas (32 bits o 64 bits).
- **IDE.** Elegir entre la instalación de Eclipse con el plugin ADT.
- **Versión de Java.** La versión 6 de Java corre bien con Android. La puedes descargar desde [este link](#).

Equipo Android

- Uno de los puntos importantes que se recomienda, es tener es un teléfono que corra con Android. Esto es porque de esta manera tendrás una auténtica forma de probar cómo es que corren tus aplicaciones en un ambiente real.

Conocimientos técnicos

- **Programación orientada a objetos.** Si has programado en algún lenguaje que se base en este paradigma (PHP, Python, Java, .NET, C o C++, etc.) entonces vas por buen camino. Si este no es tu caso, puedes empezar a [revisar unos cuantos conceptos básicos](#).
- **Lenguaje Java.** Esto viene a complementar el punto anterior y si bien no lo pongo como algo obligatorio, si has programado en Java, la migración a Android será mucho más sencilla ya que las aplicaciones de Android se programan en este lenguaje de programación.

- **Manejo de algún IDE de desarrollo.** Cualquiera que sea el lenguaje en el que hayas programado seguramente has hecho uso de algún **IDE** (entorno de desarrollo integrado). Para el caso de Android, tendrás ventaja si has utilizado **Eclipse**.

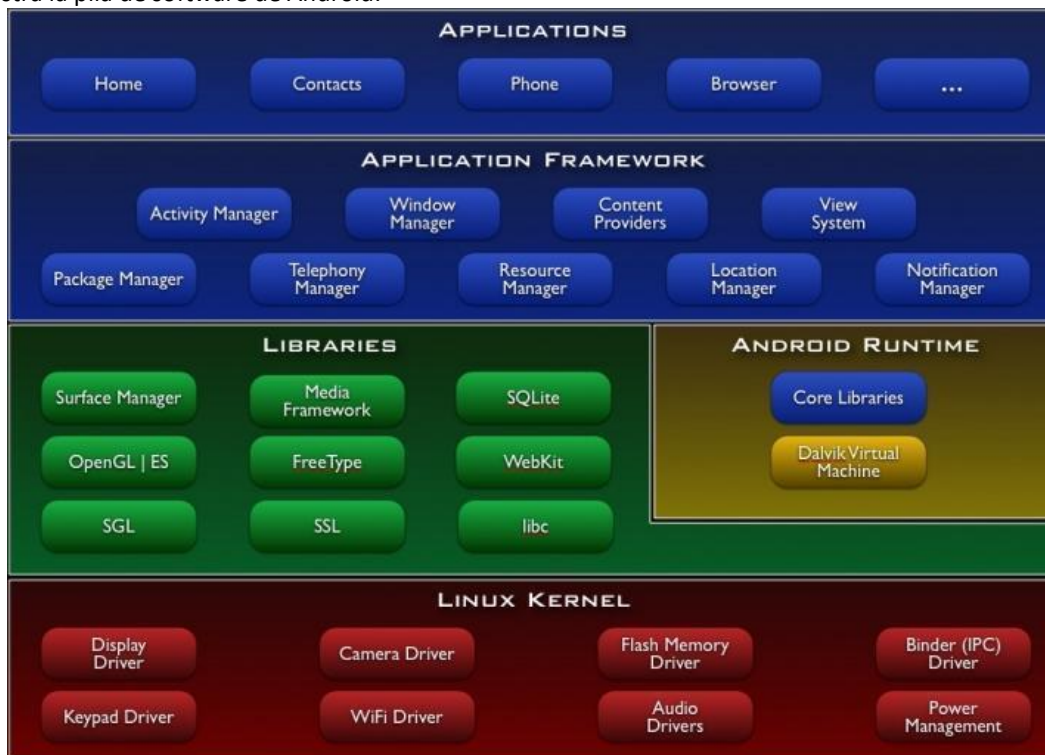
2. Arquitectura del Sistema

2.1. Arquitectura general

Android es un sistema operativo basado en linux para dispositivos móviles, como tabletas o teléfonos. El sistema Android es una pila de software para dispositivos móviles que incluye un sistema operativo, un middleware(software intermedio entre las aplicaciones y el sistema operativo) y unas aplicaciones de básicas. Android provee una plataforma de desarrollo abierto y ofrece a los desarrolladores la capacidad de crear aplicaciones ricas e innovadoras y la libertad de tomar las ventajas que ofrece el hardware del dispositivo. Los desarrolladores tienen total acceso las APIs del framework, ya que la arquitectura de aplicación en Android está diseñada para simplificar el reuso de los componentes, así como APIs para el control de los dispositivos para conectividad, sensores, etc. Detrás de todas las aplicaciones esta un conjunto de servicios y sistemas, llamado el marco de trabajo de la aplicación (Application Framework).

Los componentes mostrados en la sección Application Framework, pueden ser usados en el desarrollo de aplicaciones para Android, normalmente escritos en Lenguaje Java.

Aquí se muestra la pila de software de Android:



Kernel de Linux. El núcleo del sistema operativo Android está basado en el kernel de Linux versión 2.6, similar al que puede incluir cualquier distribución de Linux, como Ubuntu, solo que adaptado a las características del hardware en el que se ejecutará Android, es decir, para dispositivos móviles.

El núcleo actúa como una capa de abstracción entre el hardware y el resto de las capas de la arquitectura. El desarrollador no accede directamente a esta capa, sino que debe utilizar las librerías disponibles en capas superiores. De esta forma también nos evitamos el hecho de quebrarnos la cabeza para conocer las características precisas de cada teléfono. Si necesitamos hacer uso de la cámara, el sistema operativo se encarga de utilizar la que incluya el teléfono, sea cual sea. Para cada elemento de hardware del teléfono existe un controlador (o *driver*) dentro del kernel que permite utilizarlo desde el software.

El kernel también se encarga de gestionar los diferentes recursos del teléfono (energía, memoria, etc.) y del sistema operativo en sí: procesos, elementos de comunicación (*networking*), etc.

Librerías. La siguiente capa que se sitúa justo sobre el kernel la componen las bibliotecas nativas de Android, también llamadas librerías. Están escritas en C o C++ y compiladas para la arquitectura hardware específica del teléfono. Estas normalmente están hechas por el fabricante, quien también se encarga de instalarlas en el dispositivo antes de ponerlo a la venta. El objetivo de las librerías es proporcionar funcionalidad a las aplicaciones para tareas que se repiten con frecuencia, evitando tener que codificarlas cada vez y garantizando que se llevan a cabo de la forma “más eficiente”.

Entre las librerías incluidas habitualmente encontramos OpenGL (motor gráfico), Bibliotecas multimedia (formatos de audio, imagen y video), Webkit (navegador), SSL (cifrado de comunicaciones), FreeType (fuentes de texto), SQLite (base de datos), entre otras.

Entorno de ejecución. Como podemos apreciar en el diagrama, el entorno de ejecución de Android no se considera una capa en sí mismo, dado que también está formado por librerías. Aquí encontramos las librerías con la funcionalidades habituales de Java así como otras específicas de Android.

El componente principal del entorno de ejecución de Android es la máquina virtual **Dalvik**. Las aplicaciones se codifican en Java y son compiladas en un formato específico para que esta máquina virtual las ejecute. La ventaja de esto es que las aplicaciones se compilan una única vez y de esta forma estarán listas para distribuirse con la total garantía de que podrán ejecutarse en cualquier dispositivo Android que disponga de la versión mínima del sistema operativo que requiera la aplicación.

Cabe aclarar que Dalvik es una variación de la máquina virtual de Java, por lo que **no es compatible con el bytecode Java**. Java se usa únicamente como lenguaje de programación, y los ejecutables que se generan con el SDK de Android tienen la extensión `.dex` que es específico para Dalvik, y por ello no podemos correr aplicaciones Java en Android ni viceversa.

Framework de aplicaciones. La siguiente capa está formada por todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones. La mayoría de los componentes de esta capa son librerías Java que acceden a los recursos de las capas anteriores a través de la máquina virtual Dalvik. Siguiendo el diagrama encontramos:

1. Activity Manager. Se encarga de administrar la pila de actividades de nuestra aplicación así como su ciclo de vida.
2. Windows Manager. Se encarga de organizar lo que se mostrará en pantalla. Básicamente crea las superficies en la pantalla que posteriormente pasarán a ser ocupadas por las actividades.
3. Content Provider. Esta librería es muy interesante porque crea una capa que encapsula los datos que se compartirán entre aplicaciones para tener control sobre cómo se accede a la información.
4. Views. En Android, las vistas los elementos que nos ayudarán a construir las interfaces de usuario: botones, cuadros de texto, listas y hasta elementos más avanzados como un navegador web o un visor de Google Maps.
5. Notification Manager. Engloba los servicios para notificar al usuario cuando algo requiera su atención mostrando alertas en la barra de estado. Un dato importante es que esta biblioteca también permite jugar con sonidos, activar el vibrador o utilizar los LEDs del teléfono en caso de tenerlos.
6. Package Manager. Esta biblioteca permite obtener información sobre los paquetes instalados en el dispositivo Android, además de gestionar la instalación de nuevos paquetes. Con paquete nos referimos a la forma en que se distribuyen las aplicaciones Android, estos contienen el archivo .apk, que a su vez incluyen los archivos .dex con todos los recursos y archivos adicionales que necesite la aplicación, para facilitar su descarga e instalación.
7. Telephony Manager. Con esta librería podremos realizar llamadas o enviar y recibir SMS/MMS, aunque no permite reemplazar o eliminar la actividad que se muestra cuando una llamada está en curso.
8. Resource Manager. Con esta librería podremos gestionar todos los elementos que forman parte de la aplicación y que están fuera del código, es decir, cadenas de texto traducidas a diferentes idiomas, imágenes, sonidos o layouts. En un post relacionado a la estructura de un proyecto Android veremos esto más a fondo.
9. Location Manager. Permite determinar la posición geográfica del dispositivo Android mediante GPS o redes disponibles y trabajar con mapas.

10. Sensor Manager. Nos permite manipular los elementos de hardware del teléfono como el acelerómetro, giroscopio, sensor de luminosidad, sensor de campo magnético, brújula, sensor de presión, sensor de proximidad, sensor de temperatura, etc.
11. Cámara: Con esta librería podemos hacer uso de la(s) cámara(s) del dispositivo para tomar fotografías o para grabar vídeo.
12. Multimedia. Permiten reproducir y visualizar audio, vídeo e imágenes en el dispositivo.

Aplicaciones. En la última capa se incluyen todas las aplicaciones del dispositivo, tanto las que tienen interfaz de usuario como las que no, las nativas (programadas en C o C++) y las administradas (programadas en Java), las que vienen preinstaladas en el dispositivo y aquellas que el usuario ha instalado.

En esta capa encontramos también la aplicación principal del sistema: Inicio (Home) o lanzador (launcher), porque es la que permite ejecutar otras aplicaciones mediante una lista y mostrando diferentes escritorios donde se pueden colocar accesos directos a aplicaciones o incluso widgets, que son también aplicaciones de esta capa.

Como podemos ver, Android nos proporciona un entorno sumamente poderoso para que podamos programar aplicaciones que hagan cualquier cosa. Nada dentro de Android es inaccesible y podemos jugar siempre con las aplicaciones de nuestro teléfono para optimizar cualquier tarea.

Licencia

Algunas partes del sistema Android son desarrolladas en open source, así que el código fuente siempre está disponible; sin embargo, otras partes son desarrolladas primero en un árbol privado, y luego es liberado cuando la siguiente versión de la plataforma esta lista.

Mas informacion:

<http://source.android.com/faqs.html>

El Proyecto de código abierto de Android (Android Open Source Project AOSP) se refiere a las personas, procesos (instrumentos y procedimientos) y el código fuente que componen Android, al ser conformado por una pila de software, hace uso de varias licencias, por ejemplo al usar el kernel de linux, usa la licencia GPL. La mayor parte del software Android ha adoptado la licencia Apache 2.0 (Apache License 2.0) esta licencia concede algunos derechos y obligaciones.

Algunos derechos de la licencia:

- **Reproducción:** Se otorga al licenciataro el derecho de ejecutar y reproducir libremente el software por cualquier medio y en cualquier forma.
- **Modificación:** El licenciataro tiene el derecho de modificar libremente el programa, incluido el derecho a traducirlo, así como la posibilidad de crear cualquier tipo de desarrollo e integración derivada del mismo.
- **Distribución:** El licenciataro tiene el derecho de distribuir libremente el software licenciado, a través de cualquier modalidad, soporte o formato.
- **Comunicación Pública:** El licenciataro podrá poner el software a disposición de una pluralidad de personas sin previa distribución de ejemplares físicos a cada una de ellas, incluida la ejecución pública del programa de ordenador.
- **Sublicenciamiento (en código fuente o código objeto):** El licenciataro tiene el derecho de sublicenciar libremente el software en código fuente o código objeto (binario).

Algunas obligaciones:

- Incluir siempre una copia de la licencia.
- Indicar cualquier modificación.
- Mantener los avisos de titularidad (copyright notice), marca o patentes.
- Inclusión de fichero notice.txt con indicaciones de autoría, modificaciones y otros contenidos de carácter legal.

Mas informacion:

<http://source.android.com/faqs.html>

<http://source.android.com/licenses.html>

<http://www.apache.org/licenses/LICENSE-2.0.html>

http://wiki.cenatic.es/wikiesp/index.php/CAP%C3%8DTULO_TRES:_An%C3%A1lisis_de_licencias_de_fuentes_y_contenidos_abiertos

2.2. Capas de Arquitectura

Vease tema 2.1. Arquitectura General

2.3. Componentes de una aplicación

Activities (actividades).

-->Contenido en Moodle

Es el bloque encargado de construir la interfaz de usuario. Puedes pensar en una actividad como el análogo de una ventana en una aplicación de escritorio. En las actividades recae la responsabilidad de presentar los elementos visuales y reaccionar a las acciones del usuario.

Si bien podemos pensar en actividades que no posean una interfaz de usuario, regularmente el código en estos casos se empaqueta en forma de *content providers* (proveedores de contenido) y *services* (servicios) que detallaremos en un momento.

Toda actividad se inicia como respuesta a un *Intent*.

-->Nuevo Contenido

Una Activity es el componente de la aplicación que provee una ventana (pudiendo tomar diferentes tamaños en la pantalla) con la que los usuarios pueden interactuar. Cada Activity es independiente de las demás, no importando si se incluyen muchas en una App; cuando se crea una Activity, la Activity anterior se detiene mediante notificaciones llamadas “callbacks” que el SO Android usa para gestionar el ciclo de vida de cada Activity. Las dos llamadas “callbacks” más importantes son:

- `onCreate()`: Se ejecuta cuando se crea la aplicación.
- `onPause()`: Se ejecuta cuando otra Activity acaba de entrar a primer plano, o acaba de ser instanciada.

Las demas “callbacks” son:

- `onRestart()`: Se ejecuta cuando el usuario vuelve a navegar a la Activity, luego de haber sido ocultada por otra Activity (`onStop()`): su ejecución se hace justo antes de `onStart()`.
- `onStart()`: Se ejecuta justo antes de que la Activity se haga visible (en primer plano) para el usuario.
- `onResume()`: Se ejecuta justo antes de que la Activity comience a interactuar con el usuario.
- `onStop()`: Se ejecuta cuando la Activity ya no es visible para el usuario.
- `onDestroy()`: Se ejecuta justo antes de destruir la Activity.

La secuencia de estas llamadas “callbacks” se muestra en la siguiente ilustración:



Intents (intenciones).

-->Contenido en Moodle

Son los mensajes del sistema que se encuentran corriendo en el interior del dispositivo. Se encargan de notificar a las aplicaciones de varios eventos: cambios de estado en el hardware (ej. cuando se inserta una SD al teléfono), notificaciones de datos entrantes (ej. cuando llega un SMS) y eventos en las aplicaciones (ej. cuando una actividad es lanzada desde el menú principal).

Así como podemos crear actividades que respondan a las intenciones, podemos crear también intenciones que lancen actividades o usarlas para detonar eventos ante algunas situaciones específicas (ej. que el teléfono nos notifique por medio de un mensaje cuando nuestra localización esté a 10 mts del punto de destino). Es importante mencionar que el sistema es el que elige entre las actividades disponibles en el teléfono y dará respuesta a la intención con la actividad más adecuada.

-->Nuevo Contenido

Los Intents son "mensajes" (llamados así y definiéndose como estructuras que describen de forma abstracta una operación a realizar o de algo que ha ocurrido) mediante los cuales se **activan 3 de los componentes básicos de una aplicación: activities, services, y Broadcast receivers** (actividades, servicios y receptores de emisiones). Hay mecanismos diferentes para la entrega de los Intents a cada tipo de componente, aquí un breve resumen:

- Un objeto Intent se pasa como parámetro a el método **Context.startActivity()** o **Activity.startActivityForResult()**, para lanzar una Activity.
- Un objeto Intent se pasa como parametro a **Context.starService()** para iniciar un servicio o entregar nuevas instrucciones a un servicio ya en marcha. También se puede pasar como parámetro al método **Context.bindService()** para establecer una conexión entre el componente y el servicio destino.
- Un objeto Intent se pasa como parámetro a cualquiera de los métodos de emisión (broadcast) como **Context.sendBroadcast()**, **Context.sendOrderedBroadcast()** o **Context.sendStickyBroadcast()**, los cuales entregan el objeto Intent a todos los receptores de emisiones (Broadcast receiver) interesados.

Estos mecanismos son mutuamente excluyentes, lo que significa que sólo se usará un mecanismo a la vez: los Intents de emisiones (Broadcast Intents) se entregan solo a receptores de emisiones (BroadcastReceivers), nunca a Activities o Servicios, los Intents para iniciar Activities solo se entregan a una Activity, nunca a un servicio o receptor de emisiones (BroadcastReceivers), etc.

En el caso específico del tercer mecanismo, los BroadcastReceivers pueden restringir quién le puede enviar Intent de emisiones, esto se logra aplicando permisos en el archivo AndroidManifest.xml en el elemento <receiver>, el permiso se verifica después de que terminó la ejecución del método Context.sendBroadcast() ya que el sistema trata de entregar la emisión a el receptor, si no tiene permiso para eso simplemente no se entrega. El permiso también se pueden suministrar con el método Context.registerReceiver() para controlar quién puede emitir a un receptor registrado.

Un Intent contiene información de interés para el componente que recibe ese Intent (como una acción a realizar y los datos que necesite esa acción), y para el sistema Android (como la categoría del componente o las instrucciones de como lanzar una Activity). Principalmente puede contener:

- **Nombre del componente:** El nombre del componente que debe manejar el Intent, en formato completo ejemplo: "com.ejemplo.NuevaActivity".
- **Acción (action):** la acción general a realizar, la clase Intent define unas constantes de estas acciones ejemplo ACTION_VIEW (constante: android.intent.action.VIEW) para mostrar información al usuario, ACTION_EDIT (constante: android.intent.action.EDIT) para mostrar información editable al usuario, ACTION_MAIN (constante: android.intent.action.MAIN) para iniciar la Activity inicial, ACTION_CALL(constante: android.intent.action.CALL) para iniciar una llamada de teléfono etc...
- **Datos (data):** Los datos necesarios, ejemplo: el id de una persona en la base de datos o alguna direccion WEB, expresada como un URI.
- **Categoría:** Contiene información adicional sobre el tipo de componente que debe manejar el Intent, también hay algunas constantes como CATEGORY_HOME para mostrar la activity en la pantalla de inicio o cuando el botón Home se presiona, CATEGORY_LAUNCHER para mostrar la actividad en el lanzador de aplicaciones.
- **Extras:** Información adicional en forma llave-valor, que son entregados al componente que maneja el Intent.

- **Banderas (Flags):** Banderas para indicarle al sistema Android, como lanzar una actividad o cómo será tratada después de lanzada (por ejemplo si debe pertenecer a la lista de las Activities recientes).

Los Intents se dividen en dos grupos:

- **Intents Explícitos:** Designan el componente objetivo por nombre, se usan, por ejemplo: para iniciar una activity, para iniciar un servicio.
- **Intents Implícitos:** No llaman por nombre al componente objetivo, el cual está en blanco, se usan para activar componentes en otras aplicaciones, el sistema Android debe buscar el mejor componente o componentes para manejar ese Intent, ya se trate de una activity o servicio para realizar la acción requerida o un conjunto de receptores de emisiones (Broadcast receivers) para responder a la emisión. Esto lo hace mediante la comparación de los contenidos del objeto Intent a los filtros de Intent (Intent filters), que son estructuras asociadas con componentes que potencialmente pueden recibir esos Intents.

De toda la información que contiene un Intent, mencionada anteriormente, solo 3 aspectos son consultados cuando se compara el objeto Intent con los Intent filter:

- Action
- Data
- Category

Ejemplos:

- **Intent Explícito:** Para iniciar otra Activity desde una que está en ejecución se llama al método `startActivity()`, el cual recibe como parámetro un Intent que describe la actividad que se quiere iniciar. Ejemplo:

```
[...]  
Intent intent = new Intent(this, OtraActivity.class);  
startActivity(intent);
```

- **Intent Implícito:** Usando un Intent para abrir una página web la implementación sería:

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.google.com"));  
startActivity(intent);
```

Más información:

<http://developer.android.com/intl/es/guide/components/intents-filters.html>

<http://developer.android.com/intl/es/guide/topics/security/permissions.html>

Content providers (proveedores de contenido).

→ Contenido en moodle

Este elemento ofrece un conjunto de datos almacenados en el dispositivo para que se puedan acceder y compartir por varias aplicaciones. Nosotros podemos guardar datos en archivos del sistema, en una base de datos en SQLite, en la web o en cualquier otro lugar de almacenamiento persistente a la que la aplicación pueda tener acceso. A través del proveedor de contenido, otras aplicaciones pueden consultar o incluso modificar los datos (solamente si el proveedor de contenidos de esa aplicación lo permite).

El modelo de desarrollo en Android nos invita a los desarrolladores a pensar siempre en hacer los datos de nuestras aplicaciones disponibles para otras aplicaciones. De manera que cuando creamos un proveedor de contenido, podemos tener un control completo de nuestros datos y definir la forma en que éstos serán accedidos.

Un ejemplo sencillo de esto es el proveedor de contenido que tiene Android para gestionar la información de contactos (agenda) del teléfono. Cualquier aplicación con los permisos adecuados puede realizar una consulta a través de un proveedor de contenido como *ContactsContract.Data* para leer y escribir información sobre una persona en particular.

-->Nuevo Contenido

Los proveedores de contenido (ContentProvider) administran el acceso a un banco central de datos, los encapsulan y proveen mecanismos para la definición de la seguridad en los datos. Los proveedores de contenido están destinados principalmente a ser utilizados por otras aplicaciones. El objeto **ContentResolver** en el cliente que solicita información, es el que se comunica con el objeto proveedor, el **ContentProvider**. Este objeto proveedor recibe solicitudes de los clientes, realiza la acción solicitada y devuelve los resultados, por ello se dice que no tiene un ciclo de vida más allá del ser referenciado por el ContentResolver. El proveedor de contenido presenta los datos a aplicaciones externas como una o más tablas de una base de datos relacional.

Cada proveedor de contenido se identifica con una URI que contiene una cadena de texto única. Esta URI de contenido tiene 3 partes:

- El esquema URI (URI scheme) que identifica que es un URI de contenido, es el máximo atributo esencial para especificar una URI, ejemplos: http://, https://, content://, file://, market://search? (El esquema URI "android://" no es soportado por el sistema Android).
- El nombre simbólico del proveedor (la autoridad o authority).
- El nombre que apunta a una tabla (la ruta o path) (separada con "/").

Clases contract

Una clase contrato (contract) define constantes que ayudan a las aplicaciones a lidiar con la URI de contenido, nombres de columnas, opciones de Intent y otras características de un proveedor de contenido. El programador del "proveedor" tiene que definir estas clases contrato y luego hacerlas disponibles para otros programadores.

El paquete `android.provider` contiene clases para acceder a los proveedores de contenido estándar de Android que almacenan datos comunes como informaciones de contactos, información del calendario y archivos de medios como audio o video, estas clases proporcionan métodos simplificados para añadir o recuperar datos de esos proveedores de contenido. Algunas de esas clases son:

- **Browser.BookmarkColumns:** Definiciones de columnas para los elementos favoritos (bookmarks) y el historial.
- **CalendarContract:** El contrato entre el proveedor (provider) de calendario y las aplicaciones.
- **CallLog:** Este proveedor contiene información sobre llamadas realizadas y recibidas.
- **ContactsContract:** El contrato entre el proveedor (provider) de contactos y las aplicaciones.
- **MediaStore:** Este proveedor contiene metadatos para todos los medios ya sea en dispositivos de almacenamiento internos y externos.
- **Settings:** Este proveedor contiene las preferencias del dispositivo a nivel del sistema.
- **UserDictionary:** El proveedor de las palabras definidas por el usuario para métodos de entrada para usar introducción de texto predictivo.
- **VoicemailContract:** El contrato entre el proveedor de correo de voz y las aplicaciones.

Implementación:

En Archivo `AndroidManifest.xml`:

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

En código:

```
Cursor personas = getContentResolver().query(ContactsContract.Contacts.CONTENT_URI,
                                             null, null, null, null);
while(personas.moveToNext()){
    int indiceNombre = personas.getColumnIndex(PhoneLookup.DISPLAY_NAME);
    String nombre = personas.getString(indiceNombre);
    Log.d("CONTACTOS", nombre);
}
```

Services (servicios).

-->Contenido en Moodle

Las actividades, las intenciones y los proveedores de contenido explicados arriba son de corta duración y pueden ser detenidos en cualquier momento. Por el contrario, los servicios están diseñados para seguir corriendo, y si es necesario, de manera independiente de cualquier actividad. El ejemplo más simple para aterrizar este concepto es el del reproductor de música, que es un servicio que puede mantenerse corriendo mientras mandamos un SMS o realizamos alguna otra función en nuestro teléfono.

-->Nuevo Contenido

Un servicio es un componente de la aplicación que no provee de una interfaz de usuario y que puede realizar, en segundo plano y cuando otro componente de la aplicación lo inicia, operaciones de larga duración, incluso si el usuario cambia a otra aplicación. Adicionalmente, un componente puede vincularse a un servicio (Bound Service) para interactuar con él, llegando a realizar comunicación entre procesos (IPC). Por ejemplo un servicio puede manejar las operaciones de red, reproducir

música, realizar operaciones con de entrada y salida con archivos o interactuar con un proveedor de contenido, todo desde segundo plano, ejecutándose, de forma predeterminada, en el hilo principal de la aplicación.

Un servicio puede tomar 2 formas no excluyentes:

- **Iniciado:** Cuando un componente de la aplicación (como una Activity) inicia un servicio llamando al método `startService()`, lo que resulta en una llamada al método `onStartCommand()` del servicio. Una vez iniciado, el servicio puede correr indefinidamente en segundo plano, incluso si el componente que lo inicio es destruido. Usualmente un servicio iniciado realiza una sola operación y no regresa un resultado, por ejemplo, para descargar o cargar un archivo en la red. Una vez realizada la operación, el servicio mismo debería detenerse, llamando al método `stopSelf()`. Puede darse el caso de que haya muchas llamadas al método `startService()`, lo que ejecutaría muchas veces el método `onStartCommand()` del servicio, pero no importa cuantas veces se inicie un servicio, será detenido llamando al método `stopSelf()` dentro del servicio o `stopService()` desde fuera de él.
- **Vinculado (bound):** Cuando un competente de la aplicación se vincula con un servicio llamando al método `bindService()`, lo que resulta en una llamada al método `onBind()` del servicio. El servicio de vinculación (Bound Service) es el servidor en la interfaz cliente-servidor, permite a los componentes (los cliente, como las Activities) vincularse a el servicio, solicitar peticiones, recibir respuestas e incluso hacerlo a través de IPC. El cliente se comunica con el servicio a través de la interfaz IBinder que es devuelta por el método `onBind()` del servicio. Un servicio de vinculación se ejecuta solo mientras algún componente de la aplicación está vinculado a él. Muchos componentes pueden vincularse al servicio, sin embargo el método `onBind()`, solo se ejecutará la primera vez, el resto solamente devolverá el objeto IBinder. Cuand todos se desvinculen (llamando al método `unbindService()`) , el servicio es destruido (a menos que también se haya iniciado por el método `startService()`).

La parte más importante de un servicio de vinculación (**Bound Service**) es definir la interfaz IBinder que el método `onBind()` regresara.

La interfaz IBinder (La parte central de un mecanismo ligero de llamada a procedimientos remotos diseñado para un alto rendimiento cuando se realizan llamadas en procesos y entre ellos) define la interfaz de programación que los clientes pueden usar para interactuar con el servicio, para definir esta interfaz hay tres maneras de hacerlo:

- **Extender o heredar de la clase Binder:** Se usa cuando el servicio que se esta creando es solamente un proceso en segundo plano para la misma aplicación. No se debe usar este método cuando el servicio que se esta creando se podrá usar por otras aplicaciones o en otros procesos. Para implementarlo, se debe regresar la instancia en el método `onBind()`, el cliente recibirá el objeto Binder y lo podrá usar para acceder a los métodos públicos disponibles.
- **Usando un Messenger:** Es la forma más sencilla de realizar la comunicación entre procesos (IPC), ya que Messenger pone todas las peticiones en cola en un solo hilo y se atienden una a la vez, lo que hace que el servicio sea seguro en hilos (thread-safe). Messenger permite que la interfaz trabaje en varios procesos, definiendo un controlador (handler) que responde a los diferentes tipos de objetos Messenger, el controlador puede compartir el IBinder con el cliente permitiendo que el cliente mande comandos a el servicio usando objetos Message, además de que el cliente puede definir un Messenger propio para que el servicio pueda mandarle mensajes de vuelta.
- **Usando AIDL:** El AIDL (Lenguaje de definición de interfaz de Android) realiza todo el trabajo de descomponer objetos en primitivas que el sistema operativo pueda entender y ordenar a través de procesos para realizar la comunicación entre procesos(IPC). Contrario a Messenger, usar AIDL permite que el servicio gestione varias peticiones a la vez, implicando con ello que el servicio debe ser capaz de hacer trabajo multi-hilo y ser seguro en hilos (thread-safe). Para usar AIDL, se debe crear un archivo .aidl que define la interfaz de programación.

Todos los servicios se deben declarar en el archivo `AndroidManifest.xml` de la aplicación. Se debe agregar el elemento `<service>` como elemento hijo del elemento `<application>`. El elemento `<service>` deberá especificar, en el atributo `android:name=""`, el nombre completo (incluyendo el paquete) de la clase del servicio. Ejemplo:

