



¿Qué es Android?

- **Android** es un sistema operativo de código fuente abierto para dispositivos móviles basado en el **núcleo de Linux versión 2.6**.
- **Android** es un **proyecto de código fuente abierto** desarrollado por la Open Handset Alliance (<http://www.openhandsetalliance.com/>) y liderado por Google. La OHA es una alianza de empresas de hardware, software y de telecomunicaciones que promueve la innovación y apertura del mundo móvil.
- **Android** a pesar de ser un proyecto de código fuente abierto NO lo es en el sentido que cualquiera pueda contribuir en el desarrollo de una nueva versión. No es una distribución, es un producto único cuyo desarrollo controla Google. La apertura de **Android** comienza una vez que es liberada una versión (<http://source.android.com/source/downloading.html>).
- **Android** está liberado bajo una licencia “business-friendly”, **Apache 2.0 (ASL)** “más permisiva para el uso comercial”. No es una licencia *copyleft*

¿Qué es Android?

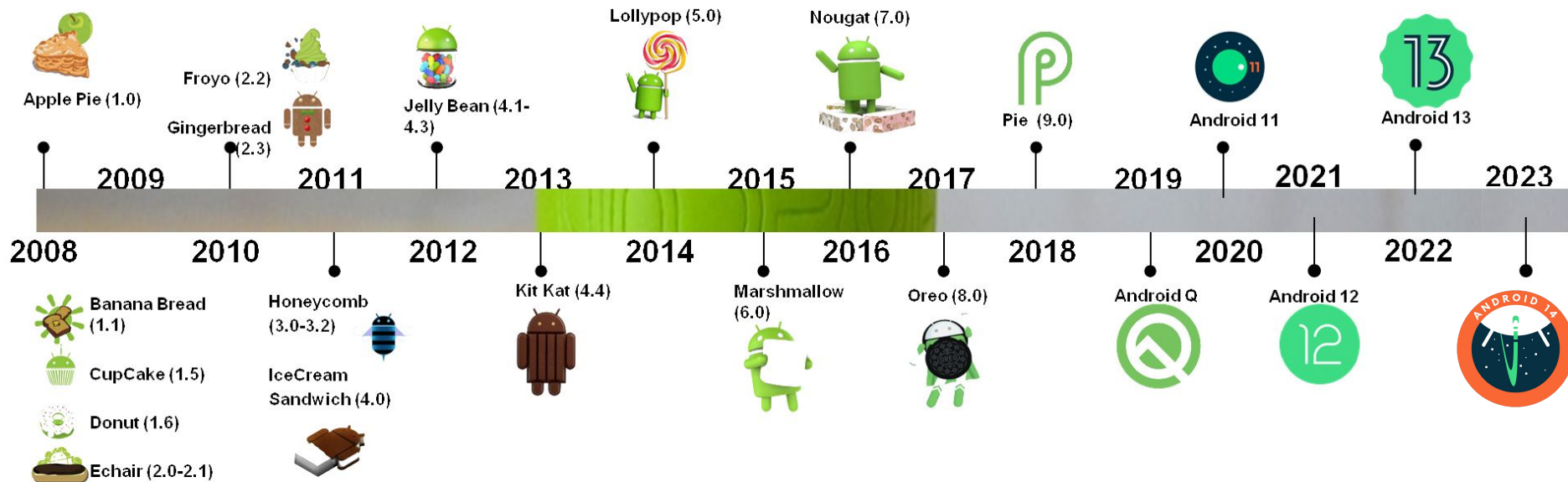
- Las aplicaciones nativas para **Android** son desarrolladas en **JAVA** y **Kotlin** usando el Android SDK (Android Software Development Kit), que incluye un conjunto de herramientas y APIs. Actualmente el IDE oficial para desarrollar en Android es el Android Studio.
- **Kotlin** es patrocinado por Google, anunciado como el lenguaje oficial para los desarrollos en Android a partir del año 2017.
- No hay diferencia entre las aplicaciones que vienen incorporadas a la plataforma **Android** y las construidas con el **Android SDK**. Es posible desarrollar aplicaciones que aprovechen los recursos disponibles por el dispositivo (cámaras, reproductor de media, receptor de GPS, pantallas táctiles, hardware para NFC-Near Field Communication-).
- Google ofrece una tienda global online de distribución digital de aplicaciones móviles llamada **Google Play** en la que los desarrolladores pueden ofrecer sus aplicaciones a los usuarios **Android**. **Google Play** también ofrece servicios de actualización de aplicaciones a sus usuarios y a los programadores acceso a los servicios y librerías (Google Maps). También es una tienda de medios digitales.

Android y otros entornos móviles

- **Android** se ubica junto a una nueva gama de sistemas operativos móviles modernos diseñados para dar soporte al desarrollo de aplicaciones en un hardware móvil cada vez más poderoso.
- **Android** propone nuevas posibilidades para aplicaciones móviles al ofrecer un entorno de desarrollo abierto construido sobre Linux. El acceso al hardware y datos almacenados en el dispositivo están disponibles para todas las aplicaciones a través de una serie de APIs. Soporta interacción entre aplicaciones instaladas (intents) y todas las aplicaciones tienen igual jerarquía (no hay diferencia entre las apps nativas y las desarrolladas por terceras partes).
- Windows Mobile y iPhone de Apple también ofrecen un rico entorno de desarrollo para aplicaciones móviles, sin embargo, a diferencia de **Android**, son sistemas operativos propietarios que dan prioridad a las aplicaciones nativas sobre las creadas por terceros y restringen la comunicación entre las aplicaciones y los datos nativos del teléfono.
- El objetivo del proyecto **Android** es crear un producto que mejore la experiencia móvil de los usuarios.