```
<manifest ... >
    ...
    <application ... >
        <service android:name="com.ejemplo.Servicios.ServicioEjemplo" />
        ...
    </application>
</manifest>
```

AIDL

AIDL es similar a otros Lenguajes de definición de interfaces (IDL). Se debe definir la interfaz AIDL en un archivo .aidl usando la sintaxis del lenguaje Java, y guardarlo en el directorio src/ en la aplicación que hospeda al servicio y en cualquier otra aplicación que se enlace con ese servicio. Cuando se construye la aplicación que contiene ese archivo, se genera una interfaz IBinder basada en el archivo anterior y se guarda en el directorio gen/. El servicio debe implementar esa interfaz y, entonces, las aplicaciones cliente pueden enlazarse al servicio y llamar a los métodos de la interfaz IBinder para realizar la IPC.

Para crear un servicio usando AIDL, se necesita:

1. **Crear el archivo .aidl:** permite declarar una interfaz con 1 o más métodos que pueden tomar parámetros y devolver valores que pueden ser de cualquier tipo, incluso otras interfaces AIDL.
2. **Implementar la interfaz:** después de que se genera la interfaz en el lenguaje Java a partir del archivo anterior. Esta interfaz tiene una clase interna abstracta llamada Stub que extiende de Binder e implementa los métodos de la interfaz AIDL. Se debe extender de la clase Stub e implementar esos métodos.
3. **Exponer la interfaz a los clientes:** implementar el servicio y sobrescribir el método onBind() para devolver la implementación de la clase Stub.

Más información:

<http://developer.android.com/intl/es/reference/android/app/Service.html>

<http://developer.android.com/intl/es/guide/components/services.html>

<http://developer.android.com/intl/es/guide/components/bound-services.html>

Broadcast receivers

-->Contenido en Moodle

Dependiendo de la fuente consultada o el autor, podrán encontrar también un elemento llamado **Broadcast receivers** (receptores de radiodifusión). Este componente responde a las notificaciones de difusión de todo el sistema como por ejemplo que la pantalla se ha apagado, que la batería del teléfono está por terminarse o que una fotografía ha sido tomada. Las aplicaciones también pueden lanzar este tipo de notificaciones, un ejemplo de ello es el aviso de que alguna información ha terminado de descargarse en el dispositivo y se encuentran disponibles para su utilización.

Aunque los broadcaster receivers no muestran una interfaz de usuario, pueden crear notificaciones en la barra de estado del teléfono para avisarle al usuario cuando un evento de este tipo se genera.

Podemos pensar en estos elementos como el medio por el cual otros componentes pueden ser iniciados en respuesta a algún evento. Cabe mencionar que cada una de las emisiones de los broadcaster receivers se representa como una intención.

→ Nuevo Contenido

Los BroadcastReceivers permiten a las aplicaciones recibir Intents que se emiten por el sistema o por otras aplicaciones, incluso cuando los otros componentes de la aplicación no se están ejecutando. Un objeto BroadcastReceiver sólo es válido mientras dure la llamada al método `onReceive(Context, Intent)`, durante el cual se considera que es un proceso en primer plano, que es el nivel más alto de importancia que el sistema Android destruye únicamente como último recurso; cuando termina la ejecución de ese método el sistema considera que el objeto ya no está activo y puede terminarlo. No es posible mostrar un diálogo o vincularse a un servicio desde dentro de un BroadcastReceiver, si es necesario hacerlo se debe usar el API NotificationManager para mostrar un diálogo y `Context.startService()` para enviar un comando al servicio.

La implementación se hace creando una clase que extienda o herede de la clase BroadcastReceiver y sobrescribiendo el método `onReceive(Context, Intent)`, mencionado anteriormente. Aquí un ejemplo:

```
public class MiBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        //Hacer algo
    }
}
```

-->Contenido en Moolde

Un aspecto único y útil del diseño del sistema operativo Android es que cualquier aplicación puede hacer uso de otro componente de otra aplicación. Supongamos que la aplicación requiere que el usuario tome una fotografía con la cámara del teléfono; es probable que exista otra aplicación que haga exactamente eso, por lo que resultará más fácil reutilizar esa función que programar una específica en nuestra aplicación. Para ello no es necesario incluir o vincular el código de la aplicación de la cámara, simplemente hará falta iniciar la actividad que se encargue de capturar la foto. Una vez hecho, la foto estará disponible para usarla en la aplicación “principal”. Para el usuario parecerá que la cámara de su teléfono es una parte de la aplicación.

Debido a que el sistema Android ejecuta cada una de las aplicaciones en un proceso separado con permisos de archivos que pueden restringir el acceso a otras aplicaciones, las aplicaciones no pueden activar de forma directa un componente de otra aplicación. El único que puede hacer esto es el sistema operativo en sí. Por lo tanto, para activar un elemento de otra aplicación, debemos pasarle un mensaje al sistema dónde se especifique qué elemento es el que necesitamos correr. De esta manera el sistema activará el componente y podremos manipularlo según sea nuestra necesidad.

Todos los elementos expuestos en este tópico son objetos y su comportamiento está definido en su correspondiente clase base convirtiendo a cada elemento en una subclase del elemento original. Esto

lo hacemos por medio de la [herencia](#), una característica básica de Java y de los lenguajes orientados a objetos, y que nos evitará tener que volver a implementar aspectos comunes en todos los objetos del mismo tipo, y poder al mismo tiempo, personalizar su comportamiento tanto como lo necesite nuestra aplicación por medio de la sobre escritura de los métodos de la clase padre.

2.4. Dalvik - Sobre librerías de núcleo

--> Contenido en Moodle
La máquina virtual Dalvik



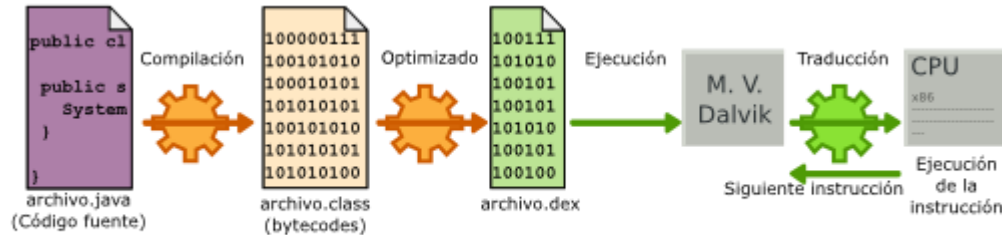
El Sistema Operativo Android hace uso de una máquina virtual llamada **Dalvik** que se encuentra en la capa de ejecución y que ha sido diseñada para optimizar la memoria y los recursos de hardware en el entorno de los teléfonos móviles.

Un dato curioso para “*telonear*” este post: el nombre “**Dalvik**” proviene de un pueblo en Islandia, lugar de dónde proviene el ingeniero que diseñó la máquina virtual para Android.

Los engranes que mueven a Dalvik

Dalvik es una máquina virtual intérprete que ejecuta archivos en el formato Dalvik Executable (*.dex), un formato optimizado para el almacenamiento eficiente y ejecución mapeable en memoria. Su objetivo fundamental es el mismo que cualquier máquina virtual, permite que el código sea compilado a un bytecode independiente de la máquina en la que se va a ejecutar, y la máquina virtual interpreta este bytecode a la hora de ejecutar el programa. Una de las razones por las cuáles no se optó por utilizar la

máquina virtual de Java es la necesidad de optimizar al máximo los recursos y enfocar el funcionamiento de los programas hacia un entorno dónde los recursos de memoria, procesador y almacenamiento son escasos.



Dalvik está basada en registros y puede ejecutar clases compiladas (vágase la redundancia) por un compilador Java y que posteriormente han sido convertidas al formato nativo usando la herramienta “dx”. El hecho de que corra sobre un kernel Linux le permite delegar las tareas relacionadas con la gestión de hilos y memoria a bajo nivel.

Otra característica importante de Dalvik es que ha sido optimizada para que múltiples instancias de ella puedan funcionar al mismo tiempo con un impacto muy bajo en el rendimiento de la memoria del dispositivo. El objetivo de esto es proteger a las aplicaciones, de forma que el cierre o fallo inesperado de alguna de ellas no afecte de ninguna forma a las demás.

Dalvik != Java Virtual Machine

La máquina virtual de Java, que podemos encontrar en casi todas las PC's actuales, se basa en el uso de las pilas. De modo contrario, Dalvik utiliza los registros, ya que los teléfonos móviles están optimizados para la ejecución basada en los mismos.

Aunque utilizamos el lenguaje Java para programar las aplicaciones Android, el bytecode de Java no es ejecutable en un sistema Android. De igual forma, las librerías Java que utiliza Android son ligeramente distintas a las utilizadas en Java Standard Edition (Java SE) o en Java Mobile Edition (Java ME), guardando también características en común.

Dalvik y la optimización en aplicaciones Android

El uso de Dalvik permite reducir bastante el tamaño del programa buscando información duplicada en las diversas clases y reutilizándola.

Lo que llamamos en Java como “recolectar basura”, que libera el espacio en memoria de objetos que ya no utilizamos en nuestros programas, ha sido perfeccionada en Android con el fin de mantener siempre libre la máxima memoria posible.

De igual forma, el hecho de que Android haga un uso extenso del lenguaje XML para definir las interfaces gráficas y otros elementos, implica que estos archivos deben ser linkeados a la hora de compilar y para que su conversión a bytecode pueda mejorar el rendimiento de nuestras aplicaciones.

A lo largo de nuestro aprendizaje en Android veremos también algunos tips y mejores prácticas para que no dependamos únicamente de Dalvik a la hora de hablar de rendimiento en aplicaciones.

---> Nuevo Contenido

Android incluye un conjunto de librerías en el núcleo que proveen la mayoría de las funciones disponibles en las librerías del núcleo del lenguaje de programación Java.

Cada aplicación de Android, se ejecuta en su propio proceso, con su propia instancia de la máquina virtual Dalvik, esta ha sido escrita para que un dispositivo pueda ejecutar múltiples máquinas virtuales eficientemente, está basada en registros y corre clases compiladas por el compilador de Java y convertidas en el formato ejecutable .dex por la herramienta dx, que, explícitamente, se usa mediante línea de comandos (no tiene interfaz gráfica) o es llamada, automáticamente, por Eclipse cuando esta contruyendo la aplicación.

La librería de clases del núcleo de Dalvik provee una base de desarrollo familiar a la programación con Java Standard Edition (Java SE), pero orientado a las necesidades de un pequeño dispositivo móvil.

2.5. Procesos en Android

-->Contenido en Moodle



En la mayoría de los casos, una aplicación Android se ejecuta en su propio proceso de Linux. Este proceso es creado para la aplicación cuando la arrancamos y seguirá corriendo hasta que no sea necesario y el sistema reclame recursos para otras aplicaciones y se los dé a éstas.

Así es, el tiempo de vida de un proceso en Android es manejada por el sistema operativo, basándose en las necesidades del usuario, los recursos disponibles, etc. Si tenemos una aplicación que está consumiendo muchos recursos y arrancamos otra nueva aplicación, el sistema operativo probablemente le diga a la aplicación que se queda en segundo plano que libere todo lo que pueda, y si es necesario la cerrará.

En Android los recursos son normalmente muy limitados y por eso el sistema operativo tiene más control sobre las aplicaciones que en programas de escritorio.

Para determinar qué procesos eliminar ante un escenario donde el dispositivo tenga poca batería u otros en los que sea de relevante importancia administrar los recursos, Android les asigna una prioridad a cada uno de ellos basándose en la siguiente jerarquía:

1. **Foreground Process:** Es la aplicación que contiene la actividad ejecutada en primer plano en la pantalla del usuario y con la cuál está interactuando ahora mismo (Se ha llamado a su método `onResume()`). Por lo regular habrá muy pocos procesos de este tipo corriendo a la vez en el sistema y son aquellos que se eliminarán como última opción si la memoria es tan baja que ni matando al resto de procesos tenemos los recursos necesarios.
2. **Visible Process:** Es un proceso que aloja una Activity que no se está ejecutando en primer plano (es decir, su método `onPause()` ha sido llamado). Un ejemplo puede ser la aplicación de correo en la cuál demos click en algún enlace de interés que nos lance el navegador, este pasaría a ser el Foreground Process dejando a la aplicación de correo en el concepto de Visible Process. Este tipo de procesos se cerrarán únicamente cuando el sistema no tenga los recursos necesarios para mantener corriendo todos los procesos que estén en primer plano.
3. **Service Process:** Son aquellos que corren cuando un Service ha sido invocado. Estos procesos hacen cosas en segundo plano que normalmente son importantes para el usuario (conexión con servidores, actualización del GPS, reproductor de música, etc.), el sistema nunca va a liquidar un servicio a menos que sea necesario para mantener vivos todos los Visible y Foreground.
4. **Background Process:** Es un proceso que contiene una Activity que actualmente no es visible por el usuario y que ya no tienen demasiada importancia. Por ejemplo, los programas que arrancó el usuario hace tiempo y no los ha vuelto a usar, pasan a estar en background. Por eso es importante que cuando nuestra aplicación pase a Background, el sistema libere, en la medida de lo posible, todos los recursos que pueda para que su rendimiento sea óptimo.

5. **Empty Process:** Es un proceso que no aloja ningún tipo de componente. Su razón de ser es el de tener una caché disponible para la próxima aplicación que lance el usuario. Es común que el sistema elimine este tipo de procesos con frecuencia para así poder obtener memoria disponible.

→ Nuevo Contenido

Android puede decidir apagar un proceso en el mismo punto, cuando la memoria es baja y se requiere memoria por otros procesos que están más inmediatamente sirviendo al usuario. Los componentes de la aplicación que están corriendo en el proceso que se apaga, son consecuentemente destruidos y cuando se queda pendiente trabajo para ellos, se inicia un nuevo proceso. Android utiliza un sistema de "jerarquías de importancia" en el cual coloca cada proceso que se inicia en base a los componentes que se ejecutan en ese proceso y el estado de los mismos, Los de importancia más baja se eliminan primero.

Android sigue unas reglas para decidir el orden en que los procesos son terminados:

Este sistema de "jerarquías de importancia" tiene 5 niveles, organizados en orden de más importante a menos importante, son:

1. **Proceso de primer plano (Foreground).** Un proceso que es requerido para lo que el usuario está haciendo actualmente.
2. **Proceso visible.** Un proceso que no tiene componentes en el primer plano, pero que puede afectar lo que el usuario ve en la pantalla.
3. **Proceso de servicio.** Un proceso que está ejecutando un servicio que se ha iniciado con el método `startService()` y no esté incluida en alguna de las categorías de arriba.
4. **Proceso en segundo plano (Background).** Un proceso que contiene una Activity que no es actualmente visible para el usuario (cuando se llama al método `onStop()`).
5. **Proceso vacío (Empty).** Un proceso que no contiene ningún componente activo de la aplicación, y que solo están creados para fines de almacenamiento de caché o mejorar el tiempo de arranque de un componente la siguiente vez que se ejecutará.

Una Activity tiene 3 modos de presentación en la pantalla durante su ciclo de vida:

- Todo el tiempo de vida de la Activity ocurre entre la llamada a los métodos `onCreate()` y `onDestroy()`, incluso si la activity tiene un hilo corriendo en segundo plano (**background**) éste se debe crear y destruir en estos métodos. La Activity está en estado **Background** cuando es ocultada por otra diferente Activity.
- El tiempo de vida **visible** de una activity ocurre entre la llamada a los métodos `onStart()` y `onStop()`, durante este tiempo el usuario puede ver la activity en la pantalla e interactuar con ella, también se encuentra en este estado cuando pierde el foco de entrada del usuario, por la aparición de un diálogo o un Toast.
- El tiempo de vida en primer plano (**foreground**) ocurre entre los métodos `onResume()` y `onPause()`, durante este tiempo la activity está en el frente de todas las demás activities en la pantalla y tiene el foco de entrada del usuario.

3. Entorno de Desarrollo

3.1. SDK (Instalación)

-->CONTENIDO EN MOODLE



Para instalar el entorno de desarrollo se necesita:

- **IDE Eclipse** que te puedes descargar desde [este link](#). En esta página verás una lista con varias versiones de esta herramienta. Se recomienda la versión más reciente de Eclipse Classic.
- **SDK de Android** que se encuentra disponible en [la página oficial de Android developers](#). Este kit de desarrollo incluye un conjunto de herramientas tales como un debugger, librerías, un emulador, documentación, código de ejemplo y tutoriales.
- **Plugin ADT**. Es conveniente revisar la documentación que está en la [página oficial de Android Developers](#). Este plugin está diseñado especialmente para hacer de Eclipse un ambiente integrado y poderoso para desarrollar aplicaciones en Android.

Recomendación: Descargar el paquete "todo en uno" o **ADT Bundle** desde [este link](#).

→ NUEVO CONTENIDO

El Kit de desarrollo de software de Android (Android SDK) provee librerías y herramientas de desarrollo de la Interfa de programación de aplicaciones (API) necesarias para construir, probar y depurar aplicaciones para Android. Para iniciar rápidamente a desarrollar aplicaciones se recomienda descargar el paquete de herramientas de desarrollo de Android (ADT Bundle) que incluye:

- Eclipse con el plugin ADT
- Herramientas del SDK de Android
- Herramientas de la plataforma Android
- La última plataforma de Android
- La última imagen del sistema de Android para el emulador.

El kit de desarrollo de software (SDK) de Android, desde su primera versión, ha recibido importantes actualizaciones que se liberan en lanzamientos oficiales de la plataforma. Las cuales añaden nuevas características y mejoran las ya existentes. Estos lanzamientos se identifican a través de un número incremental (con cada lanzamiento) llamado código de la versión del SDK. Haciendo posible, al desarrollar una aplicación, especificar la versión mínima y la versión objetivo en la cual la aplicación funcionará con los elementos `android:minSdkVersion="X"` y `android:targetSdkVersion="X"` del elemento `<uses-sdk>` del archivo `AndroidManifest.xml`. Algunas versiones se pueden ver en la siguiente tabla.

Version	Codename	API	Distribution
1.6	Donut	4	0.1%
2.1	Eclair	7	1.7%
2.2	Froyo	8	3.7%
2.3 - 2.3.2	Gingerbread	9	0.1%
2.3.3 - 2.3.7		10	38.4%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	27.5%
4.1.x	Jelly Bean	16	26.1%
4.2.x		17	2.3%

Para el desarrollo de aplicaciones se recomienda que el nivel mínimo de la API sea la 1.6, aun si se está desarrollando para la plataforma 1.5, ya que para la versión 1.5 se liberó un AVD Manager, pero sin las funciones de administración del SDK.

Más información (ver todos los códigos de versión):

http://developer.android.com/intl/es/reference/android/os/Build.VERSION_CODES.html

ADT Plugin

El plugin de Herramientas de desarrollador de Android (Android Developer Tools ADT Plugin) integra al IDE Eclipse un conjunto de herramientas. Ofrece un acceso a muchas características que ayudan al desarrollo de aplicaciones de Android entre las que destacan:

- Integración al IDE de Eclipse para la creación, construcción, empaquetamiento, instalación y depuración de proyectos de Android.
- Integración con las herramientas de la SDK (SDK Tools).
- Editores XML para archivos XML específicos de Android (recursos, layouts, etc) como el Layout Editor, el cual permite modificar un archivo `layout.xml` en una interfaz de usuario, mediante arrastrar y soltar elementos.
- Integración de la documentación para la API de Android.

3.2. Instalar Plataformas (Comando android)

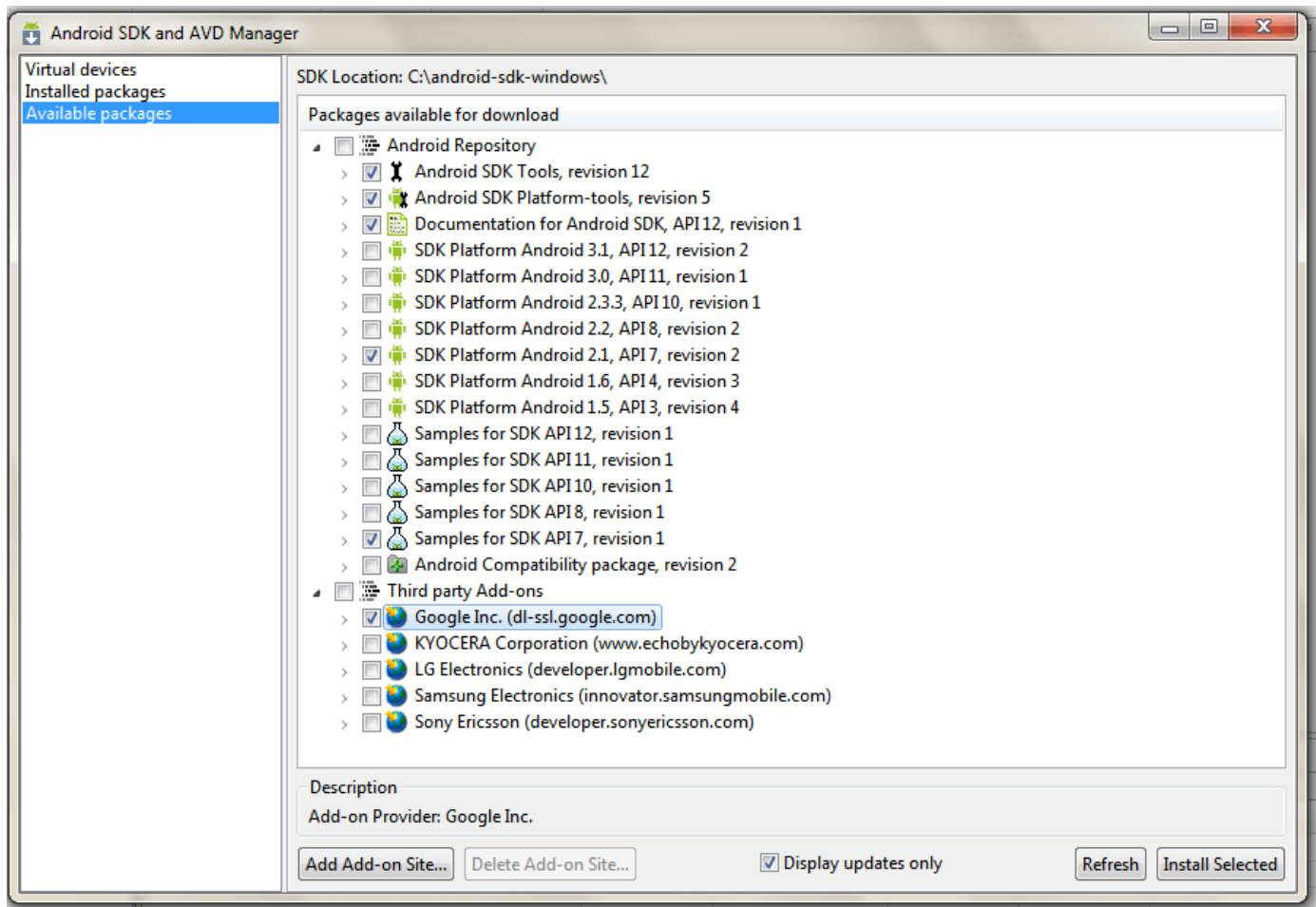
→ NUEVO CONTENIDO

Instalar Plataformas

El SDK de Android utiliza una estructura modular que separa las partes principales del mismo (las versiones del SDK, complementos, herramientas, ejemplos y documentación) en un conjunto de componentes que podemos instalar por separado. El SDK que descargamos incluye únicamente la última versión de las herramientas del SDK.

Para desarrollar una aplicación en Android necesitamos al menos descargar una plataforma Android con sus herramientas asociadas. Se puede instalar tantos componentes como plataformas se requieran.

- Seleccionar la opción *Window > Android SDK and AVD Manager*.
- Elegir la opción *Available packages* del panel situado a la izquierda.
- En el panel principal se pueden ver todas las opciones disponibles para descargar. En este link se puede consultar los componentes mínimos que se recomiendan descargar desde [la página oficial de Android developers](#).



Si se desea desarrollar para varias versiones se debe elegir las plataformas correspondientes. De igual forma, si se desea que las aplicaciones corran específicamente en alguna marca de teléfono se debe considerar el descargar los complementos de terceros en caso de que se encuentren en esta lista.

Dependiendo del número de plataformas y componentes que se decida descargar, este proceso puede tardar varias minutos.

→ NUEVO CONTENIDO

La herramienta Android permite crear, en un ambiente de línea de comandos, los 3 tipos de proyectos:

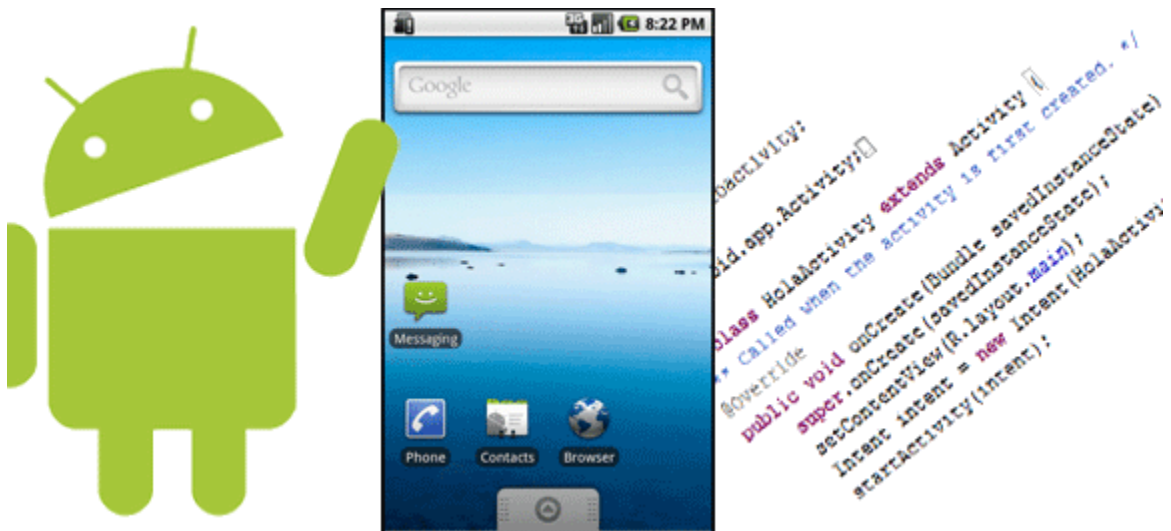
- Un proyecto de Android que contiene todos los archivos y recursos necesarios para construir un proyecto a un archivo instalable .apk.
- Un proyecto designado como librería, que permite ser compartida con otros proyectos que dependen de esa librería.
- Un proyecto de prueba que extiende de JUnit.

Esta herramienta es útil, además, para:

- **Actualizar un proyecto:** Útil cuando se está actualizando un proyecto desde una versión anterior del SDK o si se quiere crear un nuevo proyecto a partir de código ya existente.
- **Administrar los dispositivos virtuales de android (AVD):** Permite crear, borrar y listar los AVD.
- **Actualizar el Kit de Desarrollo de Software (SDK) de Android:** Actualizar el SDK con nuevas plataformas, nuevas API's, nuevos complementos (Add-ons) y documentación.

3.3. AVD Manager

-->CONTENIDO MOODLE



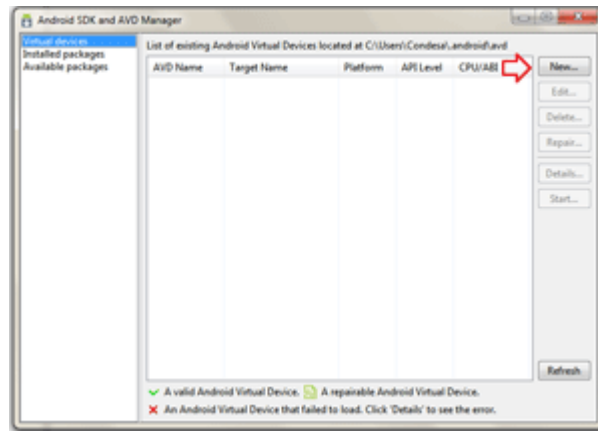
Antes de iniciar a desarrollar la primera aplicación en Android es importante tener disponible una **AVD** (**Android Virtual Device**) en Eclipse.

Las **AVD** son unas herramientas imprescindibles para los desarrolladores y testers, ya que permiten emular en una computadora un entorno móvil a los que apuntarán las aplicaciones en Android. Cuando recién se ha instalado el SDK (que incluye el *AVD Manager*) no se cuenta con ningún dispositivo virtual.

1. Abrir el *AVD Manager* desde la opción *Window > Android SDK and AVD Manager* o desde el siguiente icono que se encuentra en la barra de herramientas de Eclipse:



2. Elegir la opción *Virtual devices* del panel situado a la izquierda y dar clic sobre el botón *New...*



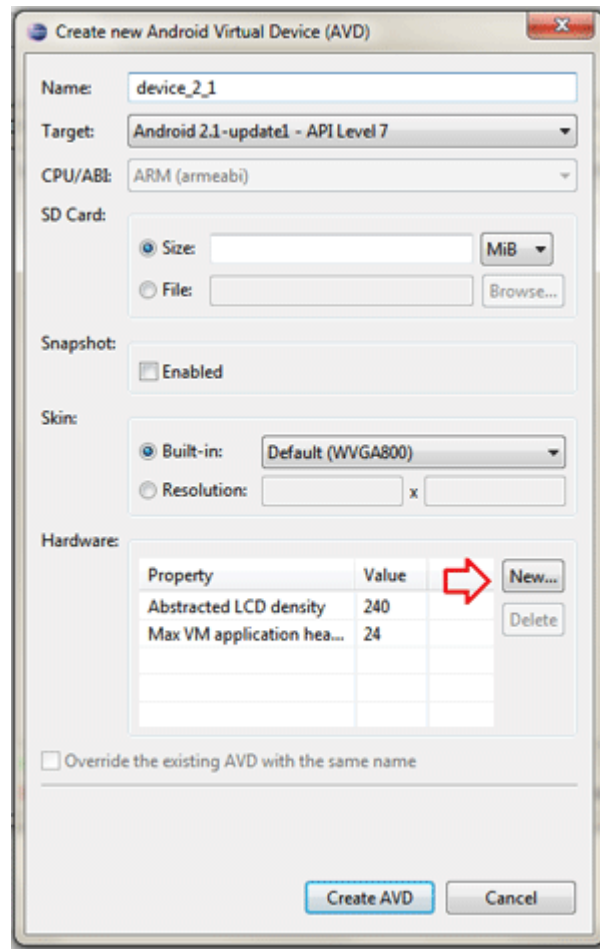
3. En la ventana *Create new Android Device (AVD)* llenar los siguientes campos:

- **Name:** El nombre que recibirá el dispositivo virtual y el que aparecerá en la bandeja de dispositivos disponibles. Se recomienda asignar un nombre descriptivo a la versión de Android con la que está configurada.
- **Target:** Es la versión de Android que correrá en el dispositivo virtual. Esta versión debe ser como mínimo la utilizada para crear los proyectos. De igual forma, se pueden crear tantos dispositivos virtuales como versiones de plataformas se tengan instaladas en Eclipse.

De forma opcional se puede llenar los campos:

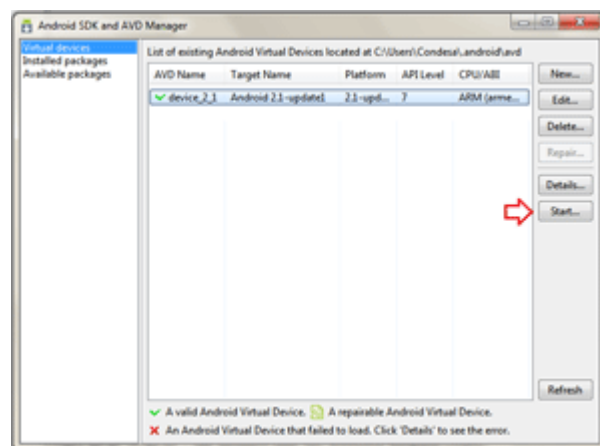
- **SD Card:** Aquí se configura lo relacionado con la SD Card. Se puede seleccionar el tamaño o cargar el archivo de una SD Card existente.
- **Skin:** Es la sección donde se configuran las características de la pantalla del dispositivo. Se puede dejar la opción por default si se tiene un monitor grande, de caso contrario se recomienda seleccionar la opción HVGA que permitirá que el emulador se adapte al tamaño de la pantalla de la computadora y se facilite ver el resultado visual.

En el último apartado de *Hardware*, se tiene la opción de agregar características de hardware más específicas del dispositivo que se vaya a emular como el acelerómetro, cámara, teclado físico, etc. Lo único que se tiene que hacer es dar clic sobre el botón *New...* que se resalta en la siguiente imagen y agregar las opciones necesarias.



En [este link](#) se puede encontrar la lista de opciones de hardware disponibles con sus respectivas descripciones.

4. Dar clic en el botón **OK** y automáticamente aparecerá esta AVD en el panel principal. Para probarla, sólo dar clic sobre el botón **Start...**



5. En la siguiente ventana dar clic sobre el botón *Launch* y se puede ver el ADV corriendo.



La primera vez que se ejecuta una AVD puede resultar tardado el proceso, así que se recomienda que cuando se estén haciendo constantes cambios a un proyecto en Android, no se detenga la AVD, es mejor dejarla corriendo en segundo plano, de esta forma cada vez que se ejecute una aplicación con las modificaciones hechas automáticamente se instalará en el dispositivo que esté corriendo (siempre y cuando sean compatibles en versiones).

→ NUEVO CONTENIDO

El administrador de Dispositivos Virtuales de Android (AVD Manager) es una interfaz gráfica de usuario muy fácil de usar para administrar configuraciones de los dispositivos virtuales Android. Se utiliza el término AVD como una configuración de dispositivo que el emulador de Android lee para emular diversos dispositivos Android. Después de crear una configuración AVD podemos iniciar el emulador con esa configuración, justo antes de lanzar el emulador nos preguntará algunas opciones para la sesión actual, como si se quiere escalar el contenido de la pantalla, borrar los datos anteriores de usuario o iniciar desde un estado previamente guardado (snapshot).

Para ejecutar el Administrador del SDK de Android (Android SDK Manager) o el AVD Manager (comandos **android** y **android avd** respectivamente) basta con agregar la ruta `<directorio_instalacion_sdk>/tools` a la variable de entorno PATH.

Usar el emulador de Android, configurado con el AVD Manager, permite realizar funciones como si de un dispositivo físico se tratase, algunas de las operaciones que podemos realizar en emulador son:

- Instalar aplicaciones.
- Simular una SD Card.
- Simular llamadas salientes.
- Mandar un SMS a otra instancia del emulador.
- Simular eventos de hardware.
- Terminar un proceso.

El emulador proporciona capacidades de red muy versátiles que se pueden usar para crear modelos complejos y ambientes de prueba, relacionados con el uso de redes, para la aplicación. Una de estas capacidades de red es el estado de la red (Network Status), la cual permite usar la consola para simular las características: estado, retraso y velocidad de la red. Por

ejemplo, para simular diferentes velocidades de transferencia en el arranque del emulador se usa el comando emulator con la opción -netspeed y pasándole como parámetro un valor de velocidad soportado, ejemplos:

```
emulator -netspeed gsm
emulator -netspeed 14.4 80
```

La lista de los valores soportados se muestra en la siguiente tabla:

Value	Description	Comments
gsm	GSM/CSD	(Up: 14.4, down: 14.4)
hscsd	HSCSD	(Up: 14.4, down: 43.2)
gprs	GPRS	(Up: 40.0, down: 80.0)
edge	EDGE/EGPRS	(Up: 118.4, down: 236.8)
umts	UMTS/3G	(Up: 128.0, down: 1920.0)
hsdpa	HSDPA	(Up: 348.0, down: 14400.0)
full	no limit	(Up: 0.0, down: 0.0)
<num>	Set an exact rate used for both upload and download.	
<up>:<down>	Set exact rates for upload and download separately.	

Esta velocidad se puede cambiar mientras la aplicación está ejecutándose en el emulador, además de brindar la capacidad de simular llamadas y envíos de SMS. Consulte <http://developer.android.com/intl/es/tools/devices/emulator.html#emulatonetworking>

Además el emulador también permite el utilizar la perspectiva DDMS de eclipse, y obtener información de depuración de aplicaciones como:

- Ver el uso de la estructura heap (espacio en memoria donde se asignan los objetos y arreglos) de un proceso.
- Examinar la información de los hilos (thread) que ocupa una aplicación, y rastrear el uso de la pila (stack) de cada hilo.
- Capturar pantalla del emulador o dispositivo.
- Realizar un seguimiento de la asignación de memoria de los objetos.
- Invocar al recolector de basura y verificar el estado actual de la estructura heap (espacio en memoria donde se asignan los objetos y arreglos).
- Suministrar datos falsos de ubicación.

4. Fundamentos de Aplicaciones Android

4.1. Desarrollo de Aplicaciones Android

Para crear un proyecto de Android en eclipse, una vez instalado el Android Development Tools Plugin (ADT), se debe iniciar un asistente llamado New Project Wizard, haciendo click en el menu File -> New -> Android Application Project, una vez terminado este asistente se crean la siguiente estructura de archivos y directorios en el nuevo proyecto:

```
src/
    Directorio para los archivos fuente de la aplicación.
gen/
    Directorio que contiene los archivos Java generados por el ADT como el R.java y los archivos AIDL.
assets/
    Directorio vacío, se puede usar para guardar archivos de datos en bruto (raw)
res/
    Directorio para los recursos de la aplicación como los archivos drawable, archivos layout, valores de cadenas, etc,
    organizados en sub directorios:

    drawable-hdpi/
        Directorio para los objetos drawable (como imágenes) que son diseñadas para pantallas de alta densidad
        (hdpi). Hay otros directorios que empiezan con "drawable-" pueden contener elementos diseñados por
        otras densidades de pantallas.
    layout/
        Directorio para los archivos que definen la interfaz de usuario de la aplicación.
    values/
        Directorio para otros archivos XML que contienen una colección de recursos como cadenas, estilos, etc.
libs/
    Librerías que necesita el proyecto.
bin/
    Contiene entre otras cosas el archivo .apk, instalable en el dispositivo Android.
AndroidManifest.xml
    El archivo AndroidManifest.xml para el proyecto.
default.properties
    Archivo que contiene las propiedades del proyecto como la API objetivo de construcción (build target).
```

4.2. Creación y estructura de un proyecto

Vease tema 4.1. Desarrollo de Aplicaciones Android

4.3. Trabajando con el archivo AndroidManifest.xml

Cada App debe tener un archivo, llamado específicamente, AndroidManifest.xml en su directorio raíz, el cual le presenta información esencial al sistema Android, que este necesita antes de correr cada componente y código de la App. Alguna de la información que contiene este archivo es:

- Establece cualquier permiso de usuario que la aplicación requiera para usar funciones de redes o elementos de hardware como camara, bluetooth, etc. El usuario final deberá aprobar estos permisos para la aplicación, una de las funciones de seguridad de Android.
- Declara el mínimo nivel de la API que la aplicación requiera.
- Declara las características de software y hardware requeridas para la aplicación.
- Describe los componentes de la aplicación: las Activities, Services, Broadcast receivers y Content providers.
- Se listan las librerías que la aplicación requiera.

El archivo, que es un recurso del proyecto, contiene una colección estructurada de elementos en XML, los cuales, junto con los atributos de cada uno de ellos sirven para proporcionarle la información descrita anteriormente al sistema Android, algunos de estos elementos y sus atributos son:

- **<manifest>**: Es el elemento raíz del archivo AndroidManifest.xml, debe contener un elemento <application> y especificar los atributos xmlns:android y package:
 - xmlns:android: Define el espacio de nombres (namespace) de Android. Siempre debe ser "<http://schemas.android.com/apk/res/android>".
 - package: El nombre completo del paquete en estilo del lenguaje java, debe ser único.
 - android:versionCode: El número interno de versión, usado para determinar si una versión es más reciente que otra.
 - android:versionName: El número de version mostrado al usuario
- **<uses-sdk>**: Expresa la compatibilidad de la aplicación con una o más versiones de Android, mediante el número de nivel de API. Atributos:
 - android:minSdkVersion="X": Designa el mínimo nivel de API requerido para correr la aplicación.
 - android:targetSdkVersion="X": Designa el nivel de API al cual se destina la aplicación.
- **<uses-permission>**: Solicita un permiso que la aplicación requiere que se le conceda para que funcione correctamente. Atributos:
 - android:name="abcd": El nombre del permiso que solicita, Ejemplos:
 - "android.permission.CAMERA" para usar la cámara,
 - "android.permission.READ_CONTACTS" para leer los contactos,
 - "android.permission.ACCESS_FINE_LOCATION" para funciones de ubicación mediante GPS, Torres de telefonía móvil o Wi-Fi.
 - "android.permission.INTERNET" para tener acceso a internet como peticiones HTTP e implementación de Sockets
- **<application>**: Elemento que contiene subelementos que declaran cada uno de los componentes de la aplicación y atributos que pueden afectar a todos los componentes. Atributos:
 - android:icon="recurso drawable": un icono para la aplicación o para cada uno de los componentes.
 - android:label="recurso string": una etiqueta para la aplicación o para cada uno de los componentes.
 - android:theme="recurso o tema": La referencia al tema que se aplicará a toda la aplicación.
 - android:debuggable=["true" | "false"]: permite establecer si la aplicación puede ser depurable, si no se especifica, por default es false.
- **<activity>**: Todas las Activities que sean parte de la aplicación deberán estar representadas por elementos <Activity>. Cualquiera que no se declare no será visto por el sistema y no se puede ejecutar. Atributos:
 - android:name="abcd": El nombre de la clase, incluyendo paquete, que implementa la Activity.
 - android:label="recurso string": una etiqueta para cada activity.
 - android:theme="recurso o tema": La referencia al tema que se aplicará a toda la Activity.
- **<intent-filter>**: Especifica el tipo de Intents a los que una Activity, Service o Broadcast receiver puede responder, es decir, las capacidades del componente padre (Activity, Service o Broadcast receiver).
- **<action>**: Agrega una action a un <intent-filter>, el cual debe contener al menos 1. Si el elemento <intent-filter> no contiene ningún <action>, no se obtendrá ningún Intent a través del filtrado de <intent-filter>.
 - android:name="abcd": El nombre de la <action>. Se pueden ocupar las actions estándar definidas por Android anteponiendo "android.intent.action." a alguna action predefinidas, o bien en el caso de actions personalizadas, para asegurar que el elemento sea único, se antepone el nombre del paquete.
- **<category>**: Contiene información adicional sobre el tipo de componente que debe manejar el Intent, puede ser más de una y como las actions también hay algunas constantes definidas por Android anteponiendo "android.intent.category." por ejemplo "android.intent.category.LAUNCHER" para listar la aplicación en lanzador de aplicaciones de Android.

Aquí un ejemplo de un archivo AndroidManifest.xml

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

```
<uses-sdk
    android:minSdkVersion="10"
    android:targetSdkVersion="10" />

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

4.4. Creación y manejo de Activities

Para crear una interfaz de usuario para una aplicación de Android, esta se debe diseñar usando una estructura de jerarquía de vistas (views) y grupos de vistas (ViewGroup). Las views, usualmente son elementos gráficos como botones o campos de texto, y ViewGroups son contenedores invisibles que definen como las views son presentadas. La clase ViewGroup es la clase base para los layouts manager, de los cuales hay varios tipos:

- **Linear Layout:** Posiciona las views hijas en una única fila o columna, dependiendo de la orientación seleccionada.
- **Frame Layout:** Su propósito es asignar un área de la pantalla.
- **Table Layout:** Organiza las views hijas en un formato de cuadrícula de filas y columnas.
- **Grid Layout:** (*Introducido como parte de Android 4.0*) Un objeto GridLayout se divide en líneas invisibles que forman una cuadrícula conteniendo filas y columnas de celdas. Las views hijas pueden ocupar una o varias celdas de forma horizontal o vertical
- **Absolute Layout:** (*Obsoleto desde la API 3*) Permite a las views hijas ser posicionadas en coordenadas específicas X y Y.
- **Relative Layout:** Posiblemente la más potente y flexible, permite, a las views hijas, posicionarse en relación a otras views y al propio contenedor.

Hay algunos otros como:

- **List View:** Una view que muestra los elementos de una lista vertical con barra de desplazamiento.
- **Grid View:** Un ViewGroup que muestra elementos en una tabla bidimensional con barra de desplazamiento
- **Tab Layout:** Permite crear una interfaz de usuario con pestañas, las cuales pueden contener views dentro de una sola Activity o usar pestañas para cambiar entre diferentes Activities.

Este diseño en estructura de jerarquías se puede construir ya sea en código, el cual en tiempo de ejecución se convierte en elementos gráficos; o definiendo elementos en un archivo XML (layout XML). Es más útil definir el archivo XML porque así podemos crear dos o más versiones para usarlas en pequeñas o grandes pantallas, según sea el caso.

Si se desea construir este diseño (layout) por código, se tienen que definir todas las propiedades llamando a métodos los cuales, en tiempo de ejecución, construirán la interfaz del usuario, aquí un ejemplo:

Para definir un LinearLayout que contenga un botón:

```
public class Principal extends Activity {
    //se crea la variable global
    Button miBoton ;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //se crea un layout
        LinearLayout miLayout = new LinearLayout(this);
        //se establece la orientacion = vertical
        miLayout.setOrientation(LinearLayout.VERTICAL);
        //Se crea un boton
        miBoton = new Button(this);
        //Etiqueta del boton
        miBoton.setText("Map");
        //se establecen los parametros del diseño
        miBoton.setLayoutParams(new LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.WRAP_CONTENT,
            LinearLayout.LayoutParams.WRAP_CONTENT));
        //se establece el manejador de Clicks al boton
        miBoton.setOnClickListener(manejadorClicks);
        //se agrega el boton al Layout anteriormente creado
        miLayout.addView(miBoton);
        // se le dice a la activity que use el layout anteriormente creado
        setContentView(miLayout);
    }

    private View.OnClickListener manejadorClicks = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if( v == miBoton ){
                //has algo
            }
        }
    }
}
```

En el caso de usar un archivo layout XML, una Activity puede usar el diseño con el método setContentView(), de la siguiente forma:

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

El plugin ADT de eclipse, permite crear estos archivos, al crear un proyecto o crear un archivo layout XML este se crea en el directorio res/layout, el cual se puede modificar directamente en texto plano o añadiendo elementos en un editor gráfico de layout (Layout Editor) y de esa forma visualizar la presentación en diferentes tamaños y orientaciones.

4.5. Uso de Intents explícitos

Vease tema 2.3. Componentes de una aplicación Subtema Intents (intenciones).

4.6. Uso de Intents implícitos

Vease tema 2.3. Componentes de una aplicación Subtema Intents (intenciones).

4.7. Creación y uso de recursos

Al ejecutar una aplicación, ya sea en el emulador de Android o en un dispositivo físico, los archivos que la aplicación usa, aparte de los archivos fuente, son manejados como recursos dedicados para la aplicación, estos se deben agrupar en directorios de recursos en `res/` con un nombre especial. El sistema Android usa automáticamente el recurso basado en la configuración del dispositivo. Aquí un breve resumen de cada tipo de recurso:

- **Recursos de animación**
Animaciones predeterminadas. Contiene subdirectorios para:
 - Animaciones "Tween", son guardadas en `res/anim/` y accesados por la clase `R.anim`.
 - Animaciones "Frame", son guardadas en `res/drawable/` y accesados por la clase `R.drawable`.
- **Lista de recursos de estado de colores**
Definen un recurso de color que cambia según el estado del componente View. Guardados en `res/color/` y accesados por la clase `R.color`.
- **Recursos dibujables**
Definen varias imágenes de mapa de bits o "bitmaps" (.png, .jpg o .gif) o en archivos XML "bitmap" (XML que apunta a un archivo "bitmap"). Guardados en `res/drawable/` y accesados por la clase `R.drawable`.
- **Recursos Layout**
Definen la disposición (layout) de la interfaz de usuario. Guardados en `res/layout/` y accesados por la clase `R.layout`.
- **Recursos de menu**
Definen los contenidos de los menús de la aplicación. Guardados en `res/menu/` y accesados por la clase `R.menu`.
- **Recursos de cadenas de texto**
Definen cadenas de texto, arreglos de cadenas de texto y "plurales" (opciones para reglas de concordancia gramatical con la cantidad, Ejemplo 1 libro, 2 libros) y pueden incluir formato y estilo de cadenas de texto. Guardados en `res/values/` y accesados por las clases `R.string`, `R.array` y `R.plurals`.
- **Recursos de estilo**
Definen el aspecto y el formato de los elementos de la interfaz de usuario. Guardados en `res/values` y accesados por la clase `R.style`.
- **Recursos de datos en bruto (raw)**
Contiene archivos que son guardados en su forma original (raw) y que se necesitan leer mediante flujos de datos (manejados por objetos Stream como `InputStream`), como archivos de audio o video.
- **Mas tipos de recursos**
Definen valores como booleanos, enteros, dimensiones, colores y otros arreglos. Guardados en `res/values` pero accesados por subclases R únicas (como `R.bool`, `R.integer`, `R.dimen`, etc).

Un recurso String provee cadenas de texto para la aplicación con estilo y formato de texto opcionales. Hay 3 tipos de recursos String:

- **String (cadenas):** recurso XML que provee cadenas simples.

- **String Array (Arreglo de cadenas):** recurso XML que provee arreglos de cadenas.
- **Quantity Strings (Plurals) (Cadenas de cantidad, plurales):** recurso XML que contiene diferentes cadenas para pluralización.

Como ejemplo de un recurso XML que contiene un Arreglo de cadenas (String Array):

- Se almacenan en el directorio: /res/values/<nombreArchivo>.xml.
- Tipos de datos: Array de String.
- Referencia al recurso: se puede referenciar en Java : R.string.<nombreArregloCadenas> y en XML: @string/<nombreArregloCadenas>

Contenido del recurso XML:

```
<?xml version="1.0" encoding="utf-8"?>
<resources> <!-- Elemento raiz -->
    <string-array <!-- Definicion de un arreglo de cadenas-->
        name="arregloCadenas"> <!-- nombre para el arreglo, sera el id del recurso-
->
            <item>Uno</item> <!--cada uno de los elementos (cadenas) del arreglo--
>
            <item>Dos</item>
        </string-array>
    </resources>
```

4.8. Seguridad y permisos

Vease tema 4.3. Trabajando con el archivo AndroidManifest.xml

4.9. Depuración de una App

Cuando se depura una aplicación, es muy común, registrar mensajes de depuración para ver, por ejemplo: el estado de las variables o el flujo del programa.

Android tiene una API para el envío de estos mensajes, comúnmente llamados Logs. Generalmente se usan los métodos: Log.v() Log.d() Log.i() Log.w() y Log.e(), la letra después de "Log." indica el nivel de detalle que se desea usar: VERBOSE, DEBUG, INFO, WARN, ERROR.

Una buena convención es usar una etiqueta o TAG para los mensajes de depuración. Ejemplo:

```
private final String ETIQUETA_ERROR = "APPERROR";

Log.E(ETIQUETA_ERROR, "Ocurrio un error en ...");
```

5. Interfaz de Usuario y Controles

5.1. Unidades y Layout

LAYOUT : Vease tema 4.4. Creación y manejo de Activities

Cuando se diseña una aplicación que admita diferentes densidades de pantalla, lo NO recomendado es usar píxeles absolutos (px) para definir tamaños o distancias, lo cual genera problemas porque cada pantalla tiene densidades de píxeles diferentes, resultando en que el mismo número de píxeles absolutos (px) corresponda a varios tamaños físicos.

Lo que SÍ es recomendado es usar píxeles independientes de la densidad que tenga el dispositivo y hay dos unidades de medición posibles:

- **dp:** para píxeles independientes de la densidad, que escala el tamaño físico de un píxel a 160dpi (puntos por pulgada), usado para tamaños de diseños.
- **sp:** igual que el anterior pero nuevamente escalada en función del tamaño de la letra configurado por el usuario, usado para tamaño de letra.

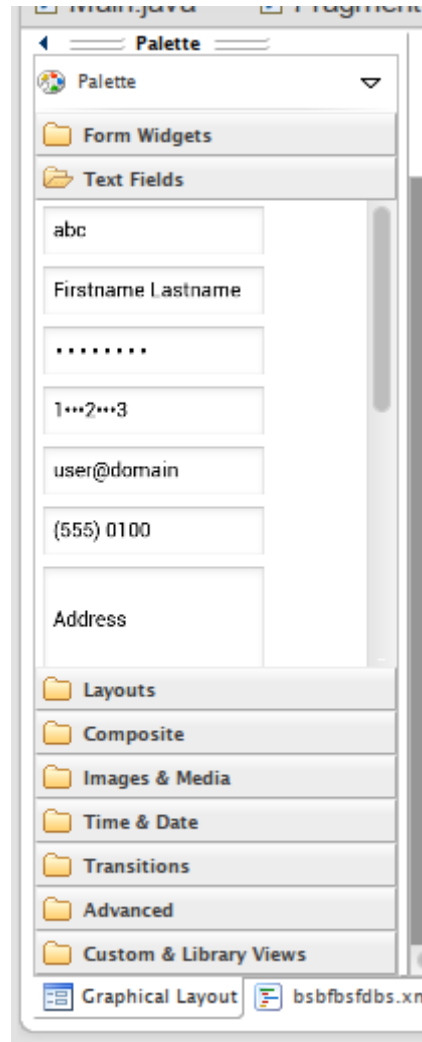
5.2. Uso de Layout Managers

Vease tema 4.4. Creación y manejo de Activities

5.3. Controles de Texto

Los controles de entrada de texto le permiten al usuario escribir texto mediante un teclado flotante en pantalla, el cual aparece cuando el usuario "toca" el control. El campo de texto puede ser de una o varias líneas y, adicionalmente, permite otras acciones, como seleccionar texto, copiarlo o pegarlo y autocompletado de texto.

Para agregar un campo de texto, desde el editor gráfico de layout en eclipse, se arrastra alguno de los tipos del objeto EditText, localizados en la paleta de componentes en forma de acordeón, ampliando el panel Text Fields, lo que añade un elemento <EditText> al archivo XML Layout.



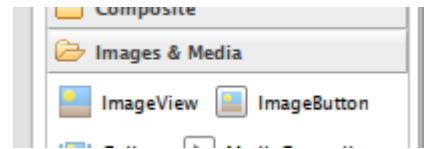
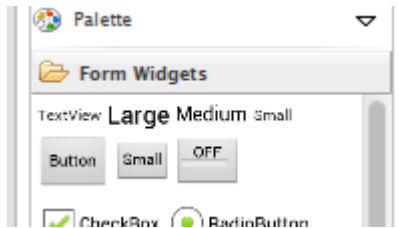
Más información:

<http://developer.android.com/guide/topics/ui/controls/text.html>

5.4. Controles de tipo Botón

Un botón es un control que consta de un texto o icono (o ambos) el cual puede responder a una acción o evento que se produce cuando el usuario lo toca.

Para agregar un botón, desde el editor gráfico de layout en eclipse, se arrastra alguno de los tipos del objeto Button, localizados en la paleta de componentes en forma de acordeón, ampliando el panel Form Widgets -> Button o en Images & Media -> ImageButton, los cuales agregan los elementos <Button> o <ImageButton>, respectivamente, al archivo XML layout.



Más información:

<http://developer.android.com/guide/topics/ui/controls/button.html>

Sobre eventos...

En Android, hay más de una forma de interceptar eventos desde la interacción del usuario con la aplicación, el enfoque principal es capturar los eventos específicos de una vista (view). Y la clase View nos proporciona los medios para hacerlo, mediante métodos "callback" que el framework de Android llama cuando la respectiva acción ocurre en esa vista (view), los cuales usan unas interfaces contenidas, también, en la clase View llamadas detectores de eventos (event listeners).

Estos detectores de eventos (event listeners) contienen un solo método "callback", son:

- **onClick():** de View.OnClickListener. Llamado cuando el usuario da click en el elemento, o presione "enter" sobre él con las teclas de navegación.
- **onLongClick():** de View.OnLongClickListener. Llamado cuando el usuario da click y sostiene el click en el elemento, o presione y mantiene el "enter" con las teclas de navegación.
- **onFocusChange():** de View.OnFocusChangeListener. Llamado cuando el usuario se desplaza sobre o fuera del elemento, usando las teclas de navegación.
- **onKey():** de View.OnKeyListener. Llamado cuando el usuario se centra en el elemento y presiona o suelta una tecla en el hardware del dispositivo.
- **onTouch():** de View.OnTouchListener. Llamado cuando el usuario realiza una acción calificada como un evento de toque, incluyendo presionar, liberar, o cualquier gesto de movimiento en la pantalla (dentro de los límites del elemento).
- **onCreateContextMenu():** de View.OnCreateContextMenuListener. Llamado cuando se esta construyendo un menú contextual (resultado de un click sostenido).

Como podemos darnos cuenta, un evento puede caer en más de una condición de las anteriores mencionadas, es por eso que algún evento pudiera generar la llamada en secuencia a varios de los detectores (si están implementados). Por ejemplo, pensemos en la simple acción de dar click en un botón, esta acción, cae en dos métodos de eventos diferentes, los cuales en orden son: onTouch() y onClick(), y como el método onTouch() se llama cuando se presione o libere el elemento, se llamará al inicio y al final de la acción. Lo que nos da como resultado el llamado de 3 métodos en secuencia:

- **onTouch(view, MotionEvent.ACTION_DOWN).** El segundo parámetro indica que fue un evento de movimiento hacia abajo (presión).
- **onClick(view).** Metodo para responder al click.
- **onTouch(view, MotionEvent.ACTION_UP).** El segundo parámetro indica que fue un evento de movimiento hacia arriba (liberación).

Como ejemplo de implementación, veamos el caso de un botón, el cual responde, entre otras cosas, a un evento onClick, algunas de las implementaciones para realizar esto son:

1. Definiendo una clase anónima

```
public class Principal extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //se obtiene la referencia a el boton definido en el layout.
        Button boton = (Button) findViewById(R.id.button1);

        //se crea un objeto de una clase anónima OnClickListener, perteneciente al paquete android.view.View
        boton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                //codigo a realizar cuando se presiona un boton.
            }
        });
    }
}
```

2. Implementando la interfaz OnClickListener

```
public class Principal extends Activity implements OnClickListener{
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //se obtiene la referencia a el boton definido en el layout.
        Button boton = (Button) findViewById(R.id.button1);
        //se hace referencia a esta clase, ya que aqui se implementa el método onClick
        boton.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        //codigo a realizar cuando se presiona un boton.
    }
}
```

3. Creando una clase interna.

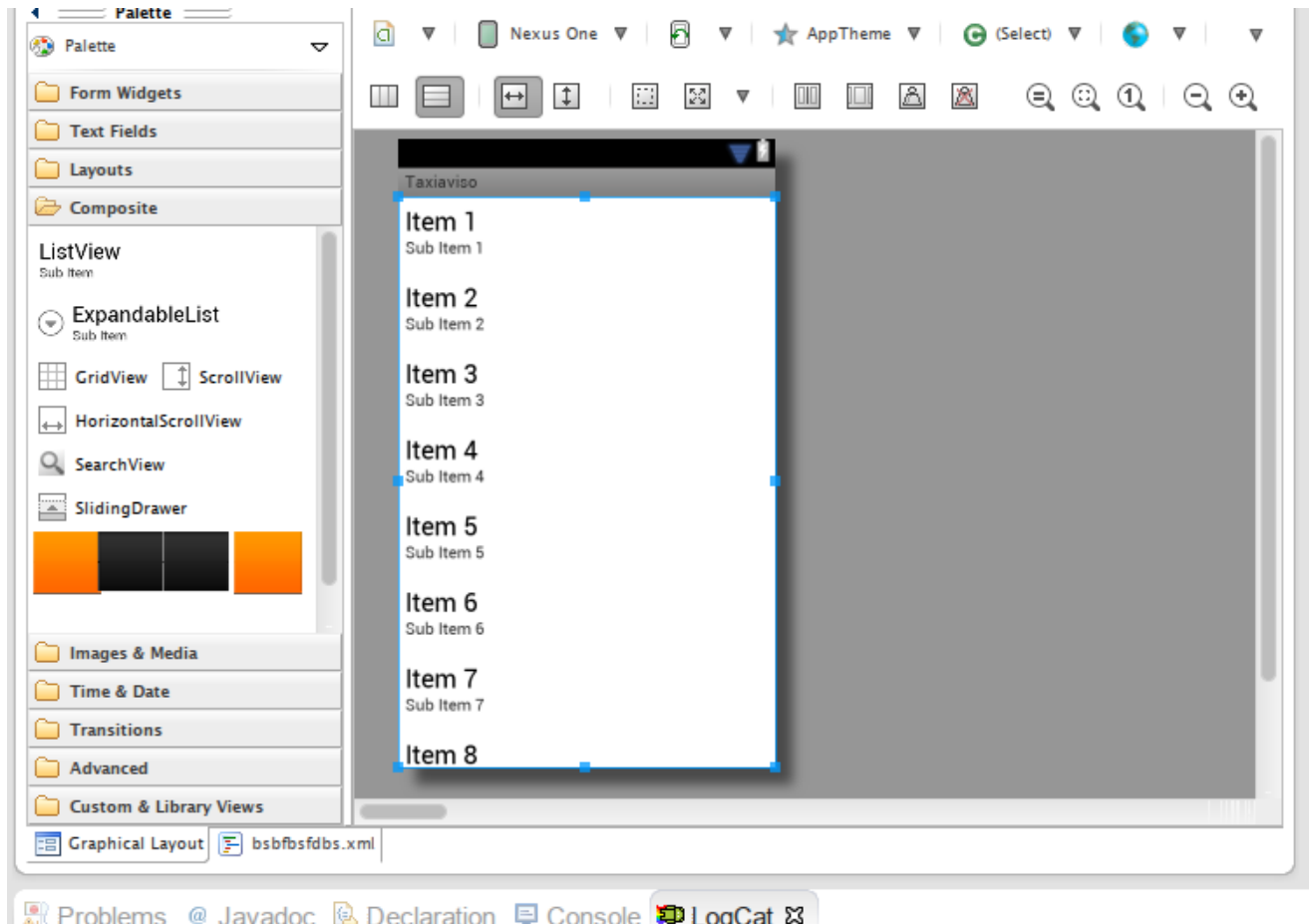
```
public class Principal extends Activity {
    //se crea la clase interna
    private OnClickListener evento = new OnClickListener(){
        @Override
        public void onClick(View v) {
            //codigo a realizar cuando se presiona un boton.
        }
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //se obtiene la referencia a el boton definido en el layout.
        Button boton = (Button) findViewById(R.id.button1);
        //se hace referencia al objeto de la clase interna
        boton.setOnClickListener(evento);
    }
}
```

5.5. Controles de tipo Lista

La vista de grupo ListView permite mostrar una lista de elementos. La lista de elementos es automáticamente insertado usando un Adapter (ListAdapter) que extrae contenido de un arreglo o una consulta a base de datos y convierte cada elemento del resultado en una vista (view) que se coloca en la lista.

Para agregar una lista, desde el editor gráfico de layout en eclipse, se arrastra el objeto ListView, localizado en la paleta de componentes en forma de acordeón, ampliando el panel Composite -> ListView, el cual agregará el elemento <ListView> al archivo XML layout.



Para la implementación de una ListView, en el código la Activity principal de la aplicación debe extender o heredar de la clase ListActivity en vez de Activity:

```
public class Lista extends ListActivity {
    //[...]
}
```

A continuación se declara el arreglo que contendrá los elementos que se desean mostrar en la lista:

```
String[] paises={"Argentina","Chile","Paraguay","Bolivia","Peru","Ecuador",
                "Brasil","Colombia","Venezuela","Uruguay"};
```

Una vez hecho esto, se crea el Adapter, el cual permite el acceso a los elementos del arreglo y también es responsable de crear una vista para cada elemento en el arreglo, en este caso será un ArrayAdapter y se añade a la ListActivity con el método setListAdapter, el cual, en uno de sus constructores, recibe en orden : el contexto, el id del recurso layout que contiene un TextView que se usará para crear la vista para cada elemento y el arreglo de Strings:

```
//[...]
setListAdapter(new ArrayAdapter<String>(this,
```



```

        android.R.layout.simple_list_item_1, paises));

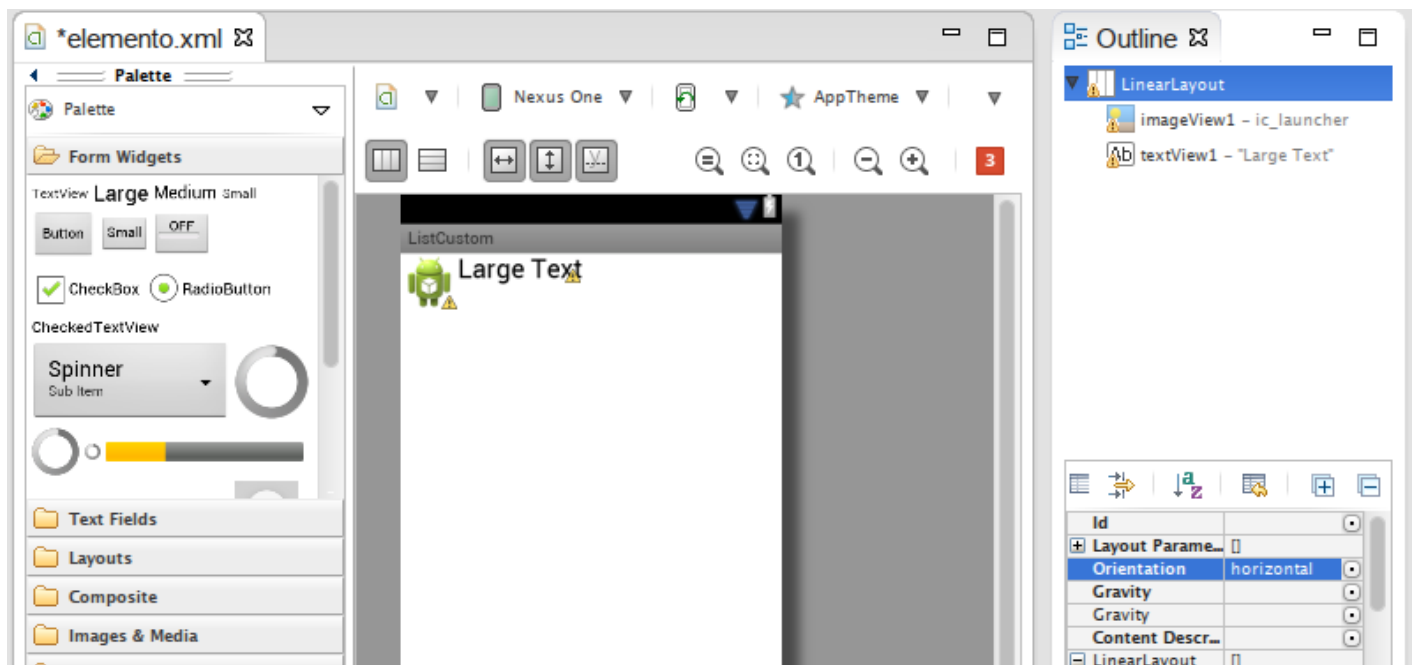
    }
    //[...]

```

5.6. Layouts de Lista Personalizados

En adición al control predeterminado de tipo lista mencionado anteriormente, es posible crear listas personalizadas, útiles para que elementos de la lista usen algún diseño creado manualmente.

Para esto primero se debe diseñar el elemento en un archivo layout que contendrá una imagen (ImageView) y un texto (TextView):



```

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

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher" />

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Large Text"
        android:textAppearance="?android:attr/textAppearanceLarge" />

</LinearLayout>

```

A continuación se declaran 2 arreglos, uno para los textos y otro para las id de las imágenes que corresponden con los textos:

```
String[] paises={"Argentina","Chile"};
int[] imagenes ={R.drawable.bandera_argentina, R.drawable.bandera_chile};
```

Ahora, se crea una clase que extiende o hereda de ArrayAdapter y en el método sobrescrito getView estableceremos el texto y el id del recurso drawable (imagen) que queremos que se muestre al lado del texto:

```
class MiAdaptador extends ArrayAdapter<String> {

    public MiAdaptador(Context context, int simpleListItem1, int textView1, String[] values) {
        super( context, simpleListItem1, textView1, values);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        LayoutInflater inflater = (LayoutInflater) getSystemService(Context.LAYOUT_INFLATER_SERVICE);

        View rowView = inflater.inflate(R.layout.elemento, parent, false);
        TextView textView = (TextView) rowView.findViewById(R.id.textView1);
        ImageView imageView = (ImageView) rowView.findViewById(R.id.imageView1);
        textView.setText(paises[position]);
        imageView.setImageResource(banderas[position]);
        return rowView;
    }
}
```

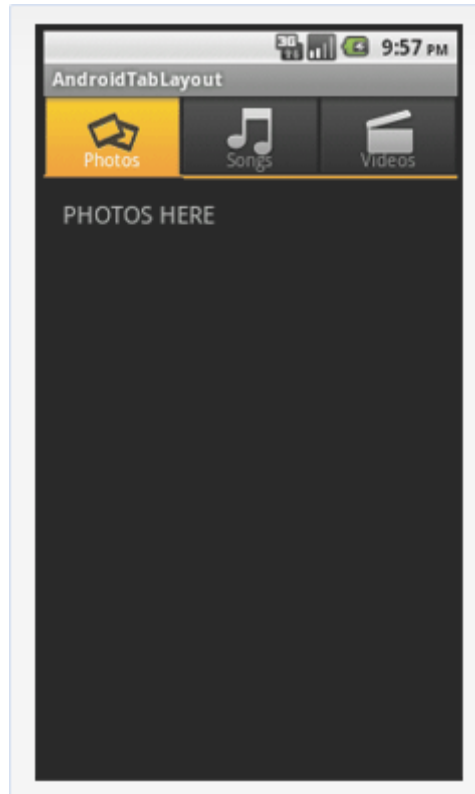
Y por último estableceremos ese adaptador propio en la ListActivity con el método setListAdapter:

```
setListAdapter(new MiAdaptador(this, android.R.layout.simple_list_item_1, R.id.textView1, paises));
```

5.7. Componentes más comunes

Aparte de los controles mencionados, Android provee muchos más controles que ayudarán a enriquecer la aplicación que se esté desarrollando, algunos de los cuales se pueden ver en funcionamiento en el emulador de Android en la aplicación "API Demos" en la sección "Views".

Como ejemplo de alguno de ellos, a continuación la implementación de un Tab Layout para cambiar entre diferentes Activities:



Se define el archivo layout XML con el elemento raíz **TabHost** que es un contenedor para una view de una ventana con pestañas, este elemento raíz tiene 2 hijos:

- **TabWidget:** que muestra un conjunto de etiquetas de las pestañas que representan cada Activity en la colección de pestañas.
- **FrameLayout:** que muestra el contenido de la Activity.

Archivo ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/contenedortabs"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"/>
    </LinearLayout>
</TabHost>
```

También se deben definir Activities para las pestañas, una por cada una, de forma normal y sobrescribir los métodos onPause() y onResume() los cuales son llamados cada vez que se sale o entra, respectivamente, de una pestaña. Ejemplo:

```
public class PrimeraActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.primer_layout);
    }

    @Override
    protected void onPause() {
        // TODO Auto-generated method stub
        super.onPause();
        Log.d("ACB", "Primera Actividad Pausada");
    }

    @Override
    protected void onResume() {
        // TODO Auto-generated method stub
        super.onResume();
        Log.d("ACB", "Primera Actividad Resumida");
    }
}
```

La Activity principal debe extender o heredar de TabActivity y se debe implementar la agregación de las Activities a las pestañas. Ejemplo:

```
public class ContieneTabsActivity extends TabActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //se obtiene la instancia de TabHost
        TabHost tabHost = getTabHost();

        // Se crea un TabSpec, que es un paquete conteniendo lo necesario para crear y navegar por la
        // pestaña
        TabSpec primerSpec = tabHost.newTabSpec("Primera etiqueta");
        // especifica el titulo de la pestaña
        primerSpec.setIndicator("Photos");
        // se crea un intent para poder lanzar la Activity
        Intent primerIntent = new Intent(this, PrimeraActivity.class);
        // se especifica el intent que lanzara la Activity
        primerSpec.setContent(primerIntent);

        // se agrega el TabSpec al TabHost
        tabHost.addTab(primerSpec); // Adding photos tab

        /*
        *
        * se agregan mas tabs para mas Activites de la misma forma...
        *
        */
    }
}
```

6. Estilos y Elementos Gráficos

6.1. Creación y uso de Estilos

Los temas, son el mecanismo que Android proporciona para aplicar un estilo coherente a una aplicación completa o a una sola Activity. Un estilo es una colección de propiedades que especifican el aspecto y el formato de una vista o ventana, especificando las propiedades visuales de los elementos que componen la interfaz de usuario, como el color, la altura y el tamaño de fuente. La plataforma de Android 4.0 (Ice Cream Sandwich) provee 3 temas que se pueden usar al construir aplicaciones:

- Holo Light
- Holo Dark
- Holo Light with dark action bars

Un estilo es definido en un recurso XML, guardado en el directorio `/res/value`, que es separado del XML que especifica el diseño (layout), permitiendo separar el diseño del contenido. El elemento raíz, de este archivo XML estilo, debe ser `<resources>`, y por cada estilo que se desee crear, se debe agregar un elemento hijo `<style>` con el argumento obligatorio `name=""` que declara el nombre de ese estilo, luego como elementos hijos del anterior, se deben agregar elementos `<item>` que declaran las propiedades que se aplicarán al usar ese estilo, ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="MiEstilo" parent="@android:style/TextAppearance.Medium">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

El atributo `parent` en el elemento `<style>` es opcional y especifica el ID del recurso, de otro estilo, del cual se quiere heredar las propiedades, sobrescribiendo algunas de ellas.

Los elementos hijos `<style>`, son convertidos, en tiempo de compilación, en recursos de la aplicación que pueden ser referenciados por valor con el atributo `name` del elemento `<style>`, en un archivo de diseño XML (layout), ejemplo:

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

Otra forma de aplicar estilos, esta vez a una Activity o a la aplicación completa, es agregando el atributo `android:theme` al elemento `<activity>` o `<application>` en el archivo `AndroidManifest.xml`, Ejemplo:

```
<application android:theme="@style/MiTema">

    o

<activity android:theme="@style/TemaAplicacion" >
```

Si se quiere aplicar un tema a una Activity de forma dinámica en tiempo de ejecución, se debe llamar al método `setTheme()` pasándole como parámetro el identificador del recurso. Esto se debe hacer antes de llamar al constructor de la clase padre y llamar al método que establece el contenido de la Activity `setContentView()`, ejemplo:

```
@Override
public void onCreate(Bundle savedInstanceState)
{
    setTheme(android.R.style.Theme_Black);
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}
```

6.2. Creación y uso de Temas

Vease tema 6.1. Creación y uso de Estilos

6.3. Creación de Iconos

Una parte importante para la originalidad y el reconocimiento de la aplicación en desarrollo es el proveer a la aplicación con un diseño bien formado como, por ejemplo, iconos personalizados. La página de desarrolladores de Android provee una extensa guía y recursos que son útiles para esta tarea.

En esta guía en línea se proveen las características, formato, tamaños y todo lo requerido por Android para la correcta creación de iconos, estilos, temas, etc para la aplicación. Como ejemplo, nuevamente, en la página:

<http://developer.android.com/design/style/iconography.html>

se proveen reglas que deben seguir los iconos que se colocaran en el Launcher de aplicaciones del dispositivo, entre otras cosas resalta la siguiente tabla, la cual indica los tamaños exactos que debe tener los iconos para presentarlos en las diferentes densidades de pantalla:

	ldpi (120 dpi) (Low density screen)	mdpi (160 dpi) (Medium density screen)	hdpi (240 dpi) (High density screen)	xhdpi (320 dpi) (Extra-high density screen)
Launcher Icon Size	36 x 36 px	48 x 48 px	72 x 72 px	96 x 96 px

Y en la página :

<http://developer.android.com/design/downloads/index.html>

se proveen recursos de ejemplo que podemos usar como base, modificandolos para crear diseños personalizados.

6.4. Creación de imágenes 9-Patch

La herramienta draw9patch permite crear una imagen PNG que se escalara solo en la forma deseada, y cuando el componente al cual se le agrega excede los límites de la imagen. Por ejemplo si se quiere poner un fondo en un botón, el tamaño de estos pueden variar dependiendo de varias cosas, así que el fondo se debe estirar para acomodarse a diferentes tamaños. El recurso NinePatch generado por esta herramienta es una imagen PNG que contiene un borde extra de un tamaño de 1 píxel, utilizado para definir las zonas estirables y las estáticas, y se guarda con la extensión .9.png dentro del directorio res/drawable del proyecto.

7. Soporte para Múltiples Pantallas

7.1. Comprensión de tamaño de pantalla y densidad

Ya que Android corre en una diversidad de dispositivos, ya sean teléfonos móviles o tabletas. Android debe lidiar con el proceso de adaptación a múltiples tamaños de pantallas. Lo cual, por fortuna, Android facilita este proceso, pero primero veamos algunos conceptos:

- **Tamaño de pantalla** : Es el tamaño físico de la pantalla en diagonal.
- **Densidad de pantalla** : La cantidad de píxeles dentro de una área física (una pulgada) de la pantalla, a mayor densidad mayor resolución, esto se conoce como DPI o píxeles por pulgada, por sus siglas en inglés.
- **Píxeles independientes de la densidad** : Un píxel virtual que escala el tamaño físico de un píxel a 160dpi.

Android divide el rango de tamaños de pantallas en :

- **small** : al menos 426dp x 320dp
- **normal** : al menos 470dp x 320dp
- **large** : al menos 640dp x 480dp
- **xlarge** : al menos 960dp x 720dp

y las densidades de pantalla en:

- **ldpi** : Para pantallas de una baja densidad (low-density) (~120dpi).
- **mdpi** : Para pantallas de una densidad media (medium-density) (~160dpi). (La densidad base.)
- **hdpi** : Para pantallas de una alta densidad (high-density) (~240dpi).
- **xhdpi** : Para pantalla de una extra alta densidad (extra high) (~320dpi)
- **xxhdpi** : Para pantallas de una extra extra alta densidad (extra extra high density) (~480dpi)

La siguiente imagen muestra algunas de estos tamaños y densidades:

	Low density (120), <i>ldpi</i>	Medium density (160), <i>mdpi</i>	High density (240), <i>hdpi</i>	Extra high density (320), <i>xhdpi</i>
<i>Small screen</i>	QVGA (240x320)		480x640	
<i>Normal screen</i>	WQVGA400 (240x400) WQVGA432 (240x432)	HVGA (320x480)	WVGA800 (480x800) WVGA854 (480x854) 600x1024	640x960
<i>Large screen</i>	WVGA800** (480x800) WVGA854** (480x854)	WVGA800* (480x800) WVGA854* (480x854) 600x1024		
<i>Extra Large screen</i>	1024x600	WXGA (1280x800)[†] 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1536 2560x1536 2560x1600

Ademas tambien estan las opciones de orientación de la pantalla, landscape (para la vista horizontal) o portrait (para la vista vertical).

Para que Android pueda lidiar por sí solo con todas estas diferentes densidades, tamaños y orientaciones de pantalla, se deben proveer recursos en todos esos tamaños y orientaciones, permitiendo con esto que Android seleccione automáticamente los recursos para la pantalla actual en tiempo de ejecución, lo cual se hace separando esos recursos por carpetas dentro de la carpeta res del proyecto, en la siguiente estructura:

Para recursos en diferentes densidades de pantalla:

```
MiProyecto/
  res/
    drawable-xhdpi/
      awesomeimage.png
    drawable-hdpi/
      awesomeimage.png
    drawable-mdpi/
      awesomeimage.png
    drawable-ldpi/
      awesomeimage.png
```

Para diseños layout en diferentes orientaciones de pantalla:

```
MiProyecto/
  res/
    layout/                # default (portrait)
      main.xml
    layout-land/           # landscape
      main.xml
    layout-large/          # large (portrait)
      main.xml
    layout-large-land/     # large landscape
      main.xml
```


Además también se recomienda usar píxeles independientes de la densidad (como dp o sp) o las opciones wrap_content o fill_parent cuando se especifican las dimensiones de un componente al diseñar el archivo XML Layout.

Más información:

http://developer.android.com/guide/practices/screens_support.html

7.2. Incluir Layouts alternos

Vease tema 7.1. Comprensión de tamaño de pantalla y densidad

8. Animación y Gráficos

8.1. Configuración de animación cuadro por cuadro

Para crear una animación cuadro por cuadro se debe hacer lo siguiente:

- Tener una serie de imágenes individuales guardadas en la carpeta res/drawable.
- Crear un archivo XML de tipo animation en la carpeta res/anim conteniendo, en orden, todas las imagenes, la duración y la propiedad android:oneshot="false | true" que establecerá si se reproduce la animación solo una vez:

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list
xmlns:android="http://schemas.android.com/apk/res/android"
android:oneshot="false">
    <item android:drawable="@drawable/blank" android:duration="210" />
    <item android:drawable="@drawable/logo" android:duration="210" />
    <item android:drawable="@drawable/logo1" android:duration="210" />
    <item android:drawable="@drawable/logo2" android:duration="210" />
    <item android:drawable="@drawable/logo3" android:duration="210" />
    <item android:drawable="@drawable/logo4" android:duration="210" />
    <item android:drawable="@drawable/logo5" android:duration="210" />
    <item android:drawable="@drawable/logo6" android:duration="210" />
    <item android:drawable="@drawable/logofinal" android:duration="210" />
</animation-list>
```

- Agregar al archivo XML layout de nuestra aplicación un objeto ImageView:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent" >
    <ImageView
        android:id="@+id/imageAnimation"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:adjustViewBounds="true" />
</RelativeLayout>
```

- En el código dentro del método onCreate(), obtener la referencia a ese imageView y declarar una variable de tipo AnimationDrawable:

```
ImageView view = (ImageView) findViewById(R.id.imageAnimation);
AnimationDrawable frameAnimation;
```

- Establecer el recurso animation-list, declarado anteriormente como recurso background del objeto ImageView y a su vez obtener ese recurso background y castearlo al tipo AnimationDrawable:

```
view.setBackgroundResource(R.drawable.animation_list);
frameAnimation = (AnimationDrawable) view.getBackground();
```

- Iniciar la animación llamando al método start():

```
frameAnimation.start();
```

8.2. Mostrar animación sincronizada

Una animación sincronizada o "tween animation" puede realizar series de transformaciones simples (posición, tamaño, rotación y transparencia) en el contenido de un objeto View. Por ejemplo, si tienes un objeto TextView puedes moverlo, rotarlo, aumentarlo, disminuirlo o cambiarle la transparencia al texto.

Estas transformaciones, dadas por las instrucciones de animación, pueden estar escritas en código pero es recomendable, escribirlas en un archivo XML.

Las instrucciones de la animación (<translate>, <rotate>, <scale>, <alpha>) pueden ser secuenciales o simultáneas y cada tipo de transformación tiene parámetros específicos, aunque algunos parámetros son comunes a todas las transformaciones como la duración.

Para crear una animación sincronizada (tween animation):

- Crear un archivo dentro de la carpeta res/anim con alguna de las instrucciones de transformación o con varias de ellas agrupadas en un elemento <set>, por ejemplo para la transformación escalar, el contenido del archivo será:

```
<scale xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="2000"
    android:fromXScale="2.0"
    android:fromYScale="2.0"
    android:toXScale="1.0"
    android:toYScale="1.0" />
```

- Añadir al archivo XML layout de nuestra aplicación el objeto al que se le quiera aplicar la transformación:

```
<Button
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
```

```
android:id="@+id/boton"  
android:text="@string/texto" />
```

- En el código, método onCreate(), obtener la referencia a ese botón, declarar una variable de tipo Animation y cargarla con el método estático AnimationUtils.loadAnimation() e iniciar la animación con el método startAnimation() de la View a la que queremos aplicar la transformación, en este caso el botón.

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
    TextView texto = (TextView) findViewById(R.id.textView);  
    Animation animacion = AnimationUtils.loadAnimation(this,  
R.anim.animacion);  
    texto.startAnimation(animacion);  
}
```

8.3. Trabajar con Gráficos 2D

Android puede hacer uso de gráficos 2D usando la API de dibujo, la cual, entre otras cosas, puede leer un archivo XML que define algunas formas gráficas. Este archivo estará almacenado en la carpeta res/drawable y conteniendo la declaración de la forma y las propiedades que tendrá la forma gráfica que se dibujara.