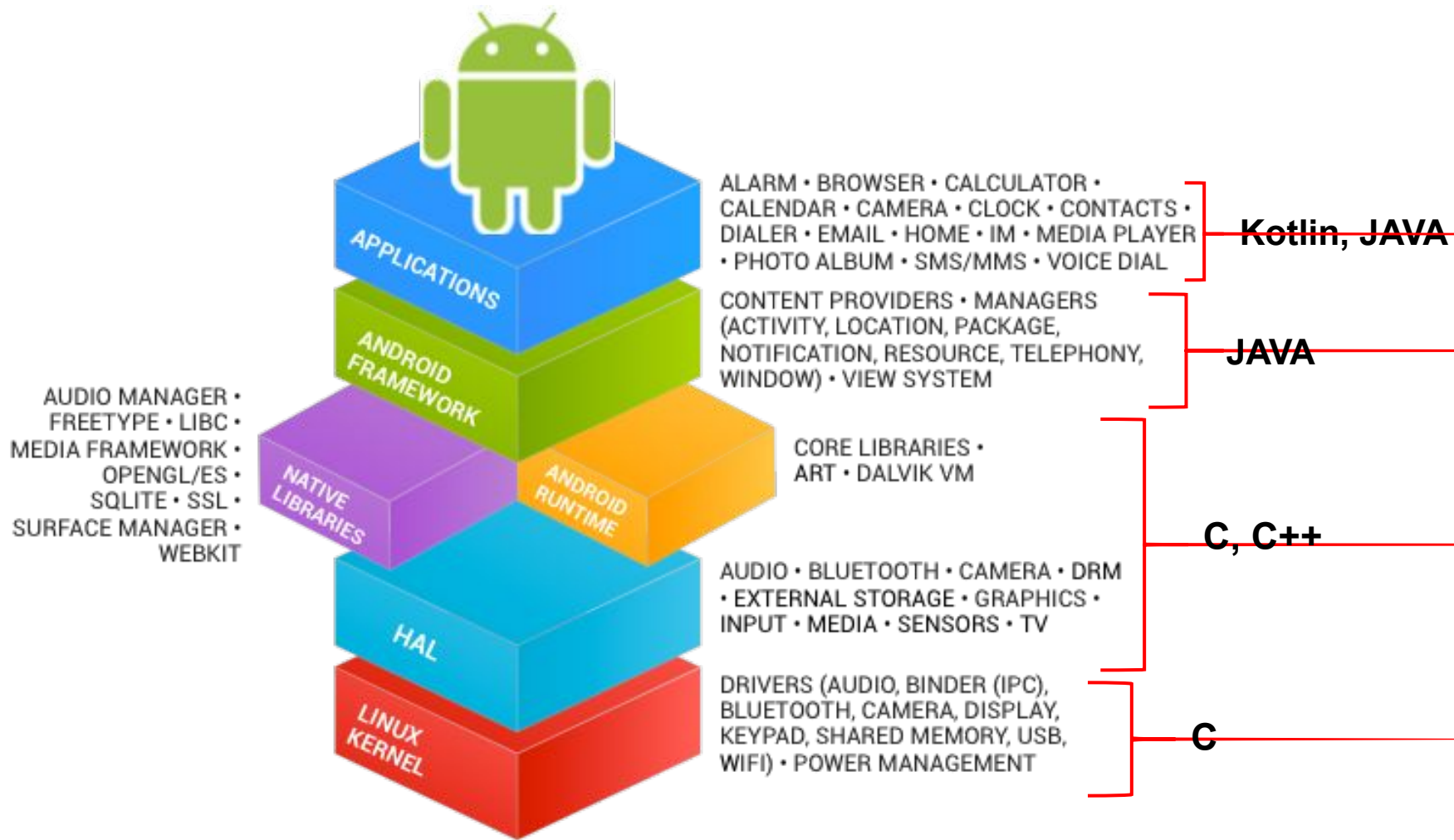
Versiones de Android



A partir de 2009 y hasta el 2018, las versiones de Android han sido desarrolladas usando nombres de golosinas/tortas de EEUU y siguiendo un orden alfabético.

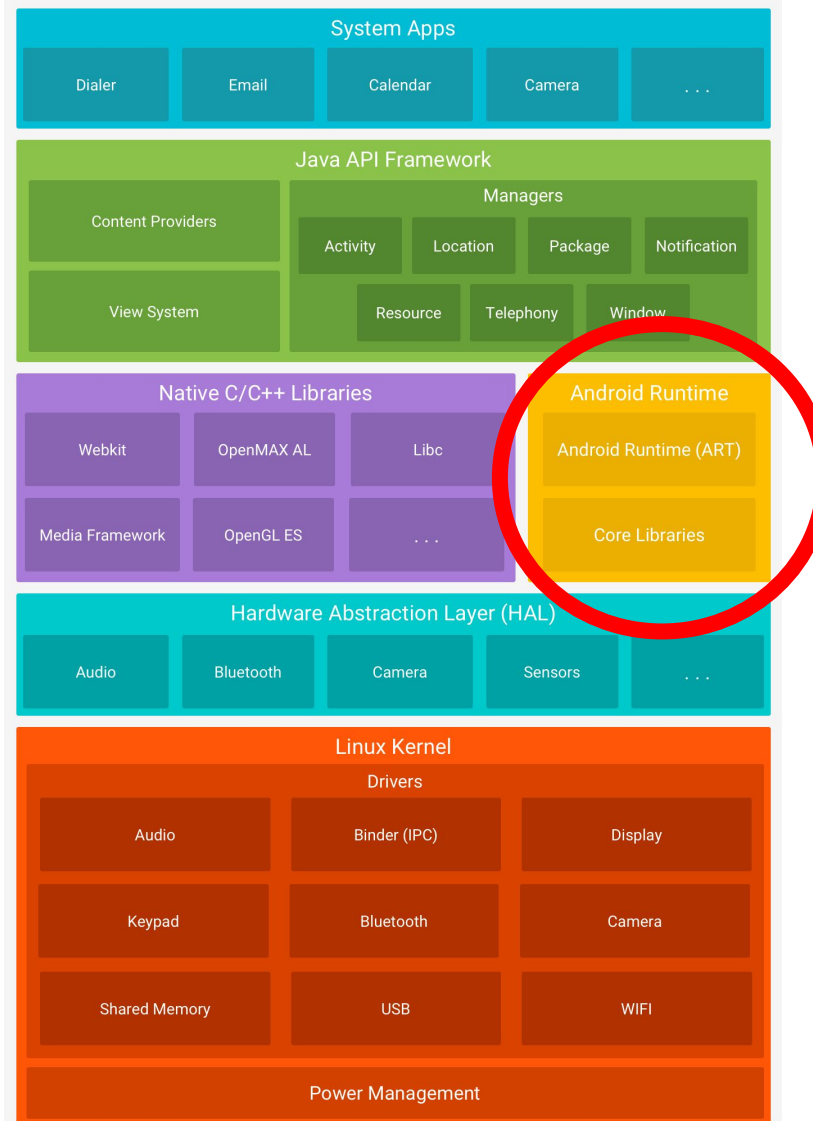
Arquitectura de Android

Android es un stack de software de código fuente abierto



Arquitectura de Android

Android es un stack de software de código fuente abierto



Arquitectura de Android

Application Framework

Esta capa provee de un conjunto de componentes de software (clases, interfaces) utilizados para crear aplicaciones. El *application framework* viene preinstalado con Android.

El conjunto de servicios y sistemas que ofrece, incluye:

Activity Manager: Gerencia el ciclo de vida de los “activities” y mantiene un “backstack” de navegación del usuario.

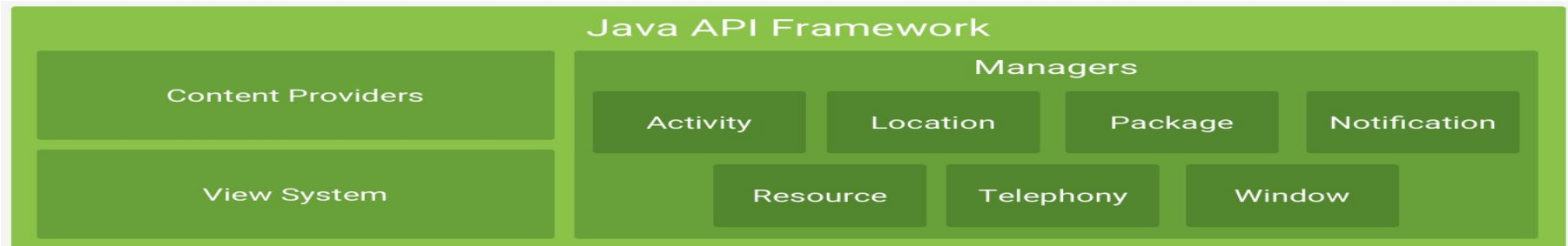
Content Provider: permite a las aplicaciones acceder a datos de otras aplicaciones (por ejemplo Contactos) y compartir sus propios datos.

Resource Manager: provee acceso a recursos que no es código, como *localized strings*, gráficos, etc.