Por ejemplo, para dibujar un rectángulo con un color degradado, un relleno en contenido (padding) y una esquinas redondeadas, el archivo rectangulo.xml contendrá:

```
<?xml version="1.0" encoding="UTF-8"?>  
<shape xmlns:android="http://schemas.android.com/apk/res/android"  
    android:shape="rectangle">  
    <stroke android:width="1px" android:color="#c50200" />  
    <gradient  
        android:startColor="#f50202"  
        android:centerColor="#dc0100"  
        android:endColor="#c50200"  
        android:angle="270" />  
    <padding android:left="2px" android:top="2px"  
        android:right="2px" android:bottom="2px" />  
    <corners android:radius="5dp" />  
</shape>
```

Ahora, como ejemplo, para aplicar esa forma gráfica a un botón basta con poner ese recurso drawable como background:

```
<Button  
    android:id="@+id/button1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignParentLeft="true"  
    android:background="@drawable/rectangulo"  
    android:text="Boton" >
```

9. Menús y Diálogos

9.1. Opciones de Menú

En Android hay tres tipos fundamentales de Menús basadas en la API de Menu.

- **Menu de opciones y la barra de acción:** Contiene acciones que impactan de forma global en la aplicación, por ejemplo “Preferencias”.
- **Menu contextual:** un menu flotante que aparece cuando el usuario realiza un “click” sostenido en un elemento.
- **Menu emergente (Popup):** una lista vertical de elementos que esta anclada a la vista (Button, TextView, etc) que lo invoca, es una buena opción para proveer más acciones que se relaciona con contenido específico.

Los Menús se pueden definir en Android en el formato XML, o mediante código el cual generara el menu en tiempo de ejecución. Para la opción que involucra el archivo XML, este se debe crear en el directorio `res/menu/` y debe contar con los siguientes elementos:

- **<menu>**: Define el elemento raíz del archivo y puede contener uno o más elementos `<item>` y `<group>`.
- **<item>**: Representa un solo elemento del menu, manejado en el código por la interfaz `MenuItem`. Este elemento puede contener elementos `<menu>` anidados con el fin de crear submenús.
- **<group>**: es opcional, contiene elementos `<item>` de forma invisible. Permite categorizar los elementos del menu para facilitar características compartidas por varios elementos, como estado o visibilidad.

Otra forma de uso muy útil para el elemento `<group>` es usar checkbox para activar o desactivar opciones agregando el atributo `android:checkableBehavior` en el elemento `<group>`, pudiendo seleccionar una sola opción de un grupo de esta forma:

```
<group android:checkableBehavior="single">
```

Nota: La forma anterior no funciona con el menu de opciones que solo muestran iconos y un pequeño texto.

Aquí un ejemplo de implementación de un menu de opciones. Lo primero será crear el archivo XML que describirá al menu:

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

    <item android:id="@+id/opcion1"
        android:title="Hola!"
        android:icon="@android:drawable/ic_menu_compass"/>

    <item android:id="@+id/opcion2"
        android:title="Adios!"
        android:icon="@android:drawable/ic_menu_call"/>

</menu>
```

Nota: El atributo `android:icon=""` es opcional.

Ahora, para asociarlo a una Activity, se debe sobrescribir el método `onCreateOptionsMenu(Menu menu)` en la Activity (para mostrar el menu que se muestra con el botón Menu en dispositivos que tengan ese botón y en sistemas android anteriores a la API 3.0), el parámetro `Menu` que recibe este método nos permitirá trabajar directamente con el menu, inicialmente se “infla” el menu a partir del recurso `mimenu.xml`:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
```

```
        getMenuInflater().inflate(R.menu.mimenu, menu);  
        return true;  
    }  
}
```

Para responder a cada elemento del menu sobreescribimos el método `onOptionsItemSelected()` (`MenuItem item`):

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    int itemId = item.getItemId();  
    switch (itemId) {  
        case R.id.opcion1:  
            //codigo para opcion1  
            break;  
        case R.id.opcion2:  
            //codigo para opcion2  
            break;  
    }  
    return super.onOptionsItemSelected(item);  
}
```

Alternativamente a la implementación ya mencionada, se puede crear un menu en tiempo de ejecución añadiendo a la variable `menu`, del método `onCreateOptionsMenu()` (`Menu menu`), cada uno de los elementos del menu llamando a alguno de los métodos sobrecargados `.add`

```
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
    menu.add(Menu.NONE, R.id.menu_addBeer, Menu.NONE, R.string.menu_addbeer);  
    menu.add(Menu.NONE, R.id.menu_settings, Menu.NONE, R.string.menu_settings);  
    return true;  
}
```

El método `.add` de la interfaz `Menu` está sobrecargado para aceptar 4 formas diferentes y regresa un objeto o instancia implementando la interfaz `MenuItem`, las 4 sobrecargas de este método son:

```
add(CharSequence title)  
    Agrega un elemento MenuItem recibiendo una cadena de texto.  
add(int groupId, int itemId, int order, int titleRes)  
    Agrega un MenuItem recibiendo números enteros para el idgrupo, itemId y el orden, como último  
    parámetro el id del recurso cadena de texto (R.id.)  
add(int titleRes)  
    Agrega un MenuItem, recibiendo únicamente el id del recurso que contiene un item de menu.  
add(int groupId, int itemId, int order, CharSequence title)  
    Agrega un MenuItem, similar al segundo caso, pero como último parámetro recibe una cadena de texto.
```

En Android, un elemento del menu puede lanzar directamente una Activity mediante un Intent, y para la diversificación de la aplicación en múltiples dispositivos Android, dispone de un mecanismo para crear dinámicamente cada elemento del menu ya sea que el dispositivo cuente o no con una aplicación que pueda lidiar con esa opción. Esta implementación se hace en el método `onCreateOptionsMenu()` (`Menu menu`) primero definiendo un Intent con la categoría `CATEGORY_ALTERNATIVE` y/o `CATEGORY_SELECTED_ALTERNATIVE`, y despues llamando al método `Menu.addIntentOptions()`, de esta forma Android hace una búsqueda para alguna aplicación que pueda lidiar con el Intent y solamente si encuentra alguna, agregara esa opcion al menu.

A continuación la implementación:

```
@Override
public boolean onCreateOptionsMenu(Menu menu){
    super.onCreateOptionsMenu(menu);

    Intent intent = new Intent(null, Uri.parse(""));
    intent.addCategory(Intent.CATEGORY_ALTERNATIVE);

    menu.addIntentOptions(
        R.id.intent_group, // grupo de menu en el cual se agregara
        0, // id unico
        0, // orden para el item
        this.getComponentName(), // nombre de la Activity actual
        null, // item especifico que se colocara primero
        intent, // Intent creado anteriormente
        0, // Banderas adicionales
        null); // Arreglo de MenuItem's que se relacionan con items especificos
    return true;
}
```

9.2. Menús Contextuales

La implementación de los menús contextuales es similar a la de los menús de opciones, también se crea el archivo XML que describe al menu, la diferencia radica en que se debe asociar el menu a un elemento (Button, TextView, etc) que responderá al "click" sostenido del usuario, haciendo que aparezca el menu contextual, esto se hace en el método onCreate():

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

    //se obtiene la referencia a algun componente
    Button b = (TextView)findViewById(R.id.button1);

    //Asociamos el menú contextual a el componente
    registerForContextMenu(b);
}
```

Lo siguiente será "inflar" el archivo XML que describe al menu en el método onCreateContextMenu():

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo)
{
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu_contextual, menu);
}
```

Y para responder a cada elemento del menu, se sobrescribe el metodo onOptionsItemSelected():

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

```
switch (item.getItemId()) {
    case R.id.opcion1:
        //codigo para opcion1
        return true;
    case R.id.opcion2:
        //codigo para opcion2
        return true;
    default:
        return super.onContextItemSelected(item);
}
```

9.3. Diálogos de Alerta

Un diálogo es una pequeña ventana que presenta información al usuario y/o para que tome alguna decisión, no ocupa toda la pantalla y normalmente se usa para eventos modales.

La clase `Dialog` es la clase base para los diálogos, pero se debe evitar usarla directamente. En vez de eso se debe usar alguna de las subclases `AlertDialog`, `DatePickerDialog` o `TimePickerDialog`. Estas clases definen el estilo y la estructura para el diálogo según sea el caso, y se recomienda que estén en contenedores `DialogFragment` (`android.support.v4.app.DialogFragment`), que proveen todos los controles necesarios para crear el diálogo y manejar su apariencia.

Para crear un diálogo básico, se sobreescribe el método `onCreateDialog(Bundle savedInstanceState)` en la clase que extienda o herede de la clase `DialogFragment`:

```
public class MiDialogo extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        return new AlertDialog.Builder(getActivity()).setMessage("Esto es un
mensaje").create();
    }
}
```

Cuando se crea una instancia de la clase anterior y se llama al método `show()` del objeto, el diálogo aparecerá.

La implementación rápida, pero no recomendada para crear una diálogo básico es la siguiente:

```
AlertDialog.Builder ab = new AlertDialog.Builder(Principal.this);
ab.setMessage("Seguro que quiere salir?");
ab.setCancelable(false);
ab.setPositiveButton("Si!", new DialogInterface.OnClickListener() {

    @Override
    public void onClick(DialogInterface dialog, int which) {
        // TODO Auto-generated method stub

        Principal.this.finish();
    }
});
```

```
ab.setNegativeButton("No!", new DialogInterface.OnClickListener() {

    @Override
    public void onClick(DialogInterface dialog, int which) {
        // TODO Auto-generated method stub
        dialog.cancel();
    }

});
AlertDialog ad = ab.create();
ad.show();
```

Se puede agregar una lista de opción simple o múltiple (radio buttons o checkboxes respectivamente). Para implementar una lista de opción simple se usa el método `setSingleChoiceItems()` el cual está sobrecargado de 4 formas, una de ellas recibe, en orden, los parámetros:

1. Arreglo de cadenas de texto (`CharSequence[]` elementos)
2. Número entero de la opción inicialmente seleccionada (`int seleccion`)
3. Detector (listener), una clase que implemente la interfaz `DialogInterface.OnClickListener` al cual se le notificará cuando se seleccione una opción.

Ejemplo:

```
public class MiDialogoOpciones extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setTitle("Seleccione una opcion");
        String[] opciones = {"Opcion1", "Opcion2", "Opcion3"};
        builder.setSingleChoiceItems(opciones, 1, null); //el listener por ahora
        es nulo.
        return builder.create();
    }
}
```

9.4. Diálogos de Progreso

Una barra de progreso es útil para indicar al usuario que algo está pasando y da al usuario una idea rápida de cuánto tardará la operación. Android incluye un tipo de diálogo de progreso en la clase `ProgressDialog`, que muestra un diálogo con una barra de progreso (extiende de `android.app.AlertDialog`), este diálogo puede solamente usar un mensaje de texto o una vista (view) al mismo tiempo, el rango de progreso es de 0 a 10000, y se puede hacer cancelable con la tecla "back". El diálogo de progreso se puede crear de dos estilos diferentes, ambos aplicados con el método `setProgressStyle(int estilo)`:

- barra de progreso horizontal: definida por la constante `STYLE_HORIZONTAL`
- barra de progreso circular girando: definida por la constante `STYLE_SPINNER`

Una de las opciones que ofrece esta clase es la posibilidad de personalizar la barra de progreso mostrada para utilizar un recurso `Drawable`, lo que se logra llamando al método `setProgressDrawable(Drawable d)`.

Implementación:


```
final ProgressDialog pb = new ProgressDialog(this);
pb.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
pb.setMessage("Esperando");
pb.setCancelable(true);
pb.setIndeterminate(true);

Button b3 = (Button) findViewById(R.id.button3);
b3.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        pb.show();
        pb.setProgress(50);
    }
});
```

9.5. Diálogos Personalizados

Si se desea crear un diálogo personalizado a partir de un archivo xml de diseño (layout), se utiliza el método `setView()` en el objeto `AlertDialog.Builder` para agregar al `AlertDialog`.

El método `setView()` recibe como parámetro un objeto `View`, para lo cual se debe "inflar" en archivo xml de diseño, para así obtener el objeto `View`, Ejemplo:

```
public class MiDialogoPersonalizado extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        AlertDialog.Builder constructorDialogo = new
        AlertDialog.Builder(getActivity());
        // Se obtiene el objeto que permite "inflar" el archivo xml
        LayoutInflater inflater = getActivity().getLayoutInflater();

        // Se "infla" el archivo, identificado por R.layout.<nombreArchivo>
        // El segundo parámetro (el View padre) es null, porque el dialogo lo va a
        // contener.
        constructorDialogo.setView(inflater.inflate(R.layout.mi_dialogo, null));
        //se crea el dialogo y se regresa como resultado de este metodo.
        return constructorDialogo.create();
    }
}
```

Otra implementación, más directa, pero no recomendada por la guía de Desarrollo de Android es usando directamente la clase `Dialog`, en cualquier parte del código de esta forma:

```
[...]
Dialog dialog = new Dialog(MainActivity.this);
dialog.setTitle("Dialog");
dialog.setContentView(R.layout.mi_dialogo);
```

```
dialog.show();  
[...]
```

10. Notificaciones y Toast

10.1. Desplegar notificaciones en barra de estatus

Una notificación es un mensaje que se puede mostrar al usuario fuera de la interfaz normal de la aplicación. Aparece como un icono en el área de notificación en la pantalla del dispositivo Android y es posible hacer que vaya acompañada con sonido, vibración y encender el LED del dispositivo (si cuenta con el) al emitir la notificación, para mostrar los detalles cuando el usuario abre el buzón de notificaciones.

Las notificaciones primero se muestran en su vista normal, la primera que se muestra al usuario, en la barra de estado. O en su forma de vista ampliada, que es cuando se expande el buzón de notificaciones o se expande la notificación. Ambas vistas pueden contener iconos en diferente tamaño.

Para implementar una notificación:

- Se especifica la información y las acciones (para ir directamente a una Activity desde la notificación con un PendingIntent llamando al método `setContentIntent()` de la interfaz de usuario en el objeto `NotificationCompat.Builder`).
- Para crear la notificación en sí, se llama al método `NotificationCompat.Builder.build()`, que regresa un objeto `Notification`.
- Para emitir la notificación se le pasa el objeto `Notification` al sistema llamando al método `NotificationManager.notify()`.

Ejemplo:

[Implementación Nueva:]

NOTA: Para este ejemplo se crea un backstack artificial para fines de mantener la navegación al presionar el botón Home.

```
NotificationCompat.Builder constructorNotificacion = new NotificationCompat.Builder (this)  
    //icono para la vista normal  
    .setSmallIcon(R.drawable.notification_icon)  
    .setContentTitle("Mi notificacion")  
    .setContentText("Primera notificacion!")  
    //se activan TODAS las opciones: la vibración, el sonido y el parpadeo del LED.  
    //Si se desea activar alguna de ellas Reemplazar Notification.DEFAULT_ALL por  
    //Notification.DEFAULT_SOUND, Notification.DEFAULT_VIBRATE, Notification.DEFAULT_LIGHTS  
    //o combinando dos de ellas con la comparacióni bit a bit, ejemplo:  
    //DEFAULT_SOUND | DEFAULT_VIBRATE  
    .setDefaults(Notification.DEFAULT_ALL);  
//Se establece el icono para la vista ampliada  
Bitmap bm = BitmapFactory.decodeResource(getResources(), R.drawable.large_icon);  
constructorNotificacion.setLargeIcon(bm);  
    //se crea un intent para una Activity  
Intent resultIntent = new Intent(this, miActivity.class);  
  
    //Se crea el backstack artificial  
TaskStackBuilder pilaArtificial = TaskStackBuilder.create(this);  
    //Se agrega el la activity a la pila artificial para el intent  
pilaArtificial.addParentStack(miActivity.class);  
    //se agrega el Intent que inicia la Activity y la coloca en el la parte superior de la pila  
pilaArtificial.addNextIntent(resultIntent);  
    //Se crea el PendingIntent que lanzara la activity desde la notificacion
```

```
PendingIntent resultPendingIntent = pilaArtificial.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);
//se asocia el PendingIntent a la notificacion
constructorNotificacion.setContentIntent(resultPendingIntent);

//se obtiene la referencia al NotificationManager
NotificationManager mNotificationManager =
(NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
//se establece que se cancele automáticamente la notificación cuando el usuario hace "click" sobre
ella.
constructorNotificacion.setAutoCancel(true);
//Se muestra la notificación, el primer parámetro es un id que servirá para identificar la notificación
mNotificationManager.notify(3, constructorNotificacion.build());
```

[Implementación Anterior:] *Obsoleta desde la API 11. Usar NotificationCompat.Builder.*

```
//se obtiene la referencia al NotificationManager
NotificationManager nm = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
//se crea un objeto Notification, parametros: icono para vista normal, texto, hora de notificacion
Notification notificacion = new Notification(android.R.drawable.stat_notify_more, "Mensaje...",
                                           System.currentTimeMillis());

Context contexto = miActivity.this;
CharSequence titulo = "Has recibido una notificacion";
CharSequence descripcion = "Continua con lo que haces";
//Se crea el PendingIntent que lanzara la activity desde la notificacion
Intent intent = new Intent(contexto, miActivity.class);
PendingIntent pendiente = PendingIntent.getActivity(contexto, 0, intent, 0);
notificacion.setLatestEventInfo(contexto, titulo, descripcion, pendiente);
//se establece que se cancele automáticamente la notificación cuando el usuario hace "click" sobre ella.
notificacion.flags |= Notification.FLAG_AUTO_CANCEL;
//Se establece el icono para la vista apliada
Bitmap bm = BitmapFactory.decodeResource(getResources(), R.drawable.large_icon);
notificacion.largeIcon = bm;
//añade el sonido por default
notificacion.defaults |= Notification.DEFAULT_SOUND;
//establece vibracion con la notificacion
notificacion.defaults |= Notification.DEFAULT_VIBRATE;
//establece parpadeo de LED (si es que el dispositivo cuanta con un LED)
notificacion.defaults |= Notification.DEFAULT_LIGHTS;
nm.notify(0, notificacion);
```

NOTA: En dispositivos con Android Jelly Bean (4.1) o superior se ha dado soporte para activar prioridades en las notificaciones, mediante una bandera que influirá en donde aparecerá la notificación respecto a otras notificaciones para que el usuario vea primero las más importantes. Estas banderas son: MAX, HIGH, DEFAULT, LOW, MIN. Para más información consulte: <http://developer.android.com/intl/es/design/patterns/notifications.html>.

IMPORTANTE: En dispositivos con Android Jelly Bean (4.1) o superior, el parpadeo del LED, solo funcionará si la prioridad de la notificación se establece a MAX o DEFAULT, ejemplo:

[Implementación Nueva:]

```
NotificationCompat.Builder constructorNotificacion = new NotificationCompat.Builder(this)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("Mi notificacion")
    .setContentText("Primera notificacion!")
    //se activan TODAS las opciones: la vibración, el sonido y el parpadeo del LED.
    //Si se desea activar alguna de ellas Reemplazar Notification.DEFAULT_ALL por
    //Notification.DEFAULT_SOUND, Notification.DEFAULT_VIBRATE, Notification.DEFAULT_LIGHTS
    //o combinando dos de ellas con la comparación bit a bit, ejemplo:
    //DEFAULT_SOUND | DEFAULT_VIBRATE
```

```
.setDefaults(Notification.DEFAULT_ALL)
.setLights(0xFF0000FF,100,3000)
.setPriority(Notification.PRIORITY_DEFAULT);
[...]
```

[Implementación Anterior:]

```
notification.setLights(0xFF0000FF,100,3000);
notification.setPriority(Notification.PRIORITY_DEFAULT);
```

10.2. Desplegar notificaciones Toast

Un Toast proporciona información en una pequeña ventana emergente, que solo ocupa el espacio requerido para el mensaje y la actual Activity permanece visible e interactiva. Para implementarlo, se crea instancia de un objeto Toast (clase incluida en el paquete android.widget), llamando a uno de los métodos estáticos `makeText()`, el cual toma 3 parámetros, el contexto (Context) de la aplicación, el texto del mensaje que se quiere mostrar y la duración del toast (`LENGTH_LONG` o `LENGTH_SHORT`), este método devuelve un objeto Toast y para mostrarlo se usa el método `show()`, una vez hecho esto se mostrará el toast en la pantalla la duración especificada y desaparecerá por sí solo:

```
Context contexto = getApplicationContext();
CharSequence texto = "Ejemplo de Toast!";
int duracion = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(contexto, texto, duracion);
toast.show();
```

Si se desea crear un diseño personalizado para un Toast, se puede crear un archivo layout en xml conteniendo el diseño del toast que se quiere mostrar, "inflarlo" y llamar al método `setView(View)` antes de mostrar el Toast con el método `show()`:

```
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.toast_propio, (ViewGroup) findViewById(R.id.raiz_linear_layout));

TextView text = (TextView) layout.findViewById(R.id.text);
text.setText("Esto es un toast personalizado");

Toast toast = new Toast(getApplicationContext());
toast.setDuration(Toast.LENGTH_LONG);
toast.setView(layout);
toast.show();
```

11. Trabajar con Multimedia

11.1. Reproducción de audio

El Framework multimedia de Android incluye soporte multimedia para reproducir diversos tipos de medios comunes, permitiendo integrar fácilmente audio, vídeo e imágenes en la aplicación. Hay 2 clases que son usadas para reproducir audio y video en el framework de Android: `MediaPlayer` y `AudioManager`, respectivamente.

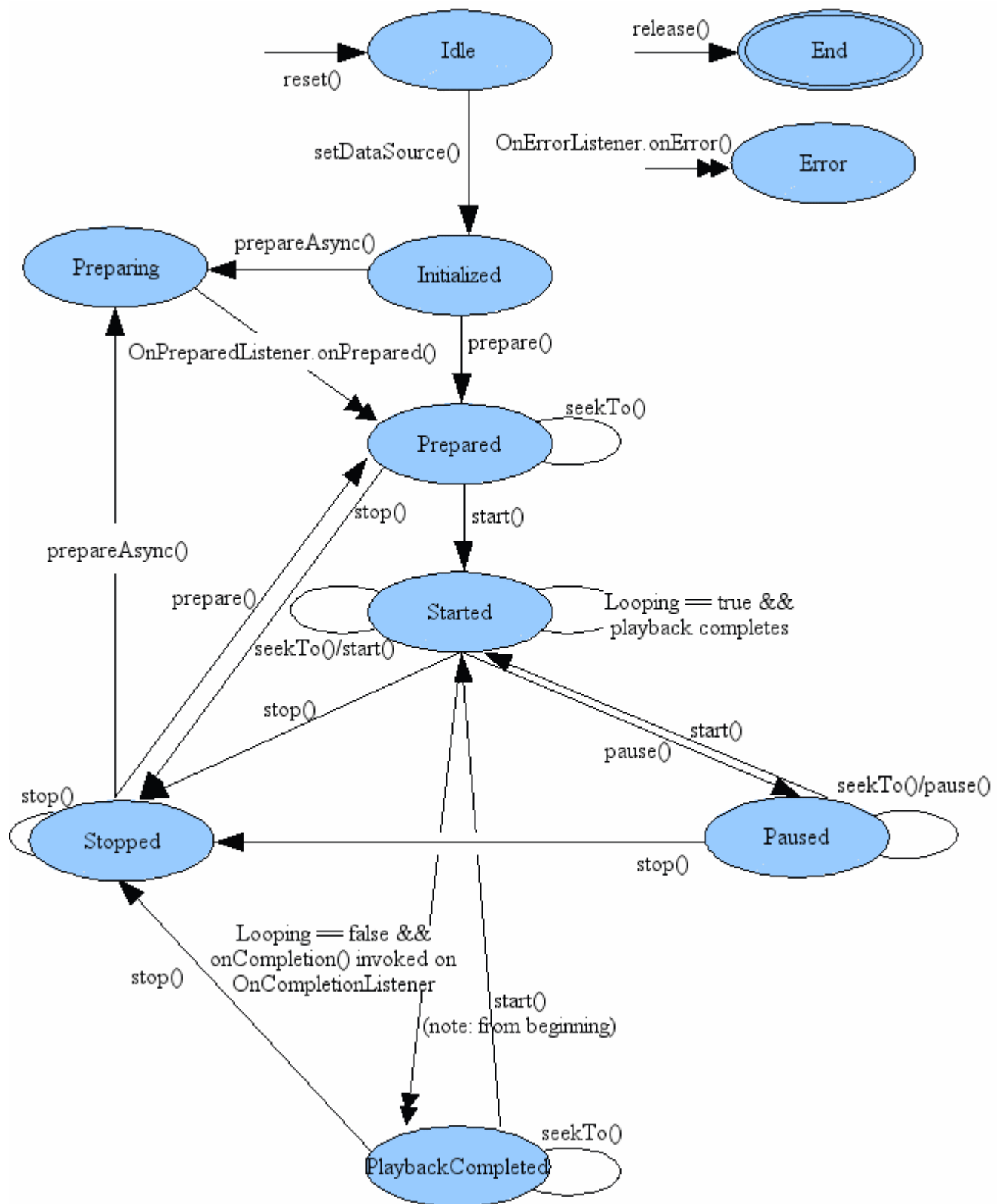
Un aspecto importante de la clase `MediaPlayer` que se debe tener en mente es que el control de la reproducción es manejado como una máquina de estados, es decir, Que el objeto `MediaPlayer` tiene un estado interno que siempre hay que tener en cuenta al escribir el código, ya que algunas operaciones solamente son válidas cuando el reproductor esta en ciertos estados, si se realiza una operación, mientras esta en un estado incorrecto, el sistema puede lanzar una excepción o causar comportamientos indeseables.

La documentación de la clase MediaPlayer muestra un diagrama de estos estados, en el cual se muestra cuáles métodos mueven al MediaPlayer de un estado a otro. Por ejemplo,

- Inicialmente, cuando se crea un nuevo MediaPlayer, está en estado reposo (idle).
- Después, cuando se llama al método `setDataSource()` se pasa al estado inicializado
- Después de eso llamando al método `prepare()` o `prepareAsync()` se realiza la preparación con lo que entra en estado preparado
- en este punto se puede llamar a `start()`, para iniciar la reproducción.

El diagrama completo se puede ver en la siguiente figura, para obtener información detallada de los métodos y los estados de MediaPlayer consulte:

<http://developer.android.com/intl/es/reference/android/media/MediaPlayer.html>:



Un objeto de la clase MediaPlayer puede buscar, decodificar y reproducir audio y video con una mínima preparación. En la implementación para reproducir un recurso video localizable localmente desde un URI (Identificador uniforme de recursos: cadena de caracteres corta que identifica inequívocamente un recurso), solo se necesita lo siguiente:

```
Uri myUri = ....; // inicializar la URI
MediaPlayer mediaPlayer = new MediaPlayer(); //crear una instancia de MediaPlayer
//establece el tipo de flujo de audio para el MediaPlayer, se debe llamar antes de
prepare()
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
//establece la fuente de datos de los medios (URI, ruta del archivo)
mediaPlayer.setDataSource(getApplicationContext(), myUri);
//establece si se quiere reproducir en ciclo (repetir la reproducción al terminar)
mediaPlayer.setLooping(true);
//se obtiene la superficie donde se reproducira el video
SurfaceView superficieView = (SurfaceView)findViewById(R.id.surface);
SurfaceHolder superficie = superficieView.getHolder();
//se establece el objeto SurfaceView en donde se quiere reproducir el video
mediaPlayer.setDisplay(superficie);
//prepara al reproductor para la reproducción, de manera síncrona.
mediaPlayer.prepare();
//inicia o reanuda la reproducción
mediaPlayer.start();
```

Si se desea reproducir solo audio en formato mp3 proporcionando la ruta del archivo, la implementación se reduce:

```
Uri myUri = ....; // inicializar la URI
MediaPlayer mediaPlayer = new MediaPlayer(); //crear una instancia de MediaPlayer
//establece la ruta del archivo)
mediaPlayer.setDataSource("/data/data/sound/archivo.mp3");
//prepara al reproductor para la reproducción, de manera síncrona.
mediaPlayer.prepare();
//inicia o reanuda la reproducción
mediaPlayer.start();
```

Hay una vista llamada VideoView, la cual nos agrega al diseño de la interfaz de una Activity (layout) un reproductor de video, solo tomando como parámetro el URI o la ruta de la fuente. Su implementación es la siguiente:

```
//Se obtiene la referencia a la vista VideoView
VideoView myVideoView = (VideoView)findViewById(R.id.myvideoview);
//Se establece el URI del video
myVideoView.setVideoURI(Uri.parse("UriFuente"));
//se establece a la Activity para controlar el video (pause, play, etc)
myVideoView.setMediaController(new MediaController(this));
//inicia el video
myVideoView.start();
```

Por otro lado, si se quiere reproducir multimedia que esté disponible como un recurso en bruto (raw) local (guardado en el directorio /res/raw/), lo que se debe hacer es adquirir una instancia de la clase MediaPlayer con el método estático create, que recibe, como segundo parámetro, el id del recurso a reproducir, ejemplo:

```
MediaPlayer mediaPlayer = MediaPlayer.create(context, R.raw.sonido);
mediaPlayer.start(); //No es necesario llamar a prepare(); create() ya lo hace.
```

11.2. Reproducción de video

Vease tema 11.1. Reproducción de audio

11.3. Acceso a la cámara

Para acceder desde nuestra aplicación en desarrollo a la cámara de nuestro dispositivo y recoger una foto, se siguen los siguiente pasos:

- Crear un objeto Intent para que lance la cámara de fotos, esto se hace con la siguiente línea de código:

```
Intent          camaraIntent          =          new  
Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
```

- Lanzar una nueva actividad y como se desea que nos devuelva la foto, tenemos que lanzar la nueva actividad esperando que nos devuelva un dato, para ello, hacemos lo siguiente:

```
startActivityForResult(camaraIntent, TOMAR_FOTO);
```

- La variable TOMAR_FOTO será una constante que identifique la actividad de la camara de fotos.

```
private static final int TOMAR_FOTO = 1;
```

- Extraer el resultado de la cámara de fotos, que en este caso será una foto. Para ello tenemos que sobrescribir la clase onActivityResult:

```
@Override  
protected void onActivityResult(int requestCode, int resultCode, Intent data)  
{  
    if (requestCode == TOMAR_FOTO) {  
        Bitmap imagen = (Bitmap) data.getExtras().get("data");  
    }  
}
```

- El resultado que devuelve la cámara es un objeto de tipo Bitmap. Bitmap es una clase de Android que sirve para manejar las imágenes. Por ejemplo, si queremos que se muestre en la interfaz de nuestra aplicación la foto que acabamos de hacer:

```
ImageView imagenFoto = (ImageView) findViewById(R.id.imageView1);  
imagenFoto.setImageBitmap(imagen);
```

12. Preferencias y Almacenamiento de Datos

Android ofrece varias opciones para guardar datos desde las aplicaciones:

- **Shared Preferences:** Guarda datos primitivos en pares llave-valor.
- **Internal Storage:** Guarda datos privados en la memoria del dispositivo.
- **External Storage:** Guarda datos públicos en el almacenamiento externo compartido.
- **SQLite Databases:** Guarda datos estructurados en una base de datos privada.
- **Network Connection:** Guarda datos en la WEB con un servidor de red.

12.1. Uso de Shared Preferences

Las preferencias compartidas (SharedPreferences) no son más que datos que una aplicación debe guardar para personalizar la experiencia del usuario, por ejemplo información personal, opciones de presentación, etc.

La clase SharedPreferences provee un framework que permite guardar y recuperar información en tipos de datos primitivos (boolean, float, int, long y string) en forma de pares llave-valor que persistirán aunque la aplicación sea terminada. Se pueden tener uno o muchos archivos de preferencias, si es solo uno se utiliza el método `getPreferences(int modo)` y si son muchos se utiliza el método `getSharedPreferences(String nombre, int modo)` el cual recibe el nombre del archivo de preferencias (que se guardará en formato XML) y ambos métodos reciben un modo de acceso, 3 de esos modos son:

- `MODE_PRIVATE`. Sólo nuestra aplicación tiene acceso a estas preferencias.
- `MODE_WORLD_READABLE`. Todas las aplicaciones pueden leer estas preferencias, pero sólo la nuestra modificarlas.
- `MODE_WORLD_WRITEABLE`. Todas las aplicaciones pueden leer y modificar estas preferencias.

Para escribir nueva información se debe:

- Llamar al método `edit()` que nos devolverá un objeto `SharedPreferences.Editor`.
- Agregar valores con métodos como `putBoolean(String llave, boolean valor)` o `putString(String llave, String valor)` entre otros.
- Confirmar los valores con el método `commit()`.

Las preferencias compartidas no son estrictamente dedicadas para guardar preferencias del usuario, como por ejemplo el tono que eligió, es posible, también, crear preferencias para la aplicación extendiendo de la clase `PreferenceActivity` que provee un framework para crear preferencias personalizadas que se guardarán de forma automática en un archivo xml usando preferencias compartidas.

Implementación:

- Archivo xml de preferencias, guardado en `res/xml`:

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

    <CheckBoxPreference android:key="primera"
        android:title="Primera opcion"
        android:summary="Es la primera opcion" />

    <CheckBoxPreference android:key="segunda"
        android:title="Segunda opcion"
        android:summary="Es la segunda opcion" />

</PreferenceScreen>
```

- Clase PreferenceActivity :

```

public class Preferencias extends PreferenceActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.preferencias);
    }
}

private EditText et;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_principal);
    et = (EditText) findViewById(R.id.editText1);
    //=====Recuperacion de Preferencias
    =====
    SharedPreferences sp = getSharedPreferences("misprefer", 0);
    et.setText(sp.getString("texto", ""));

    SharedPreferences shapre =
PreferenceManager.getDefaultSharedPreferences(this);
    TextView tv = (TextView) findViewById(R.id.textView2);
    TextView tv2 = (TextView) findViewById(R.id.textView3);
    if (shapre.getBoolean("primera", false) == true){
        tv.setText("Primera opcion seleccionada");
    }
    else {
        tv.setText("Primera opcion NO seleccionada");
    }
    if (shapre.getBoolean("segunda", false) == true){
        tv2.setText("Segunda opcion seleccionada");
    }
    else {
        tv2.setText("Segunda opcion NO seleccionada");
    }

    Button b = (Button) findViewById(R.id.button1);
    b.setOnClickListener(new OnClickListener() {

        @Override
        public void onClick(View v) {
            //=====Lanzar Activity de Preferencias
            Intent i = new Intent(Principal.this, Preferencias.class);
            startActivity(i);
        }
    });
}

```

```
    }

    //=====Almacenamiento de Preferencias
    =====
    @Override
    protected void onStop() {
        // TODO Auto-generated method stub
        super.onStop();
        SharedPreferences sp = getSharedPreferences("misprefer", 0);
        SharedPreferences.Editor spe = sp.edit();
        spe.putString("texto", et.getText().toString());
        spe.commit();
    }
```

12.2. Crear Activity de Preferences

Vease tema 12.1. Uso de Shared Preferences

12.3. Sistema de Archivos Interno y Externo

En el caso del **almacenamiento interno (Internal Storage)**, por default los archivos guardados aquí son privados, solo para la aplicación que los creó, y otras aplicaciones no pueden accederlos. Se guardan en la memoria interna del dispositivo. Cuando se desinstala la aplicación, estos archivos son eliminados.

Para crear y escribir un archivo privado en el almacenamiento interno:

1. Llamar al método `openFileOutput()` con el nombre del archivo y el modo de operación. Este método regresa un objeto `FileOutputStream`.
2. Escribir a el archivo con el método `write()`, si se desea leer se usa el método `read()`.
3. Cerrar el flujo hacia el archivo (stream) con el metodo `close()`.

Ejemplo:

```
String NOMBREARCHIVO = "ejemplo.txt";
String cadena = "Hola mundo!";

FileOutputStream fos = openFileOutput(NOMBREARCHIVO, Context.MODE_PRIVATE);
fos.write(cadena.getBytes());
fos.close();
```

El argumento `MODE_PRIVATE` (con valor constante de 0) creará o reemplazará el archivo en la ruta `/data/data/<nombre_paquete>/files` y lo hará privado a la aplicación.

En el caso del **almacenamiento externo (External Storage)**, los archivos guardados aquí no tienen permisos de acceso de Linux, por lo que son compartidos por todas las aplicaciones, se pueden guardar grandes cantidades de datos. Se guardan, la mayoría de las veces, en la tarjeta SD (SD Card), que normalmente tiene un sistema de archivos FAT, aunque Android también soporta los formatos EXT2, EXT3, EXT4.

El manejo de los archivos en Java en la tarjeta SD se hace con la clase paquete `java.io.File` y debido a que la tarjeta SD puede ser removida, o estar dañada o sin acceso por la razón que sea, se hace necesario, antes de realizar alguna operación con archivos en la tarjeta SD, verificar si es accesible el directorio de almacenamiento externo, la mayoría de las veces refiriéndose a la tarjeta SD. Esto se hace llamando al método estático `Environment.getExternalStorageState()`.

Otro punto importante que se debe tener en cuenta, al desarrollar una aplicación que realice operaciones con archivos en el almacenamiento externo, es que se debe solicitar el permiso `WRITE_EXTERNAL_STORAGE`, en caso de escritura, o `READ_EXTERNAL_STORAGE`, en caso de lectura, en el archivo `AndroidManifest.xml`

Más información sobre External Storage:

<http://developer.android.com/intl/es/guide/topics/data/data-storage.html#filesExternal>

[http://developer.android.com/intl/es/reference/android/os/Environment.html#getExternalStorageDirectory\(\)](http://developer.android.com/intl/es/reference/android/os/Environment.html#getExternalStorageDirectory())

12.4. Uso de Base de Datos en SQLite

Android tiene soporte completo para las bases de datos SQLite, la cual soporta transacciones con propiedades atómicas, consistentes, aisladas y durables (ACID), también soporta creación de vistas y disparadores (triggers). Cuando se crea una base de datos de SQLite, se guardará en un archivo, que incluye todas las tablas que pertenezcan a esa base de datos. Cualquier base de datos en SQLite será accesible por nombre por cualquier clase en la aplicación, pero no fuera de la aplicación, la base de datos es únicamente local.

Cada valor almacenado en una base de datos SQLite (y manipulado por el motor de la base de datos) tiene una de las siguientes tipos de almacenamiento:

- **NULL:** Valor nulo
- **INTEGER:** Numero entero con signo.
- **REAL:** Valor de coma flotante.
- **TEXT:** Cadena de texto.
- **BLOB:** Almacenamiento en masa de datos, guardado exactamente como se recibió.

Las transacciones en SQLite por default son diferidas, lo que significa que no se bloquea la base de datos hasta que se accede ya sea por lectura o escritura por primera vez a la base de datos.

Más información:

<http://www.sqlite.org/>

<http://www.sqlite.org/datatype3.html>

http://www.sqlite.org/lang_transaction.html

En el caso de las bases de datos SQLite, que guarda los datos en una base datos local para la aplicación. La implementación se recomienda:

- Hacer una subclase (herencia) de la clase `android.database.sqlite.SQLiteOpenHelper` (el constructor de la subclase llamará al constructor de la clase padre (`SQLiteOpenHelper`) creando la base de datos) y sobrescribir los métodos `onCreate()`, en el cual se ejecutarán los comandos SQL para crear tablas en la base de datos y `onUpgrade()`, en el cual se harán las operaciones necesarias cuando se actualice la base de datos completa (como borrar, modificar o eliminar tablas).
- Para escribir y leer desde la base de datos se llamará a los métodos `getWritableDatabase()` y `getReadableDatabase()`, los cuales regresan un objeto `SQLiteDatabase` que representa la base de datos y provee métodos para las operaciones de SQLite.
- Para ejecutar consultas se usa el método `query` de la clase `SQLiteDatabase`, que acepta varios parámetros de consulta (`select`, `group`, etc).

- Para navegar en los resultados de las consultas (leer filas y columnas), se usa el objeto Cursor, el cual contiene mecanismos para eso, todas las consultas devuelven un objeto Cursor que apunta a todos los resultados devueltos por la consulta.

Implementación básica:

```
SQLiteDatabase bd = openOrCreateDatabase("MiBD", MODE_PRIVATE, null);
//=====Almacenamiento
bd.execSQL("CREATE TABLE IF NOT EXISTS mitabla (nombre VARCHAR, apellido VARCHAR, edad INT(3));");
bd.execSQL("INSERT INTO mitabla VALUES ('Pedro', 'Perez', 42)");
bd.close();

//=====Recuperación
Cursor c = bd.rawQuery("SELECT * FROM mitabla;", null);
c.moveToFirst();

TextView tv = (TextView) findViewById(R.id.textView2);
tv.setText(c.getString(c.getColumnIndex("nombre"))+" - "+ c.getString(c.getColumnIndex("apellido"))+
        " - "+c.getString(c.getColumnIndex("edad")));
```

Más Información:

<http://developer.android.com/guide/topics/data/data-storage.html>

12.5. Acceso a la Red

Android permite implementar de manera muy simple la conexión a la red, La mayoría de las aplicaciones que requiere conexión a la red en Android usan el protocolo HTTP para mandar y recibir datos. Android incluye 3 clientes HTTP:

- java.net.HttpURLConnection
- org.apache.http.client.HttpClient
- android.net.http.AndroidHttpClient (implementación de org.apache.http.impl.client.DefaultHttpClient, tiene configurados con los ajustes por default para android)

Los 3 soportan, entre otras cosas: HTTPS, descargas y subidas en transmisión (streaming), tiempos de espera configurables, IPV6, etc.

Implementación:

- Agregar el permiso de Internet al archivo AndroidManifest.XML

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

- En código:

```
final EditText et = (EditText) findViewById(R.id.editText1);
Button b = (Button) findViewById(R.id.button1);
final TextView tv = (TextView) findViewById(R.id.textView1);

b.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View arg0) {
        // TODO Auto-generated method stub
        try{
            URL url =null;
```

```
        url = new URL(et.getText().toString());
        URLConnection con = url.openConnection();
        BufferedReader lector = new BufferedReader(new InputStreamReader(
                                                    con.getInputStream()));

        String line = "";
        while ((line = lector.readLine()) != null){
            tv.append(line);
        }
    }catch (Exception e){
    }

    });
}
```

12.6. Uso de Content Providers

Vease tema 2.3. Componentes de una aplicación -> Content Providers

13. Geo localización y Mapas

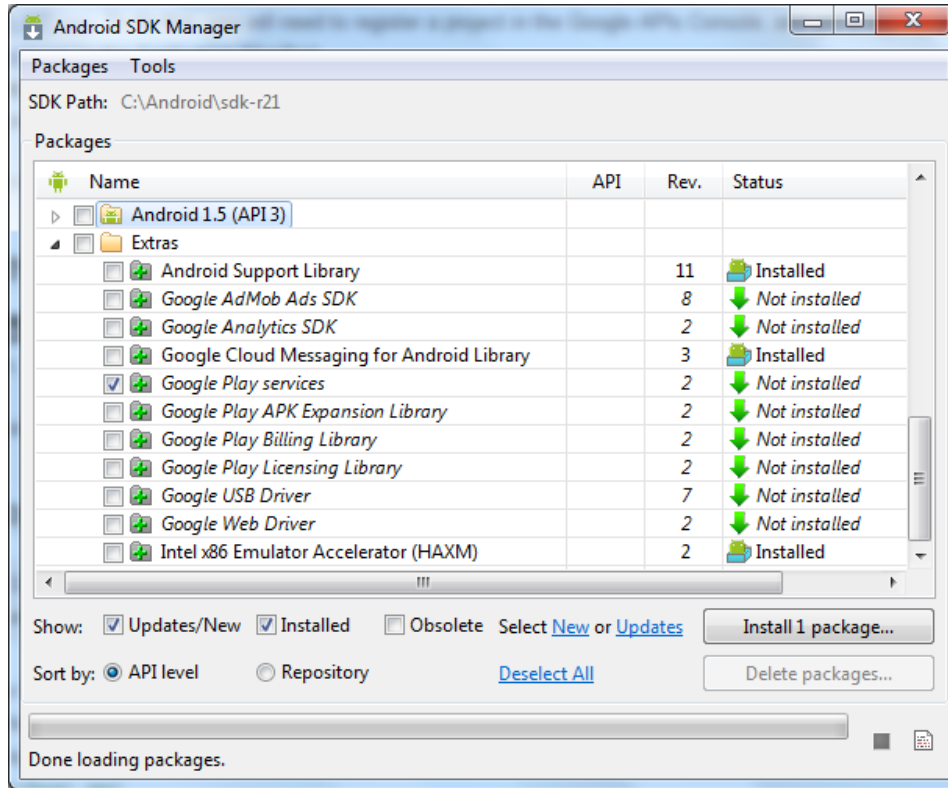
13.1. Incorporación de Google Maps Android API v2

Con la API de Android para Google Maps, se puede agregar mapas basados en Google Maps a la aplicación. La API automatiza el acceso a los servidores de Google Maps, la descarga de los datos, la visualización del mapa y la respuesta a los gestos en la pantalla. También permite agregar polígonos, marcadores y capas superpuestas (overlays) a un mapa.

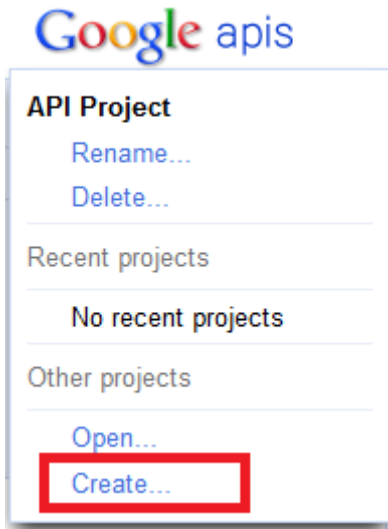
Para trabajar con la API de Android para Google Maps, se necesita seguir el siguiente proceso:

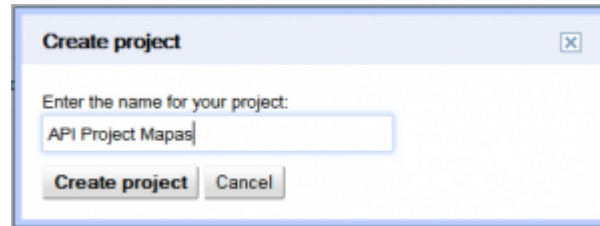
- Descargar (gratis) y configurar el SDK de Google Play Services ya que la API de Android para Google Maps es parte de ese SDK. La API v2 se proporciona como parte del SDK de Google Play Services, por lo tanto será necesario incorporar previamente a nuestro entorno de desarrollo dicho paquete. Esto se hace accediendo desde Eclipse al Android SDK Manager y descargando del apartado de extras el paquete llamado "Google Play Services". El paquete quedará instalado en nuestro sistema, en la ruta:

<carpeta-sdk-android>/extras/google/google_play_services/ :



- Obtener (gratis) una llave única de la API para Android de Google Maps V2. Esta confirma que se ha registrado con el servicio de Google Maps a través de la consola de la API de Google. Se requiere una llave por aplicación, no importando el número de usuarios de la aplicación. La llave se obtiene en la consola de la API de Google (Google APIs console) mediante:
 - Crear un proyecto





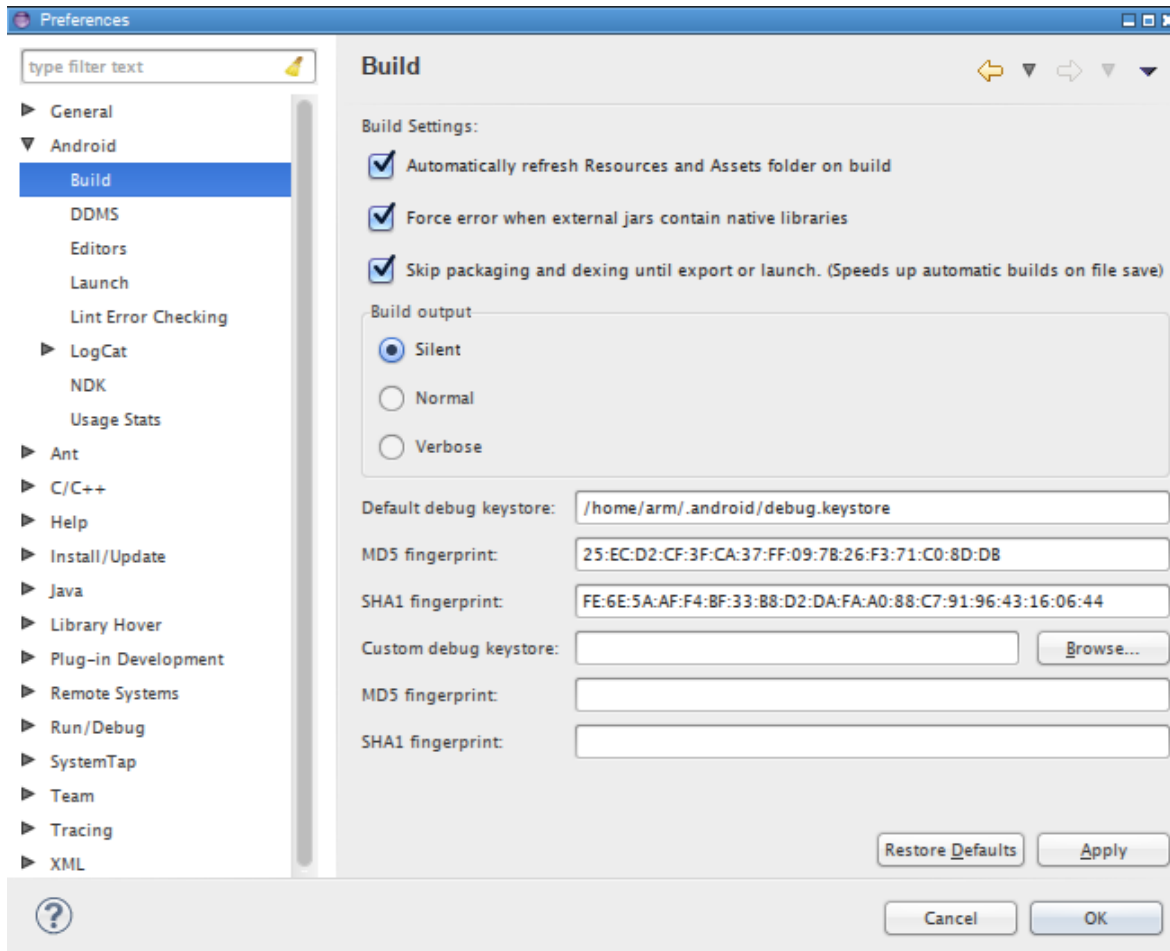
- Activar el servicio Google Maps Android API v2

	Google Compute Engine		Request access...
	Google Contacts CardDAV API		<input type="checkbox"/> OFF
	Google Maps Android API v2		<input checked="" type="checkbox"/> ON
	Google Maps API v2		<input type="checkbox"/> OFF
	Google Maps API v3		<input type="checkbox"/> OFF

- Crear la llave en la sección API Access en la cual se solicita una cadena de texto formada por la huella digital SHA-1 (SHA-1 fingerprint) del certificado ya sea de depuración (debug certificate) o de liberación (release certificate), seguido de un carácter ";" y el nombre del paquete de la aplicación. Por ejemplo:

"BB:0D:AC:74:D3:21:E1:43:67:71:9B:62:91:AF:A1:66:6E:44:5D:75;com.ejemplo..ejemplomapas"

Para obtener la huella digital SHA-1 (SHA-1 fingerprint) del certificado de depuración, esta se obtiene accediendo desde Eclipse al menú Window /Preferences y accediendo a la sección Android / Build.



Esa cadena (SHA-1 fingerprint) mas el caracter“;” y seguido del nombre del paquete se proporcionará en el diálogo que muestra la página de Google apis al dar click en “Create New Android Key...”

Google apis

API Project New

Overview

Services

Team

API Access

Reports

Quotas

API Access

To prevent abuse, Google places limits on API requests. Using a valid OAuth token or API key allows you to exceed anonymous limits by connecting requests back to your project.

Authorized API Access

OAuth 2.0 allows users to share specific data with you (for example, contact lists) while keeping their usernames, passwords, and other information private. A single project may contain up to 20 client IDs.

[Learn more](#)

Create an OAuth 2.0 client ID...

Simple API Access

Use API keys to identify your project when you do not need to access user data. [Learn more](#)

Key for browser apps (with referers)

API key: AIZA5yAwuYjLkzKEZOWObTzur9uNfplLN5ZLBcI

Referers: Any referer allowed

Activated on: Jun 12, 2013 4:18 PM

Activated by: arm.acb@gmail.com – you

[Generate new key...](#)

[Edit allowed referers...](#)

[Delete key...](#)

Create new Server key... Create new Browser key... Create new Android key... Create new iOS key...

Configure Android Key for API Project New

This key can be deployed in your Android applications.