Notification Manager: permite a las aplicaciones desplegar notificaciones de alerta en la barra de estado. Eventos del tipo: llegó un mensaje, citas, alertas de proximidad, etc pueden presentarse de una manera no intrusiva.

Location Manager: un teléfono Android siempre sabe dónde está. Además del GPS Android calcula la información de posición usando la proximidad con las torres de telefonía celular.

View System: es un conjunto rico y extensible de vistas que pueden usarse para construir aplicaciones, incluyendo listas, grillas, cuadros de texto, botones, etc



Arquitectura de Android

Librerías

Android incluye librerías escritas en C o C++, compiladas para el hardware particular que usa el dispositivo móvil y preinstaladas por el fabricante del dispositivo. Estas librerías son invocadas por programas de alto nivel.

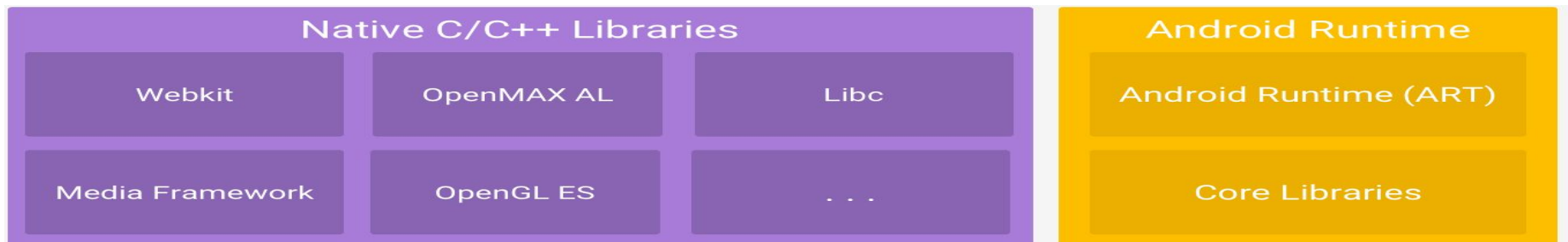
Browse Engine: es el motor de navegación en Internet de código fuente abierto basado en la librería WebKit. Es el mismo motor que usa Chrome, Safari.

SQL database: Android incluye un motor de bd liviano, SQLite (usado por firefox y iphone).

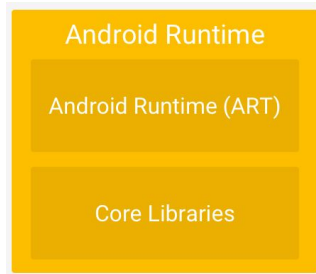
Soporte avanzado de gráficos 2D y 3D: es posible combinar elementos 2D y 3D en una interfaz gráfica. Las librerías usan hardware 3D si el dispositivo dispone del mismo o un software de *rendering* si no lo dispone; animaciones mediante SGL (Scalable Games Languages) y OpenGL.

Media Framework: con Android es posible reproducir videos, música de una amplia variedad de formatos (mp3, mpeg-4, etc)

Surface Manager: es el administrador de ventanas de Android. Los comandos de dibujo van a parar a un *bitmap offscreen* que luego se combina con otros *bitmaps* para formar la presentación que ve el usuario.



Arquitectura de Android Runtime



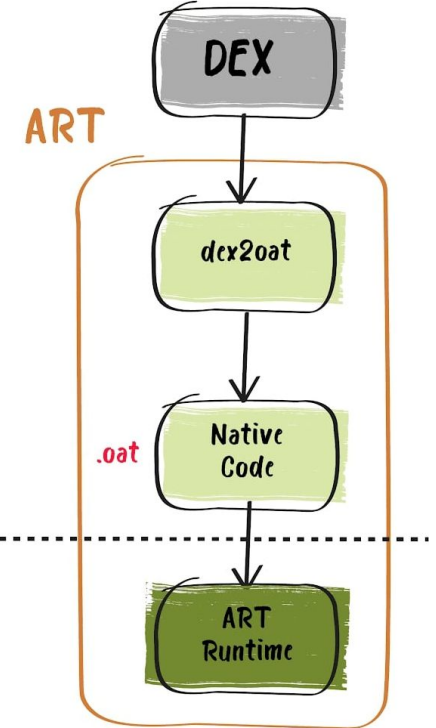
Las aplicaciones Android escritas en Kotlin o JAVA se ejecutan en el **ART (Android Runtime)**.

ART (Android Runtime) usa una compilación AOT (Ahead-Of-Time, compilación anticipada) junto con recolección optimizada de elementos no utilizados (GC) y ejecutan código **.dex (Dalvik Executable)**.

Los archivos .class y .jar son convertidos en **tiempo de compilación** a **.dex**. (por el compilador dx).

Los archivos **.dex** y los recursos de un proyecto Android se empaquetan en un archivo **.apk** (Android Package). La herramienta **aapt** realiza el empaquetado.

Las aplicaciones Android se ejecutan en su propio proceso Linux

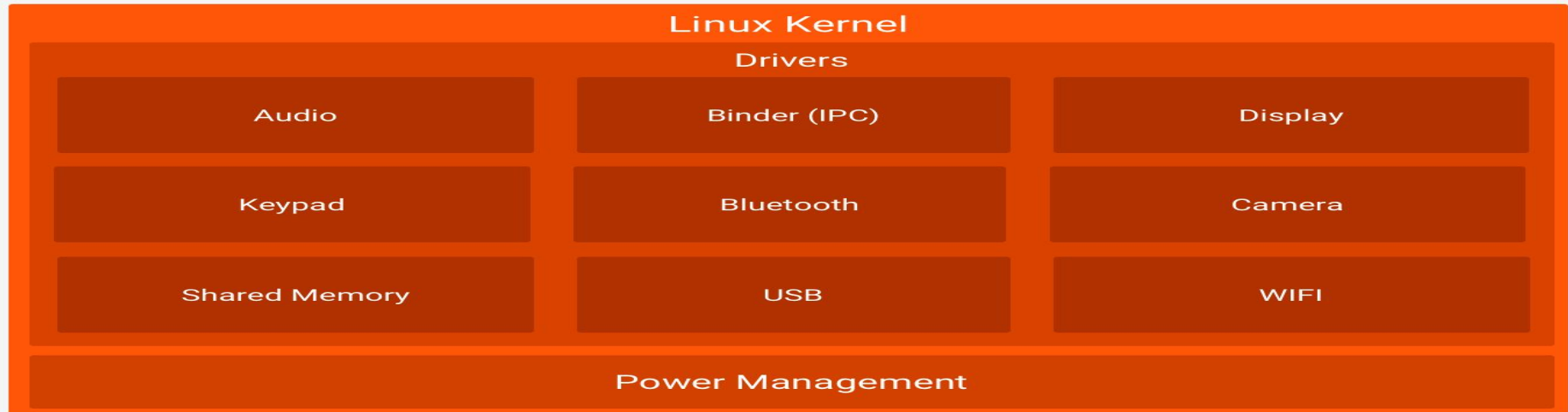


El conjunto de librerías *core* Kotlin de Android está “cerca” de JSE, incluye colecciones, corrutinas, funciones lambda, reflection, etc. La mayor diferencia con JSE son las librerías de UI que fueron reemplazadas.