API requests are sent directly to Google from your clients' Android devices. Google verifies that each request originates from an Android application that matches one of the certificate SHA1 fingerprints and package names listed below. You can discover the SHA1 fingerprint of your developer certificate using the following command:

```
keytool -list -v -keystore mystore.keystore
```

[Learn more](#)

Accept requests from an Android application with one of the certificate fingerprints and package names listed below:

```
BB:0D:AC:74:D3:21:E1:43:67:71:9B:62:91:AF:A1:66:6E:44:5D:75;com.ejemplo.  
.ejemplomapas
```

One SHA1 certificate fingerprint and package name (separated by a semicolon) per line. Example:

```
45:B5:E4:6F:36:AD:0A:98:94:B4:02:66:2B:12:17:F2:56:26:A0:E0;com.example
```

Create Cancel

Lo cual generará la llave única que será agregada como elemento hijo del elemento <application>, ejemplo:

```
<application>
  <meta-data
```

```
        android:name="com.google.android.maps.v2.API_KEY"
        android:value="llave_unica"/>
    <!-- ... -->
</application>
```

También, se deberá especificar el nivel de protección que tendrá que verificar el sistema para ejecutar la aplicación, en este caso es "signature", el cual se aprobará si la aplicación es firmada con el mismo certificado que se declara en el permiso. Ejemplo:

```
    <permission
        android:name="com.ejemplo.android.ejemplomapas.permission.MAPS_RECEIVE"
        android:protectionLevel="signature"/>
    <uses-permission
        android:name="com.ejemplo.android.ejemplomapas.permission.MAPS_RECEIVE"/>
```

NOTA: Para más información sobre niveles de protección, consulte:

<http://developer.android.com/intl/es/guide/topics/manifest/permission-element.html>

- Especificar algunos ajustes en el archivo AndroidManifest.xml:
 - Permisos que dan acceso a la aplicación a las características del sistema Android y a los servidores de Google Maps:

```
    <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"/>
    <!-- Los siguientes no son necesarios, pero son recomendados para la ubicación del usuario
    -->
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```
 - Notificación de que la aplicación requiere de OpenGL ES version 2.

```
    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true"/>
```
- Agregar un mapa a un archivo de diseño o layout en un proyecto de Android, agregando un fragmento con nombre "com.google.android.gms.maps.SupportMapFragment". Ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

- Extender la clase principal de android.support.v4.app.FragmentActivity:

```
public class MainActivity extends android.support.v4.app.FragmentActivity {

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

}
```

- Si se desea, publicar la aplicación, creando una “Android Key” con el certificado de liberación.

Más información:

<https://developers.google.com/maps/documentation/android/>

13.2. Uso de GPS para ubicar la posición actual

La implementación para obtener la ubicación geográfica por GPS de un dispositivo Android consta de los siguientes pasos:

```
public class MainActivity extends android.support.v4.app.FragmentActivity {

    // Se declara una variable de tipo GoogleMap para mostrar la ubicación geográfica actual.
    private GoogleMap mapa = null;

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

        //Se obtiene la referencia al fragmento Mapa, mediante el metodo getSupportFragmentManager
        //para compatibilidad con sistemas Android anteriores.
        mapa = ((SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.map)).getMap();

        //Obtenemos una referencia al LocationManager, la clase que provee acceso a los
        //servicios de localizacion del sistema.
        LocationManager locManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);

        //Obtenemos la ultima posicion conocida por el GPS_PROVIDER
        Location loc = locManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);

        //Mostramos, inicialmente, la ultima posicion conocida
        mostrarLocalizacion(loc.getLatitude(), loc.getLongitude());

        //Creamos un nuevo objeto de tipo listener un LocationListener, que respondera
        //a los eventos relacionados con el GPS, incluyendo el cambio de ubicacion
        LocationListener locListener = new LocationListener() {
            //Metodo callback que se llama automaticamente cuando el GPS registró un cambio de ubicación
            public void onLocationChanged(Location location) {
                // se crea un objeto de la clase CameraUpdate que se encarga de mover
                //la camara o el punto de vista del mapa de acuerdo con los parámetros que especifiquemos,
                //en este caso: recibe un objeto de tipo LatLng a partir de la latitud y longitud del
                // parametro location, el objeto LatLng es creado por el metodo estatico newLatLng de la
                // clase CameraUpdateFactory.
                CameraUpdate center=CameraUpdateFactory.newLatLng(new LatLng(location.getLatitude(),
                    location.getLongitude()));
                // se le indica al mapa que se mueva la camara con los datos del objeto CameraUpdate.
                mapa.moveCamera(center);
                // se llama a un metodo que muestra un marcador en la nueva ubicación
                mostrarLocalizacion(location.getLatitude(),location.getLongitude());
            }
            public void onProviderDisabled(String provider){
            }
            public void onProviderEnabled(String provider){
            }
            public void onStatusChanged(String provider, int status, Bundle extras){
            }
        };

        // Una vez que se creo el LocationListener, activaremos el proveedor de localización y
        //suscribiremos la aplicacion a sus eventos, lo cual se realiza mediante una
```

```
//llamada al método requestLocationUpdates(), al que deberemos pasar 4 parámetros distintos:

// -> Nombre del proveedor de localización al que nos queremos suscribir. (En este caso el GPS)
// -> Tiempo mínimo entre actualizaciones, en milisegundos. (2000 milisegundos)
// -> Distancia mínima entre actualizaciones, en metros. (0)
// -> Instancia de un objeto LocationListener, el que definimos anteriormente (locListener).

locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 2000, 0, locListener);
}

//Metodo que agrega un marcador en el mapa en la latitud y longitus recibidas...
private void mostrarLocalizacion(double lat, double lng){
    // se mandan mensajes de depuracion con las latitudes y longitudes obtenidas
    //por el objeto LocationListener...
    Log.i("Loc", "Nueva Ubicacion :" + lat + " - "+lng);
    // se Agrega un marcador a mapa en las latitudes y longitudes obtenidas
    //por el objeto LocationListener...
    mapa.addMarker(new MarkerOptions()
        .position(new LatLng(lat, lng))
        .title("Nueva ubicacion..."));
}
}
```

Un punto importante para que la aplicación puede hacer uso del GPS, es solicitar el permiso en el archivo AndroidManifest.xml:

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

14. Creación de Widgets

14.1. Widget simple home-screen

Para crear un widget simple en la pantalla de inicio (home-screen) con una Activity de configuración, se debe hacer lo siguiente:

- Crear un archivo XML layout que contendrá el diseño del widget, en este caso solo una imagen. Al realizar este diseño, se debe considerar que solo podemos usar subconjunto de todos los controles de android, lo cuales se mencionan en la página:

<http://developer.android.com/guide/topics/appwidgets/index.html#CreatingLayout>

Para este ejemplo, usaremos un ImageButton, el cual si se puede usar.

```
[Archivo widget_principal.xml]
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
```

```
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".Principal" >
```

```
<ImageButton
    android:id="@+id/imageButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:src="@drawable/ic_launcher" />
```

```
</RelativeLayout>
```

- También crear un Archivo XML Layout que contendrá el diseño para la Activity de configuración del widget:

[Archivo `activity_configuracion.xml`]

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".Configuracion" >
```

```
<EditText
    android:id="@+id/editText1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:ems="10"
    android:text="http://lolcats.com" >
```

```
<requestFocus />
</EditText>
```

```
<Button
    android:id="@+id/button1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignParentLeft="true"
    android:text="Crear Widget" />
```

```
</RelativeLayout>
```

- Crear un archivo de recurso en el directorio `res/xml`, que describe al widget y sus propiedades, este archivo contiene en sus atributos, los siguientes:
 - **`android:initialLayout=""`** : la referencia al archivo XML layout que se mostrará como widget.
 - **`android:updatePeriodMillis=""`** : rango en milisegundos para actualizar el widget, se recomienda que sea mayor a 30 minutos.

- **android:configure=""** : nombre completo (con paquete) de la clase que implementa la Activity para la configuración del widget.
- **android:minWidth=""** y **android:minHeight=""** : atributos en los cuales se establece el tamaño en ancho y alto del widget, los cuales se obtendrán de acuerdo a la fórmula:

$$70 \times \text{numero_celdas_en_pantalla} - 30 = \#dp, \text{ de acuerdo a la siguiente tabla:}$$

# of Cells (Columns or Rows)	Available Size (dp) (minWidth or minHeight)
1	40dp
2	110dp
3	180dp
4	250dp
...	...
<i>n</i>	$70 \times n - 30$

Más información:

http://developer.android.com/guide/practices/ui_guidelines/widget_design.html#anatomy_determining_size

El archivo, para este ejemplo, quedará así:

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:updatePeriodMillis="0"
    android:initialLayout="@layout/widget_principal"
    android:minWidth="144dp"
    android:minHeight="72dp"
    android:configure="com.ejemplo.widgetapp.Configuracion"
</appwidget-provider>
```

- En el archivo AndroidManifest.xml y en el elemento <application> borramos su contenido y creamos un <receiver>, que básicamente se declaran 3 cosas:
 - La clase java del widget que extenderá de android.appwidget.AppWidgetProvider y **mínimamente**, sobrescribirá el método onUpdate(), el cual actualizará el widget de acuerdo al tiempo en milisegundos.
 - El intent-filter, donde se indicara las acción a las cual puede responder este widget en este caso "android.appwidget.action.APPWIDGET_UPDATE" que se refiere a una acción de actualización de widget.
 - El elemento <meta-data> donde, entre otras cosas se establece el nombre de recurso, que describe a el widget y sus propiedades, en este caso el archivo creado anteriormente en el directorio res/xml.

Esa parte del archivo, para este ejemplo, queda de la siguiente forma:

```
<receiver android:name="com.ejemplo.widgetapp.Principal" >
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>

    <meta-data
        android:name="android.appwidget.provider"
        android:resource="@xml/recurso" />
</receiver>
```

- En adición a lo anterior, también declararemos la Activity de Configuración, nuevamente, con el intent-filter, se indicarán la acción a la cual puede responder este widget, en este caso "android.appwidget.action.APPWIDGET_CONFIGURE" que se refiere a una acción de configuración del widget.

```
<activity
    android:name="com.ejemplo.widgetapp.Configuracion"
    android:label="@string/title_activity_configuracion" >
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_CONFIGURE"/>
    </intent-filter>
</activity>
```

- Un aspecto extra que definiremos en este Archivo, sera el permiso para Internet, sólo para propósitos del funcionamiento de este ejemplo:

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

- La implementación de la clase java del widget que extiende de AppWidgetProvider, para este ejemplo, sólo abrirá un navegador con una página web (<http://google.com>), mediante un PendingIntent :

```
public class Principal extends AppWidgetProvider {

    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
        int[] appWidgetIds) {
        // TODO Auto-generated method stub
        super.onUpdate(context, appWidgetManager, appWidgetIds);

        // se recorren todos los posibles ids de widgets del mismo tipo que
        // hayan sido agregados a la pantalla de inicio.
        for (int i=0; i<appWidgetIds.length; i++){
            int appWidgetId = appWidgetIds[i];

            // se crea el PendingIntent que abra la pagina en un navegador
            Intent intent= new Intent(Intent.ACTION_VIEW,
                Uri.parse("http://google.com"));
            PendingIntent pi = PendingIntent.getActivity(context, 0, intent, 0);

            // se obtiene la referencia a las views remotas.
            RemoteViews rv = new RemoteViews(context.getPackageName(),
                R.layout.activity_principal);
            // se asocia el evento de click en el ImageButton en las views remotas,
            // para lanzar el PendingIntent.
            rv.setOnClickPendingIntent(R.id.imageButton1, pi);
            // se actualiza el o los widget con la nueva información anterior
            appWidgetManager.updateAppWidget(appWidgetId, rv);
        }
    }
}
```

- Y para la clase que implementa la Activity para configurar el widget:

```
public class Configuracion extends Activity {

    //variable para el contexto actual (la Activity actual)
    private Configuracion context;
    //variable para guardar el id del widget que se configura.
    private int widgetID;
```



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

    //en caso de que se presione el boton de regresar sin configurar nada
    setResult(RESULT_CANCELED);
    //se obtiene la referencia al contexto actual.
    context =this;

    // se obtiene el id del widget a configurar,
    // que se encuentra empaquetado en los datos extras que
    // automáticamente se agregan al Intent que lanza la Activity
    // de configuración
    Bundle extras = getIntent().getExtras();
    if (extras != null ){
        widgetID = extras.getInt(AppWidgetManager.EXTRA_APPWIDGET_ID,
                                AppWidgetManager.INVALID_APPWIDGET_ID);
    }
    // se obtiene la referencia al widget para, posterior y manualmente,
    // actualizarlo.
    final AppWidgetManager widgetManager =
        AppWidgetManager.getInstance(context);

    // se obtiene la referencia a las views remotas.
    final RemoteViews rv = new RemoteViews(context.getPackageName(),
        R.layout.activity_principal);

    // se obtiene referencia a las Views locales (de esta Activity)
    // de configuracion.
    final EditText et = (EditText) findViewById(R.id.editText1);
    Button b = (Button) findViewById(R.id.button1);

    // se establece la accion al hacer click en el boton en la
    // Activity de configuración.
    b.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {

            //nuevamente, se crea el PendingIntent que abra la pagina en el
            // navegador, pero esta vez la direccion de la pagina se obtiene del
            // EditText de la Activity actual, permitiendo al usuario establecer la
            // dirección de la pagina que decida.
            Intent intent= new Intent(Intent.ACTION_VIEW,
                                    Uri.parse(et.getText().toString()));
            PendingIntent pi = PendingIntent.getActivity(context, 0, intent, 0);
            // se asocia el evento de click en el ImageButton en las views remotas,
            // para lanzar el PendingIntent.
            rv.setOnClickPendingIntent(R.id.imageButton1, pi);
            // se actualiza el widget con la nueva información anterior
            widgetManager.updateAppWidget(widgetID, rv);

            //estableceremos el Intent de respuesta que le dirá al widget que
            // la configuración se hizo correctamente (RESULT_OK)
            Intent valorResultado = new Intent();
            valorResultado.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, widgetID);
            setResult(RESULT_OK, valorResultado);
            //una vez que hemos configurado el widget, cerramos la Activity
            finish ();
        }
    });
}
```

14.2. Activity de Configuración de Widget

Vease tema 14.1. Widget simple home-screen

15. Publicación de Aplicación

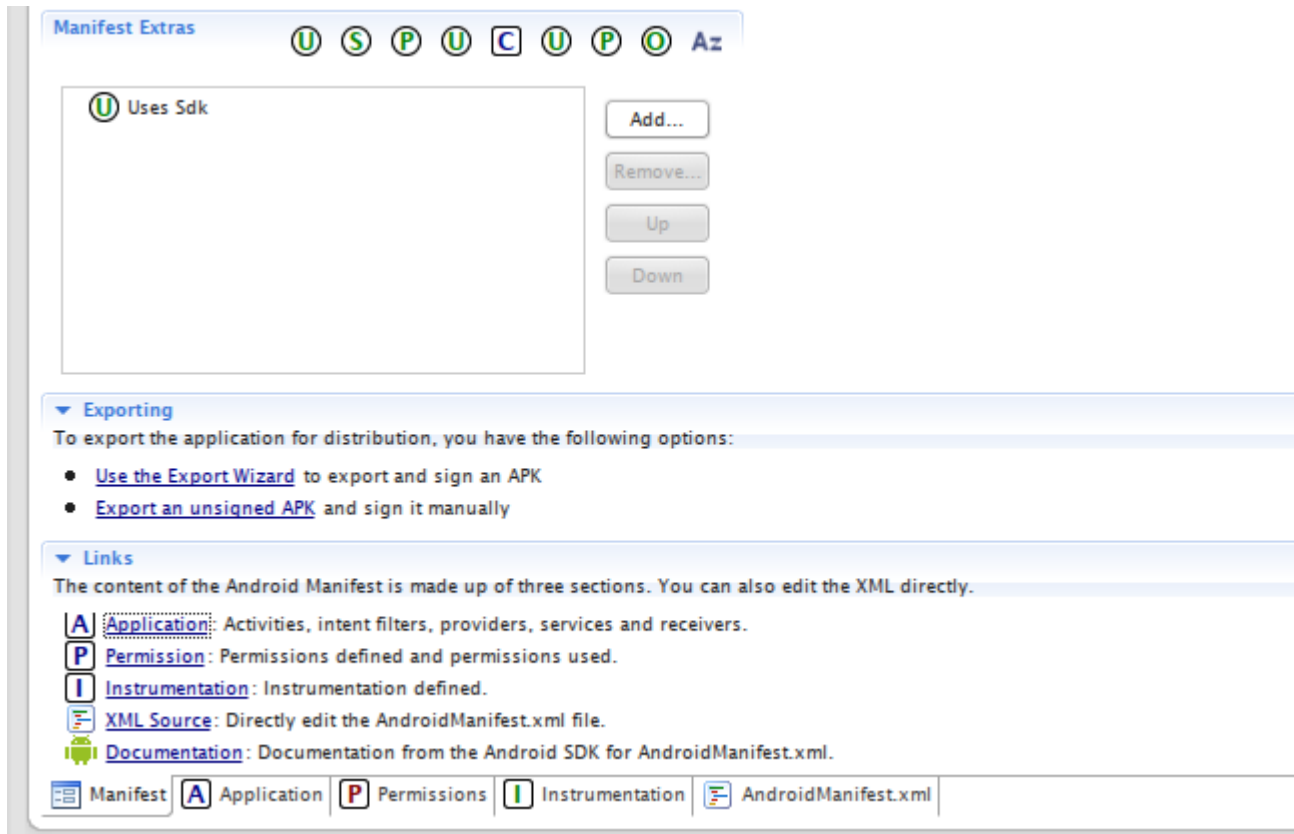
15.1. Preparación para publicación

La preparación de una aplicación para su publicación consiste en:

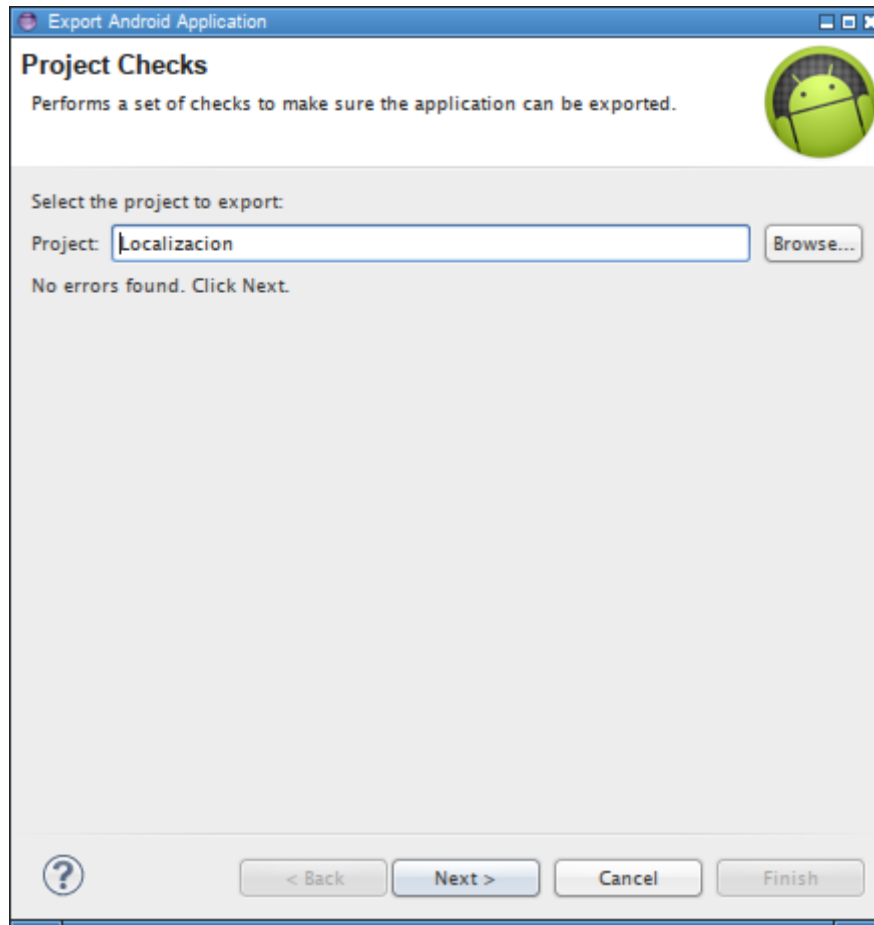
- Remover cualquier llamada a alguno de los métodos de la clase Log.
- Establecer la propiedad Debuggable a false, en el archivo AndroidManifest.xml
- Establecer los valores correctos de android:versionCode="" y android:versionName="".
- Establecer el valor apropiado de android:minSdkVersion=""
- Proveer a la aplicación de un icono identificable.
- Asegurarse que funciona correctamente.

15.2. Firma y Construcción Lección

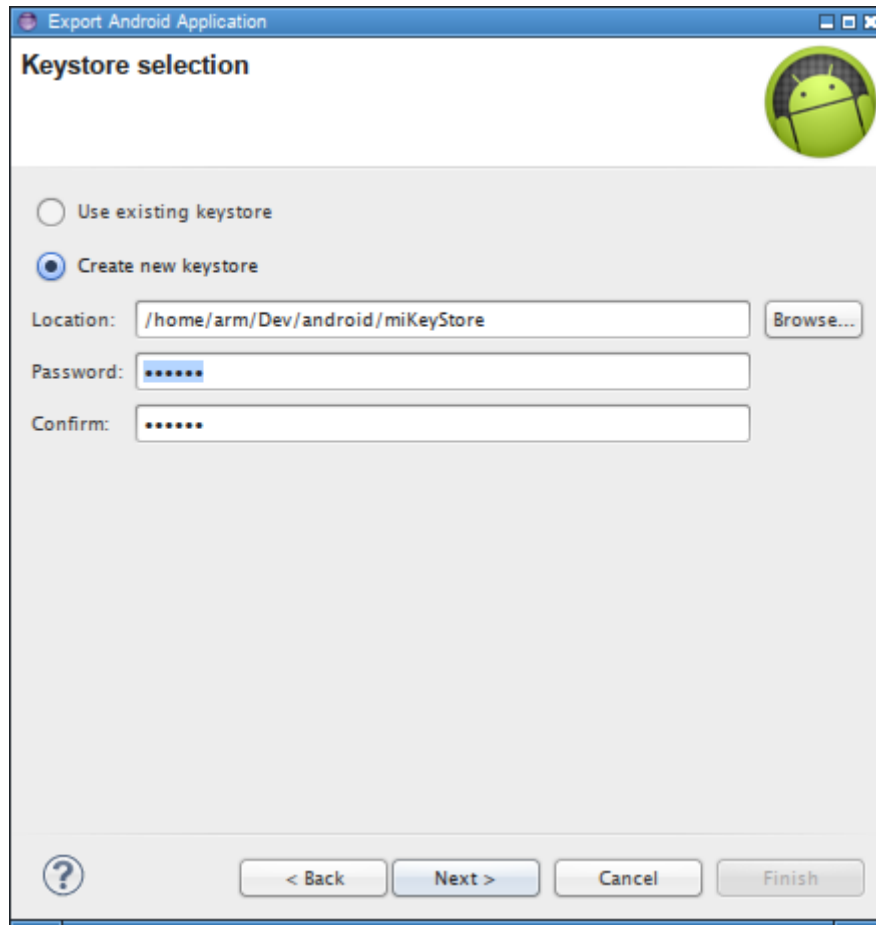
Lo siguiente será abrir el archivo AndroidManifest.xml y en la primera pestaña inferior "Manifest", apartado Exporting dar click en "Use the Export Wizard":



Lo que nos abrirá una ventana que presentará los errores en el aplicación para corregirlos, si es que hay, damos click en next:



Se nos presentará una ventana para crear un nuevo keystore o seleccionar uno ya lo hemos creado anteriormente, en caso de crear uno nuevo, debemos **guardarlo y respaldarlo muy bien**:



Si seleccionamos “Create new keystore”, la siguiente ventana nos pedirá algunos otros datos, despues damos click en next:



Export Android Application

Key Creation

Alias:

Password:

Confirm:

Validity (years):

First and Last Name:

Organizational Unit:

Organization:

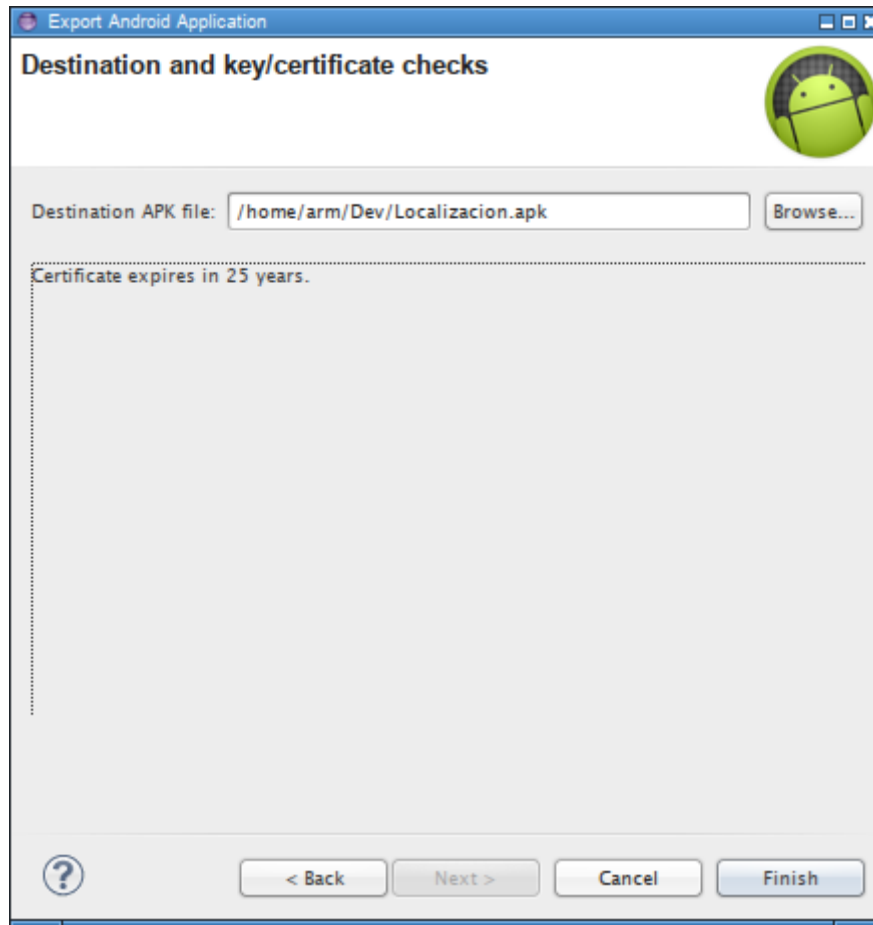
City or Locality:

State or Province:

Country Code (XX):



Lo que nos llevará a la ventana final, donde estableceremos donde queremos que se guarde el archivo .apk y damos click en finish lo que construirá, compilara y firmará la aplicación final:



15.3. Preparación de los recursos gráficos

Vease tema 15.4. Publicación a Google Play (ANTES Android Market !!!)

15.4. Publicación a Google Play (ANTES Android Market !!!)

Para publicar, una vez que ya se tiene el archivo .apk final de nuestra aplicación:

- El primer paso es registrarse un cuenta (publisher account) en el sitio de Google Play Developer Console:
<https://play.google.com/apps/publish/>.
- Proporcionar información básica para identificarse como desarrollador, como nombre, e-mail, etc.
- Leer y aceptar el acuerdo de distribución para desarrolladores que aplique en la región local, además de que las aplicaciones también deben cumplir con las políticas del programa de desarrolladores y el derecho de exportación de US (Developer Program Policies and US export law).
- Realizar un pago único de \$25 dólares, usando Google Wallet, dentro de las 48 horas siguientes al registro.