Arquitectura de Android

El Kernel Linux

- Android se basa en el kernel Linux.
- El kernel Linux ofrece una capa de abstracción entre el hardware y el resto del **stack** de software. Provee de conjunto de servicios centrales como manejo de procesos, memoria y filesystem, etc.
- ART utiliza funcionalidades del kernel como la generación de subprocesos y la administración de memoria de bajo nivel.
- Los usuarios nunca ven el subsistema Linux



Arquitectura de Android

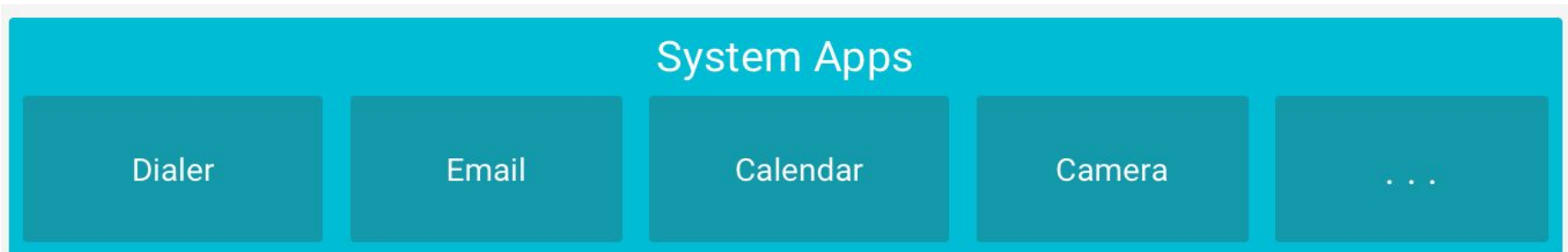
Aplicaciones y Widgets

Es la capa de más alto nivel de la arquitectura de **Android**. Los usuarios finales solamente ven esta capa.

Las aplicaciones son programas que ocupan toda la ventana e interactúan con el usuario. Los **widgets** son programas que se muestran en una porción de la ventana del **Home Application**.

Todas las aplicaciones están en el mismo campo de juego, las que vienen preinstaladas en nuestro dispositivo se desarrollan usando la misma API que las de las terceras partes e incluso pueden reemplazarlas.

La distribución de app Android es más abierta que la de sus competidores. Google ofrece la **tienda de app Google Play**, pero también existen otras tiendas de app Android como **Amazon**, **GetJar**, **Samsung App**, a través de las que los programadores pueden ofrecer sus programas Android a los usuarios (<https://play.google.com>).



Las Aplicaciones Android

- Las aplicaciones Android son **gerenciadas por un contenedor** con ciclos de vida completos que facilitan el uso eficiente de la memoria.
- Cada **aplicación Android** se ejecuta en un **proceso Linux diferente** y es independiente de las demás aplicaciones, cada proceso tiene su propio entorno de ejecución y este se crea cuando la aplicación es ejecutada por primera vez.
- Una aplicación Android es un archivo **.apk** (Android Package).
- Todo el código compilado, junto con los archivos de datos o recursos requeridos por un aplicación, son empaquetados dentro de un archivo **.apk**
- Las app Android deben firmarse digitalmente antes de ser instaladas en un dispositivo (<http://developer.android.com/tools/publishing/app-signing.html>)
- A partir del 2021, las apps nuevas deberán publicarse con **Android App Bundle** en **Google Play**, es un formato de publicación que incluye todos los recursos y el código compilado delegando la generación del **apk** y la firma a la tienda (ahorrando espacio).

¿Cómo desarrollar aplicaciones Android?

- Las aplicaciones Android se escriben en **Kotlin** y en **JAVA**.
- Durante el proceso de desarrollo, el programador crea archivos de configuración específicos de Android y escribe la lógica de la aplicación en **Kotlin** o **JAVA**.
- Las herramientas del **Android SDK** convierten estos archivos de la aplicación en una aplicación Android en forma transparente. Cuando el programador dispara el proceso de *deployment* en el IDE, la aplicación completa es compilada, empaquetada, desplegada y arrancada.

Entorno de Desarrollo

El **Software Development Kit de Android** es el único toolkit que necesitamos para desarrollar aplicaciones en Android. Se trata de herramientas basadas en lenguajes de línea de comando.

Desde Dic de 2014 el IDE oficial es el **Android Studio**. El Android Studio tiene incorporado el SDK de Android.

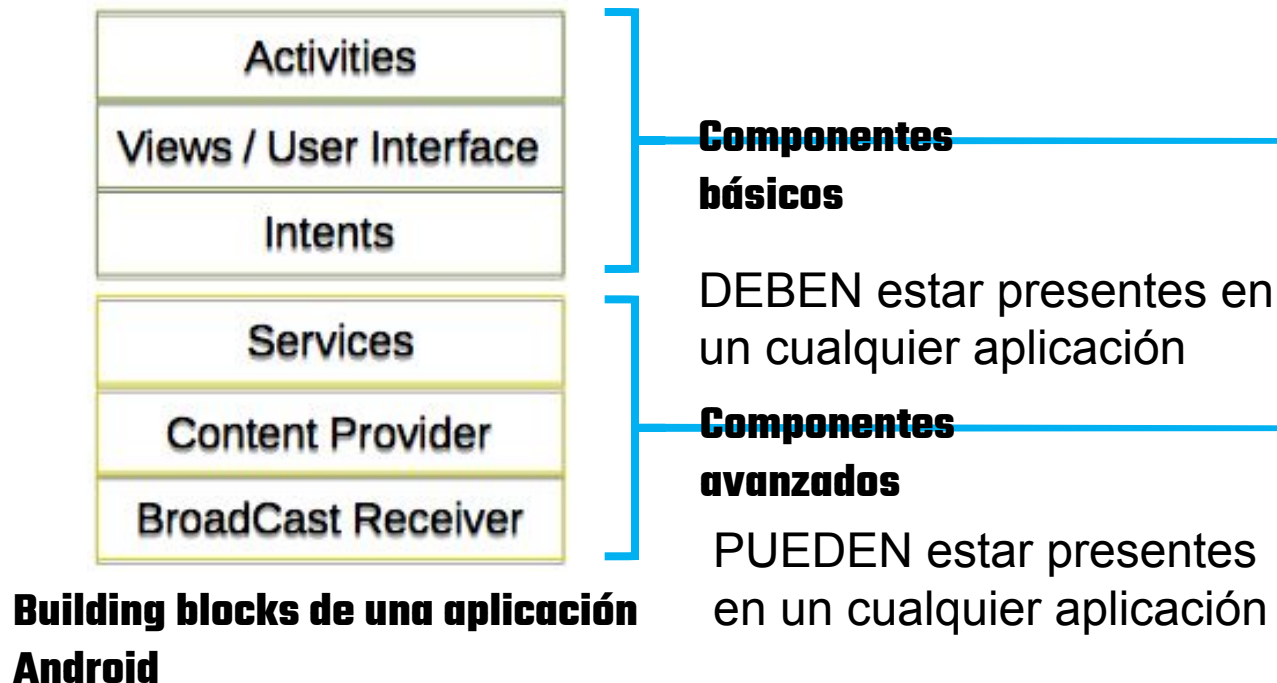
Android Studio está basado en el IDE IntelliJ.

Para descargar:

<http://developer.android.com/intl/es/sdk/index.html>

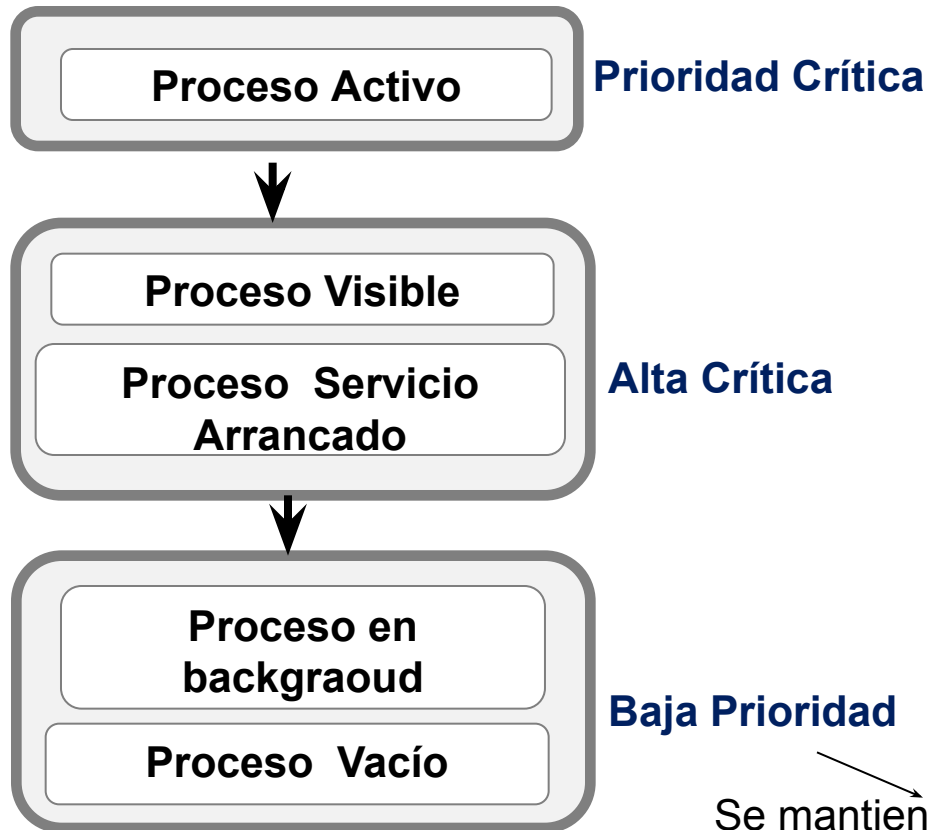
Componentes de una Aplicación Android

Las componentes o unidades de código independientes constituyen los “building blocks” de las aplicaciones Android. Cada componente es una entidad individual, juega un rol específico, tiene su propio ciclo de vida.



El ciclo de vida de las aplicaciones Android

Todas las aplicaciones Android continúan ejecutándose en memoria hasta que el sistema necesite recursos para otras aplicaciones.



Android maneja los recursos de las aplicaciones para asegurar una **buena experiencia del usuario**.

Android usa una **jerarquía de prioridades** para determinar el orden en que las aplicaciones terminarán.

Los procesos que hostean las aplicaciones podrían ser dados de baja para liberar recursos que serán asignados a aplicaciones de más alta prioridad.

Se mantienen en un *cache* de “menos usados recientemente”

Activities

- Un **Activity** es la representación visual de una Aplicación Android, es la interfaz visual (UI). Una aplicación Android típicamente está formada por múltiples pantallas ó Activities. Son las pantallas con las que interactúa el usuario en el dispositivo.
- Podemos pensar un app Android en forma análoga a un sitio web:

Sitio WEB	Aplicación Android
Páginas	Activities
Home page	Main Activity (Manifest.xml)
Navegación entre páginas	Navegación entre Activities

- Desde el punto de vista del desarrollador un **Activity** es una subclase de *Activity*. El *Activity* muestra elementos de UI que se implementan con *Views* (etiquetas, botones, formularios, etc).
- Los **Activities** usan *Views (Widgets)*, *Layout Managers* y *Fragments* para crear la interfaz de usuario e interactuar con este último.
- El ciclo de vida de los activities es manejado por un **Activity Manager**.

Ciclo de Vida de un Activity

El **Activity Manager** es el responsable de crear, destruir y **gerenciar** los activities. Android provee de un entorno gerenciado basado en un contenedor.

Posibles estados de un **Activity**:

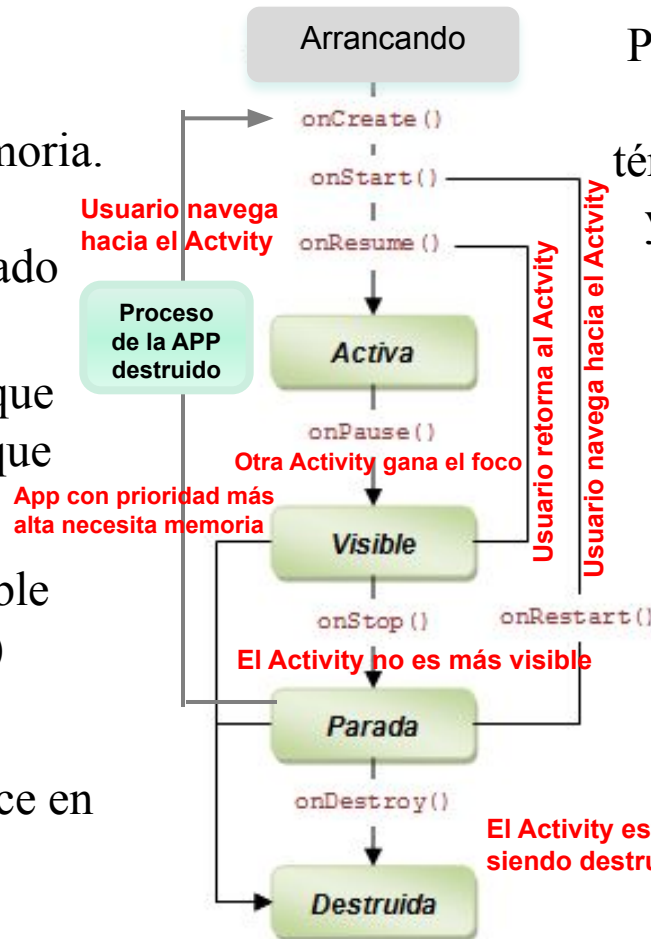
Arrancando: el Activity no está en memoria. Mientras está arrancando ejecutará un conjunto de métodos que lo pasará a estado **Activa**.

Activa (Ejecutando): el Activity es el que está actualmente en la pantalla y con el que está interactuando el usuario.

Visible (Pausado): el Activity está visible pero no tiene foco (parcialmente visible)

Parada (Detenido): el Activity no está visible, está en 2do plano pero permanece en memoria.