Para subir aplicaciones, Google Play ofrece una página que es una excelente información y detalla todos los pasos necesarios para la publicación de aplicaciones en la siguiente página:

https://support.google.com/googleplay/android-developer/answer/113469?hl=es&ref_topic=2365624

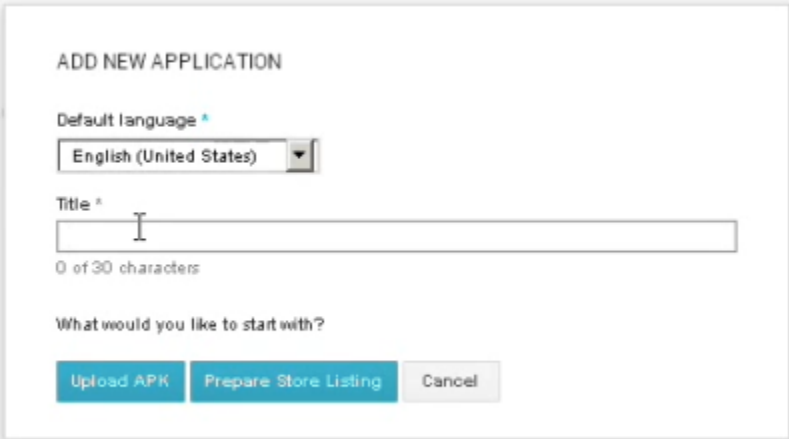
Un punto para promocionar la aplicación desarrollada es el proveer de imágenes o iconos identificables con nuestra aplicación. Google Play también ofrece un conjunto detallado de instrucciones sobre cómo deben ser estos recursos:

<https://support.google.com/googleplay/android-developer/answer/1078870>

En resumen los pasos para subir aplicaciones son:

- Después de crear la cuenta ir a la sección "ALL APPLICATIONS" y dar click en "Add new application":

- Nos pedirá un título:



ADD NEW APPLICATION

Default language *

English (United States)

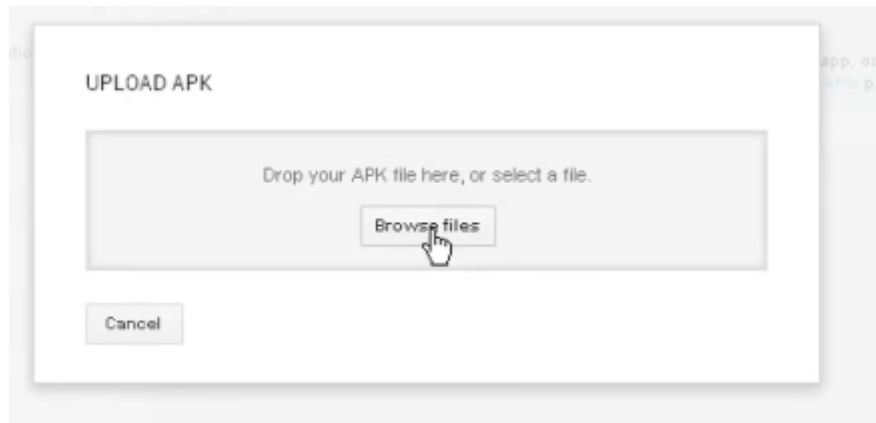
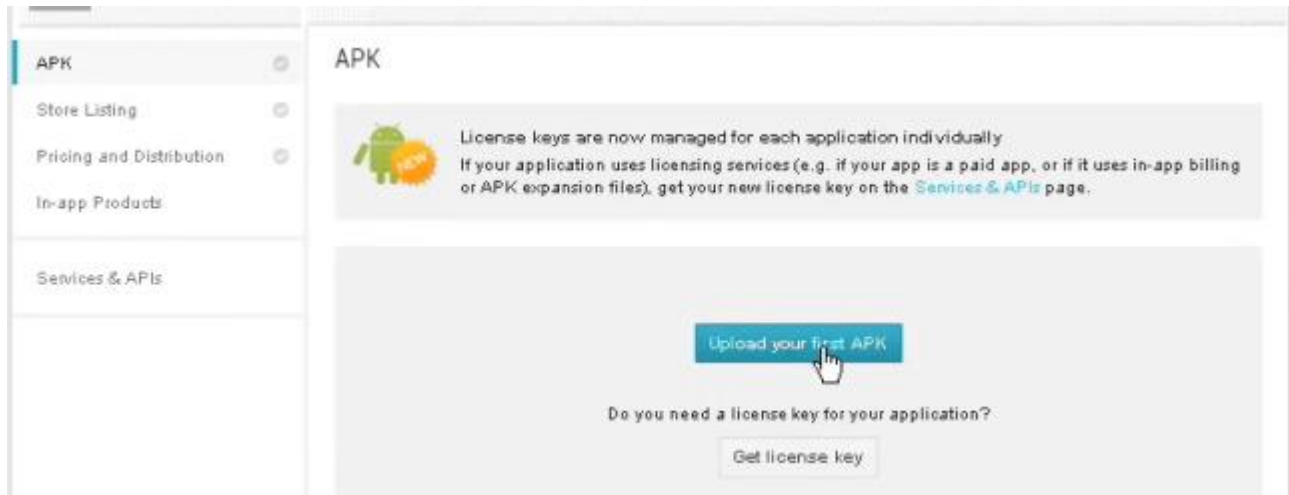
Title *

0 of 30 characters

What would you like to start with?

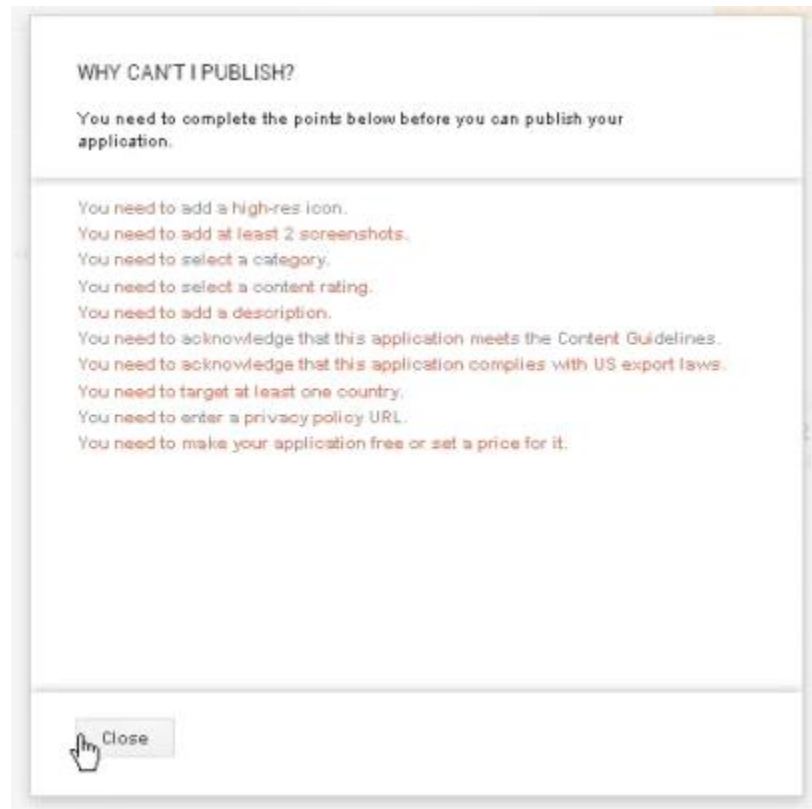
Upload APK Prepare Store Listing Cancel

- No pedirá que subamos el archivo .apk:



- Nos muestra información y capacidades del .apk

- Opcionalmente podemos ver los pasos que faltan para poder publicar:



- Haciendo click en "Store Listing", se requerirá que completemos información importante requerida para la publicación de la aplicación, como el título, la descripción:

- El texto promocional y los cambios recientes:

In-app Products

Services & APIs

14 of 4000 characters

Promo text
English (United States)

0 of 80 characters

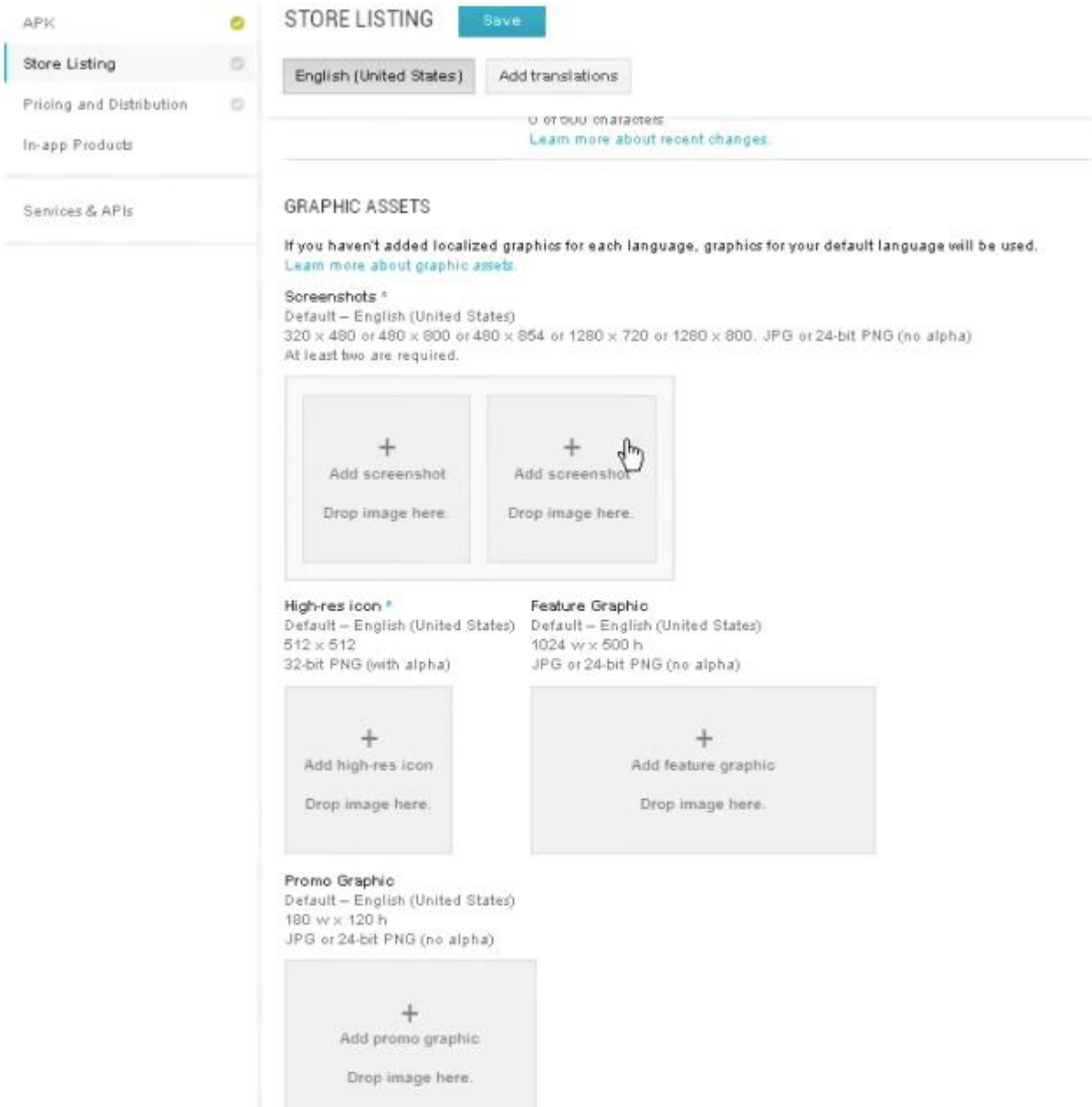
Recent changes
English (United States)

0 of 500 characters

[Learn more about recent changes.](#)

- Nos pedirá que subamos imágenes para la promoción de la aplicación, entre ellas: capturas de pantalla de la aplicación, icono de alta resolución etc, para los detalles requeridos de estas imágenes consulte:

<https://support.google.com/googleplay/android-developer/answer/1078870>



APK

Store Listing

Pricing and Distribution

In-app Products

Services & APIs

STORE LISTING

Save

English (United States) Add translations

0 or 2000 characters
[Learn more about recent changes.](#)

GRAPHIC ASSETS

If you haven't added localized graphics for each language, graphics for your default language will be used.
[Learn more about graphic assets.](#)

Screenshots *
Default – English (United States)
320 x 480 or 480 x 800 or 480 x 854 or 1280 x 720 or 1280 x 800. JPG or 24-bit PNG (no alpha)
At least two are required.

+ Add screenshot
Drop image here.

+ Add screenshot
Drop image here.

High-res icon *
Default – English (United States)
512 x 512
32-bit PNG (with alpha)

+ Add high-res icon
Drop image here.

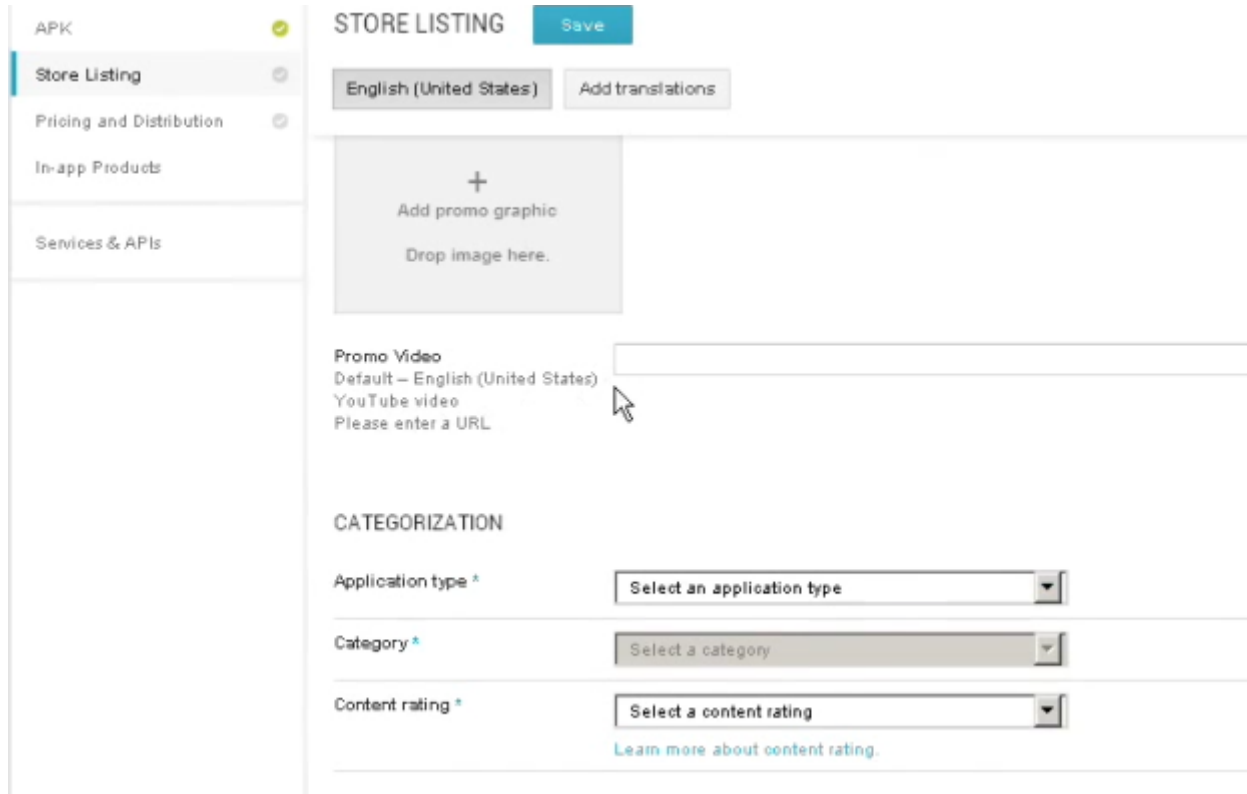
Feature Graphic
Default – English (United States)
1024 w x 500 h
JPG or 24-bit PNG (no alpha)

+ Add feature graphic
Drop image here.

Promo Graphic
Default – English (United States)
180 w x 120 h
JPG or 24-bit PNG (no alpha)

+ Add promo graphic
Drop image here.

- Nos pedirá, opcionalmente, el link a un video promocional de youtube y las clasificaciones en las que cae la aplicación, como tipo de aplicación, categoría, clasificación de contenidos de la aplicación:



The screenshot shows the 'STORE LISTING' form in the Google Play Console. On the left is a sidebar with navigation options: 'APK' (checked), 'Store Listing', 'Pricing and Distribution', 'In-app Products', and 'Services & APIs'. The main content area is titled 'STORE LISTING' and includes a 'Save' button. Below the title, there's a language selector set to 'English (United States)' and an 'Add translations' button. A large box with a plus sign and the text 'Add promo graphic' and 'Drop image here.' is present. Below this is the 'Promo Video' section, which includes a text input field, a 'YouTube video' label, and a prompt 'Please enter a URL'. The 'CATEGORIZATION' section contains three dropdown menus: 'Application type *' (with 'Select an application type' text), 'Category *' (with 'Select a category' text), and 'Content rating *' (with 'Select a content rating' text). A link 'Learn more about content rating.' is located below the content rating dropdown.

- Algunos detalles de contacto del autor de la aplicación y un link, opcional, de las políticas de privacidad:

- Una vez terminemos, guardaremos los cambios con el botón Save:
- Haciendo click en "Pricing and Distribution", estableceremos si la aplicacion es gratis o algun precio para la misma:

APK ✓
Store Listing ✓
Pricing and Distribution ✓
In-app Products
Services & APIs

PRICING AND DISTRIBUTION

Saved

This application is: Paid Free

DISTRIBUTE IN THESE COUNTRIES

You have not selected any countries

☐ SELECT ALL COUNTRIES
☐ Albania
☐ Algeria
☒ Angola
☐ Antigua and Barbuda
☐ Argentina

- Muy importante.** Se requiere aceptar los lineamientos sobre el contenido de android (Android Content Guidelines) y Las leyes de Exportación de US (US export laws) :

APK ✓
Store Listing ✓
Pricing and Distribution ✓
In-app Products
Services & APIs

PRICING AND DISTRIBUTION

Save

☒ Movistar
☐ Orange
☐ Vodafone

☒ Sri Lanka

☒ Sweden [Show options](#)

☒ Switzerland [Show options](#)

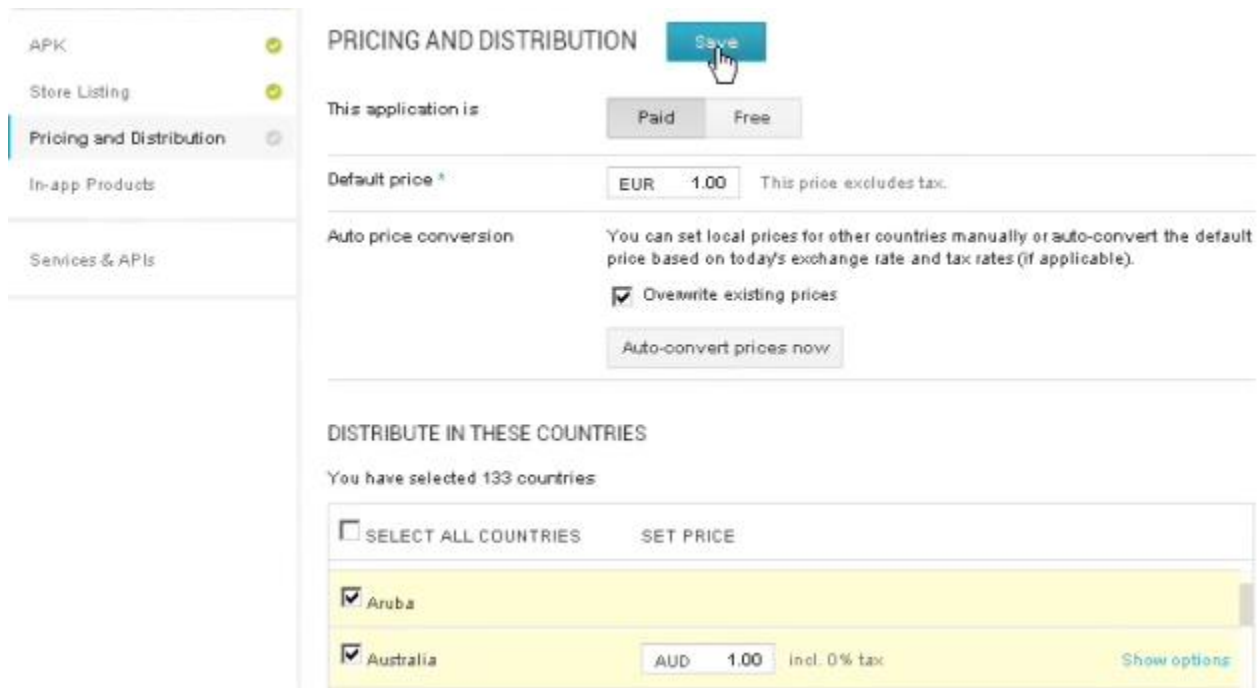
CONSENT

Marketing opt-out ☐ Do not promote my application except in Google Play and in any Google-owned online or mobile properties. I understand that any changes to this preference may take sixty days to take effect.

Content guidelines * ☒ This application meets [Android Content Guidelines](#).

US export laws * ☒ I acknowledge that my software application may be subject to United States export laws, regardless of my location or nationality. I agree that I have complied with all such laws, including any requirements for software with encryption functions. I hereby certify that my application is authorized for export from the United States under these laws. [Learn more](#)

- Guardar las opciones con Save:



PRICING AND DISTRIBUTION [Save](#)

This application is: Paid Free

Default price *: EUR 1.00 This price excludes tax.

Auto price conversion: You can set local prices for other countries manually or auto-convert the default price based on today's exchange rate and tax rates (if applicable).
☒ Oveawrite existing prices
[Auto-convert prices now](#)

DISTRIBUTE IN THESE COUNTRIES

You have selected 133 countries

☐ SELECT ALL COUNTRIES [SET PRICE](#)

<input checked="" type="checkbox"/> Aruba			
<input checked="" type="checkbox"/> Australia	AUD 1.00	incl. 0% tax	Show options

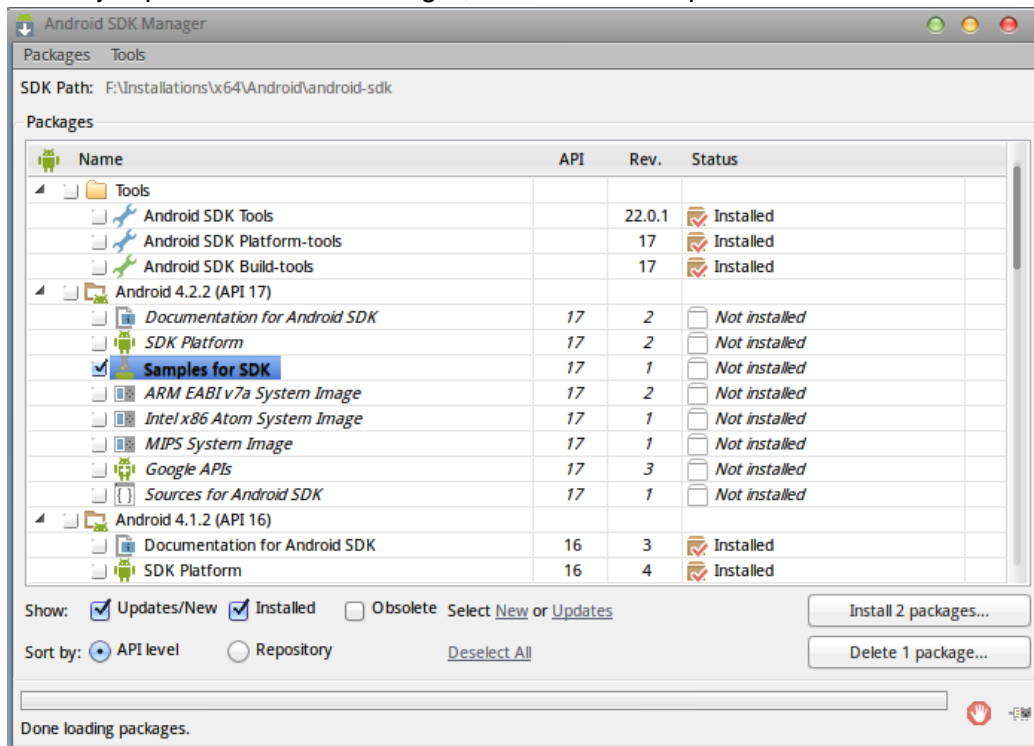
- Y estamos listos para publicar la aplicación, dando click en el botón "Ready to publish" -> "Publish this app". Esta opción NO se puede deshacer:

16. Conclusiones

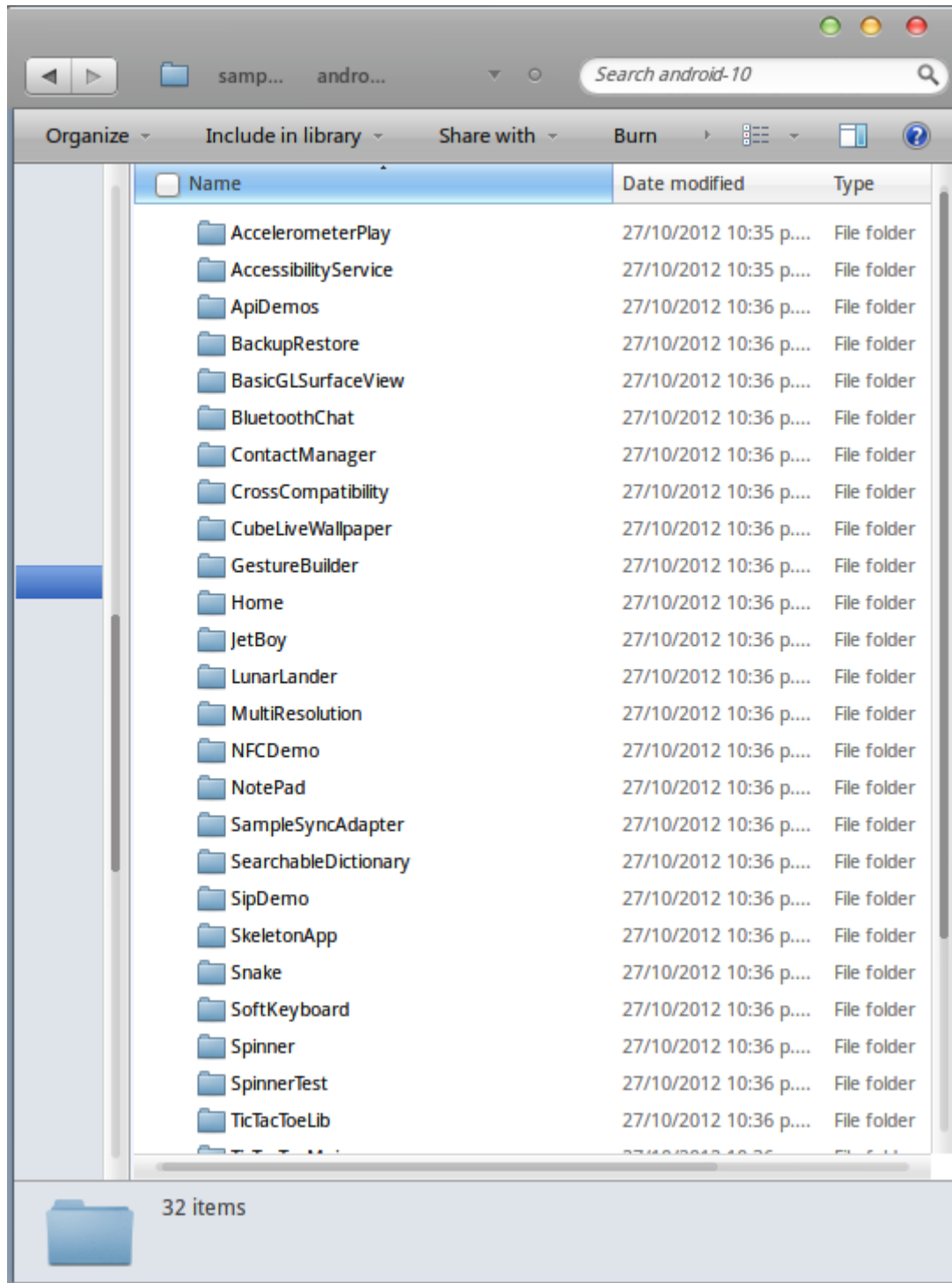
16.1. Uso de Ejemplos del SDK

Podemos crear un proyecto completamente vacío, o seleccionar “Create project from existing source”, si queremos ejecutar una de las aplicaciones de ejemplo incluidas en el SDK (las aplicaciones de ejemplo se encuentran en el directorio “samples/” bajo el directorio de instalación del SDK). Por ejemplo seleccionamos “android-sdk/samples/android-10/snake”.

Para instalar los ejemplos en el SKD Manager, instalamos Samples for SDK:



Tendremos una multitud de ejemplos los cuales podremos tomar como base para conocer diversas implementaciones:



16.2. Recursos de utilidad

- <http://developer.android.com/index.html>
- <https://android.googlesource.com/>
- <http://stackoverflow.com/questions/tagged/android>