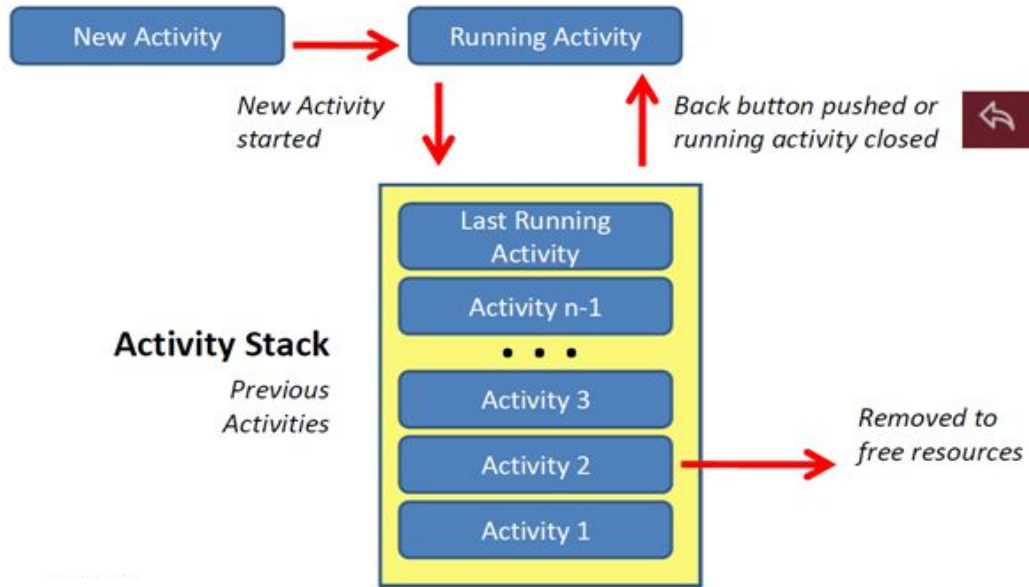
Destruída: el Activity no está más en memoria



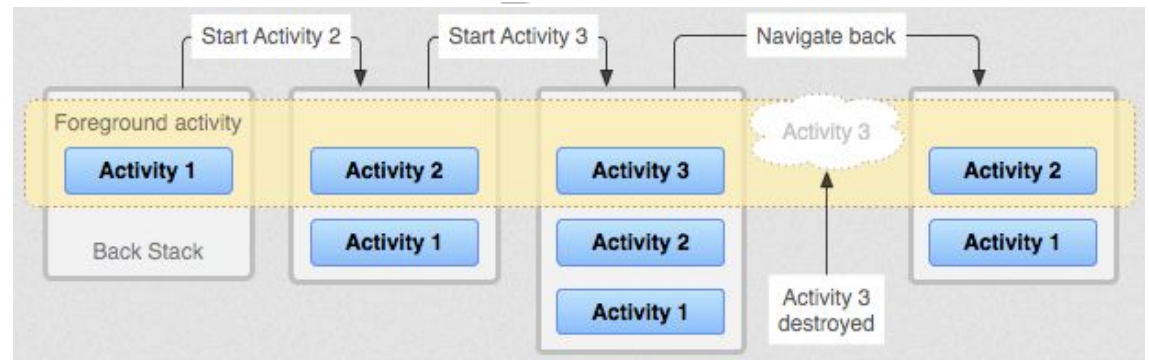
Pasar de **Arrancando** a **Activa** es costoso en términos de computación y consumo de batería.

Cada vez que un Activity cambia de estado, se invocan a los métodos del ciclo de vida. El programador puede indicar qué hacer en la transición de un estado a otro.

Pila de Activities



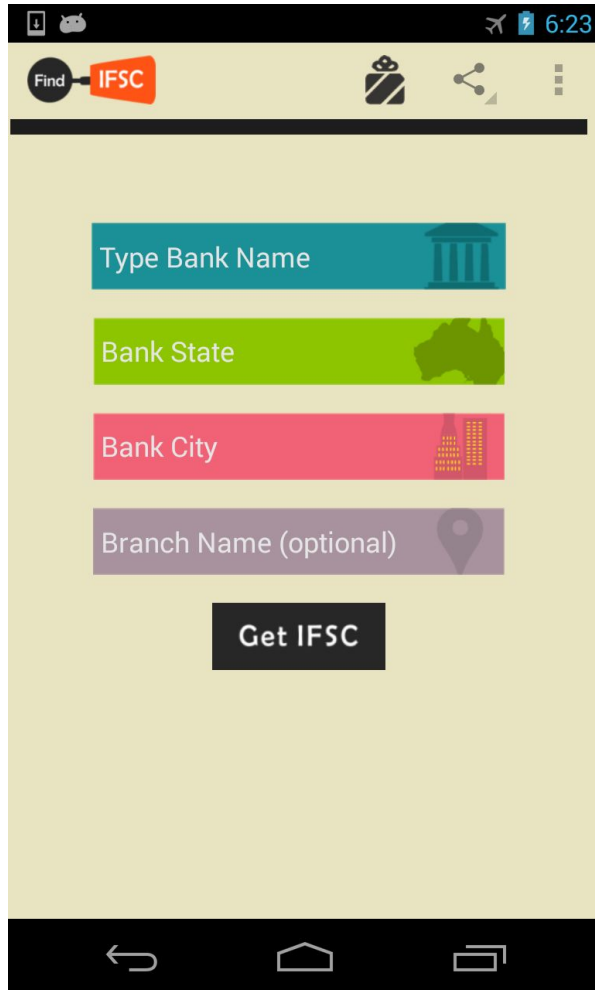
Cuando el usuario presiona el botón *Atrás* ó se cierra una app, se destruye la actividad y se reanuda la actividad anterior (se restaura el estado anterior de su interfaz de usuario).



Típicamente una aplicación contiene múltiples **Activities** y a partir de cada una de ellas se pueden iniciar otros **Activities** de la misma App o de otras App instaladas en el dispositivo .

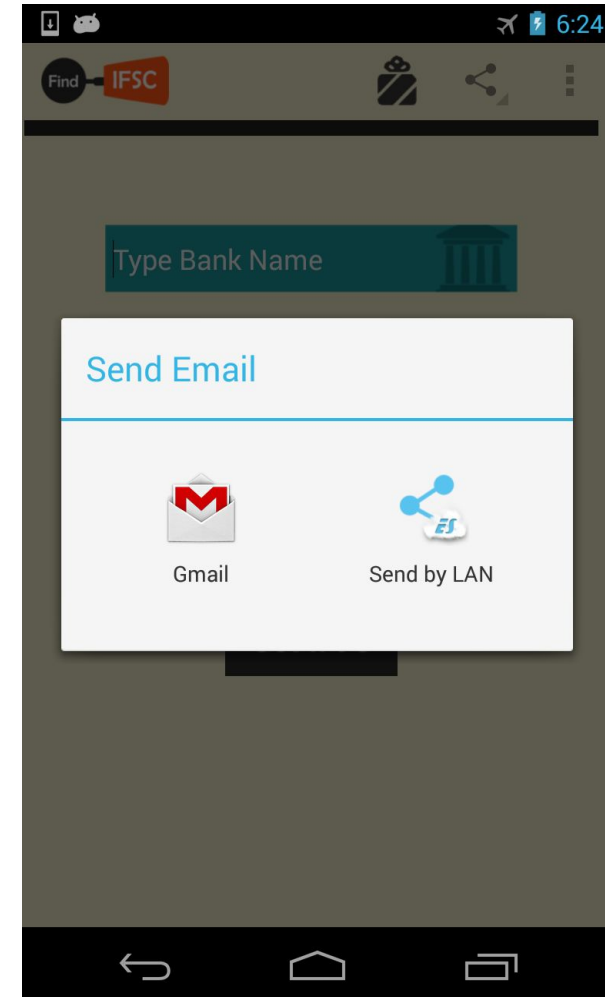
Las actividades se organizan en una **pila de actividades**

Estados de un Activity



Activa o Ejecutándose

Cada vez que una **caja de diálogo** requiere atención por ej. batería baja, enviar email, etc, el activity estará parcialmente visible, en ese punto se invoca al `onPause()`



Pausado

Métodos del Ciclo de Vida de un Activity

- **onCreate(Bundle)**: es invocado cuando el *activity* es creado por primera vez. Es invocado sólo una vez durante el ciclo de vida de un Activity. Es un buen punto para setear el layout del Activity. Es el único método que se debe sobrescribir.
- **onStart()**: indica que el *activity* será mostrada al usuario. Es un buen lugar para reinicializar variables, setear datos.
- **onResume()**: invocado cuando la *activity* comienza a interactuar con el usuario (por ejemplo, comenzar animaciones y/o música, refrescar la UI)
- **onPause()**: invocado cuando el *activity* queda en segundo plano ya que otro toma el foco. Aquí debería guardarse el estado del programa, liberar recursos o parar threads que están en background.
- **onStop()**: invocado cuando el *activity* ya no está visible al usuario. Se podría guardar el estado de la app e interrumpir operaciones de uso intensivo de CPU.
- **onDestroy()**: invocado antes que el *activity* sea removido de memoria. Es la última invocación que recibirá el Activity en su ciclo de vida. No hay garantías de que sea invocado.

Los Activities que no están ejecutándose pueden estar en estado **Detenido** y el sistema podría “matar” en cualquier momento los procesos que los alojan, por ejemplo si la AA que se está ejecutando necesita memoria. Es importante tener esto en cuenta en el momento del diseño del Activity.

Hola Mundo Android!

Un *Activity* es una componente de aplicación que se utiliza para representar las pantallas con las que el usuario interactúa. Es la IU de usuario de la aplicación, cada pantalla de la aplicación es un *Activity*.

Cada *Activity* tiene una vista o conjunto de vistas agrupadas bajo un contenedor (*Layout*).

```
package ejemplo.holamundo

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_hola_mundo)
    }
}
```

La pantalla principal de la aplicación

El método **onCreate()** es invocado por el sistema para arrancar un *Activity*. El código de este método realiza la inicialización y la configuración de la interfaz del usuario.

La actividad visualizará una determinada vista que está definida en los recursos

AndroidManifest.xml

Todas las piezas que conforman una aplicación Android deben unirse para que la aplicación pueda ejecutarse. El mecanismo usado para definir las relaciones entre las diferentes componentes es el archivo **AndroidManifest.xml**

AndroidManifest.xml es el archivo descriptor de aplicaciones **Android**.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="labo.holamundo">
    <application android:icon="@drawable/icon">
        <activity android:name=".HolaMundo" 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>
```

La acción **android.intent.action.MAIN** indica que es un punto de entrada a la aplicación. La categoría **android.intent.category.LAUNCHER** ubica a este activity en la ventana principal. El tag **intent-filter** define los intents que pueden ser usados para arrancar el Activity.

Esta aplicación contiene un único Activity, cuya clase se llama .HolaMundo.

@string: hace referencia a información guardada en uno de los archivos de recursos (string).

El atributo label es obtenido del archivo de recursos string como app_name

Elementos del proyecto Android en Android Studio

Archivo descriptor de la aplicación Android se indican todos los componentes de la app. Se declaran los permisos que requerirá la aplicación, versión mínima de Android para poder ejecutar la aplicación, el paquete, la versión de la aplicación, metadatos, etc

Contiene el código fuente.

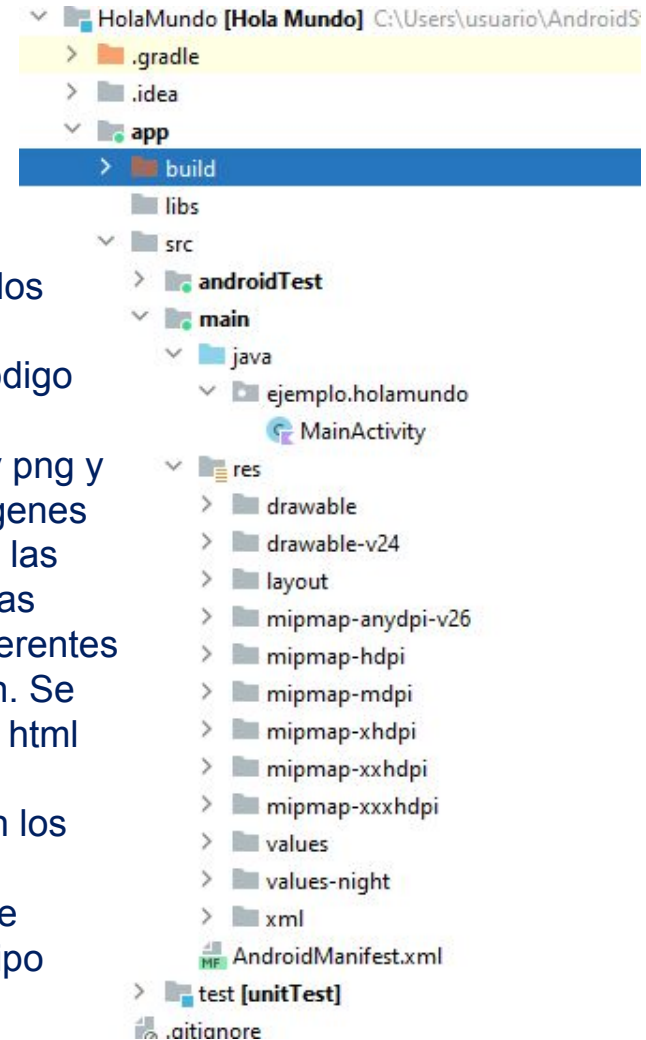
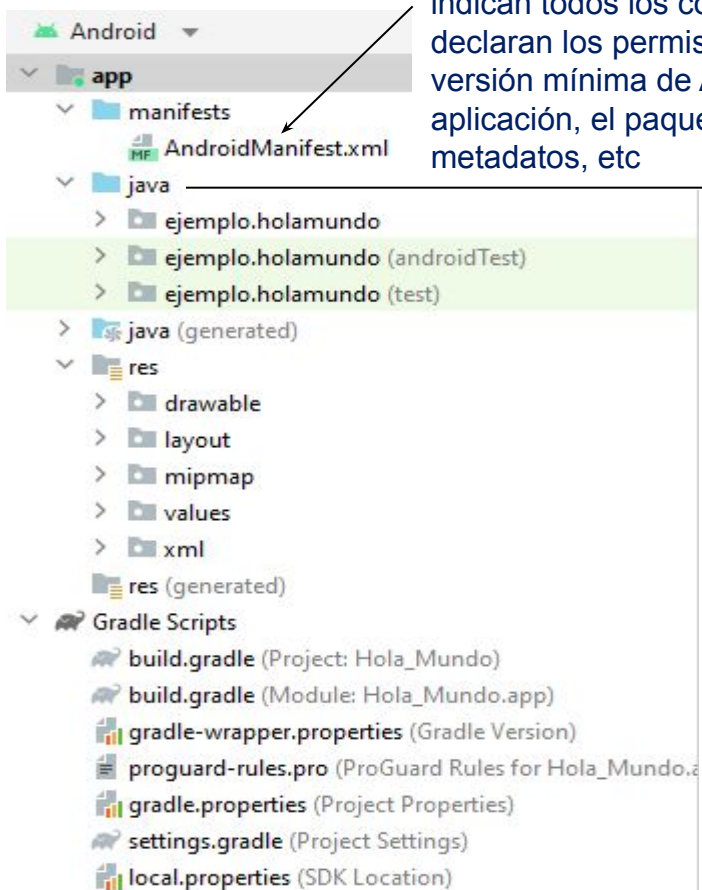
La carpeta **res** contiene los recursos usados por la aplicación que no sea código java:

drawable: archivos jpg y png y descriptores xml de imágenes

layout: archivos xml con las vistas de la aplicación. Las vistas constituyen las diferentes pantallas de la aplicación. Se usa un formato similar al html para páginas web

menu: archivos XML con los menús de cada *activity*.

value: archivos XML que representan valores de tipo string, colores o estilo.



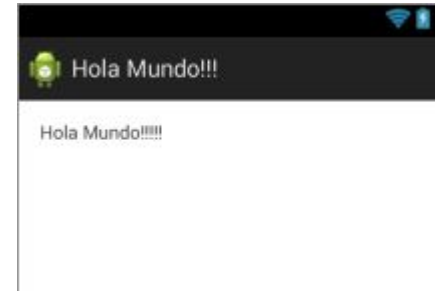
Externalizar Recursos: la carpeta /res

res/layout/activity_hola_mundo.xml: especifica el **layout** de la pantalla. En nuestro ejemplo tenemos una sola pantalla que es cargada por el código HolaMundo.java

```
<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="labo.holamundo.HolaMundo" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hola_mundo" />

</RelativeLayout>
```



res/values/string.xml: describe todos los strings que usa la app

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

    <string name="app_name">Hola Mundo!!!</string>
    <string name="hola_mundo">Hola Mundo!!!!</string>
    <string name="action_settings">Settings</string>

</resources>
```

La aplicación busca el valor del string en el archivo de recursos **res/values/string.xml**

La clase R

Android genera automáticamente la **clase R** para cada proyecto. La función de esta clase es proveer acceso a los **recursos** de la aplicación desde código JAVA.

```
package ejemplo.holamundo;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_action_search=0x7f020000;
        public static final int ic_launcher=0x7f020001;
    }
    public static final class id {
        public static final int menu_settings=0x7f070000;
    }
    public static final class layout {
        public static final int hola_mundo=0x7f0a000e;
    }
}
```

```
public static final class menu {
    public static final int activity_main=0x7f060000;
}
public static final class string {
    public static final int app_name=0x7f040000;
    public static final int hola_mundo=0x7f040001;
    public static final int action_settings=0x7f0a000f;
}
public static final class style {
    public static final int AppTheme=0x7f050000;
}
}
```

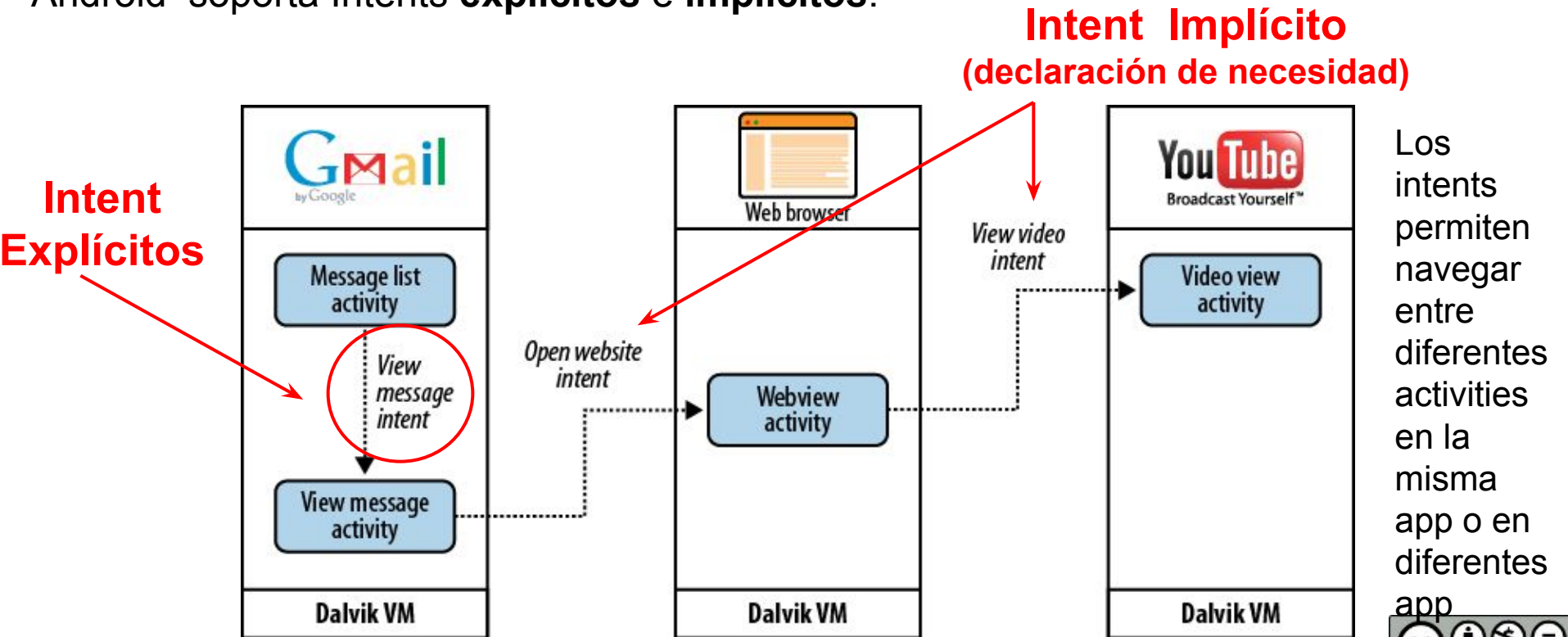
Los miembros declarados final de la clase R representan elementos de UI. Los **Recursos** son tratados especialmente en Android: se compilan a código binario. Las animaciones, vistas, layouts, string, colores, arreglos se definen en XML y luego éstos son procesados por la herramienta aapt (Android Resource Compiler) y compiladas en la **clase R**. Luego los recursos son accesibles a través de la clase R. La clase R no hay que modificarla y es recreada cada vez que se hacen cambios en la carpeta de recursos.

Intents

Los *Intents* son mensajes asincrónicos que permiten a las componentes de una app solicitar la funcionalidad de otras componentes. Los Intents pueden arrancar un activity, solicitar que un servicio arranque o se detenga o simplemente hacer difusiones.

Los *Intents* permiten navegar desde un *Activity* a otro, comunicarse y transferir datos entre *Activities*.

Android soporta Intents **explícitos** e **implícitos**.



Intents Explícitos

Los **intents explícitos** definen de manera explícita la componente que se desea invocar usando la clase Java como identificador.

Cuando se abre un activity desde otro activity en la misma app se usan **intents explícitos**.

Se transfieren
datos

Se arranca el
Activity

```
override fun onClick(v: View?) {  
    val view = v?.rootView  
    when (view?.id) {  
        R.id.about_button -> {  
            val i =  
                Intent(applicationContext, SegundoActivity::class.java)  
            i.putExtra("title", "Segunda Actividad 2")  
            startActivity(i)  
        }  
        else -> {  
            // más botones ....  
        }  
    }  
}
```

PrimerActivity



**Intent
Explícito**



SegundoActivity

PrimerActivity

Intents Implícitos

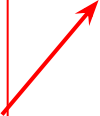
Los intents implícitos son los intents que en lugar de definir la componente exacta a invocar, definen la acción que se desea realizar.

La decisión de manejar esta acción la tiene el SO, quién decide cuál es el componente que se ajusta a la solicitud.

Siempre que se delegue la responsabilidad en una app diferente a la nuestra, debemos usar intents implícitos.

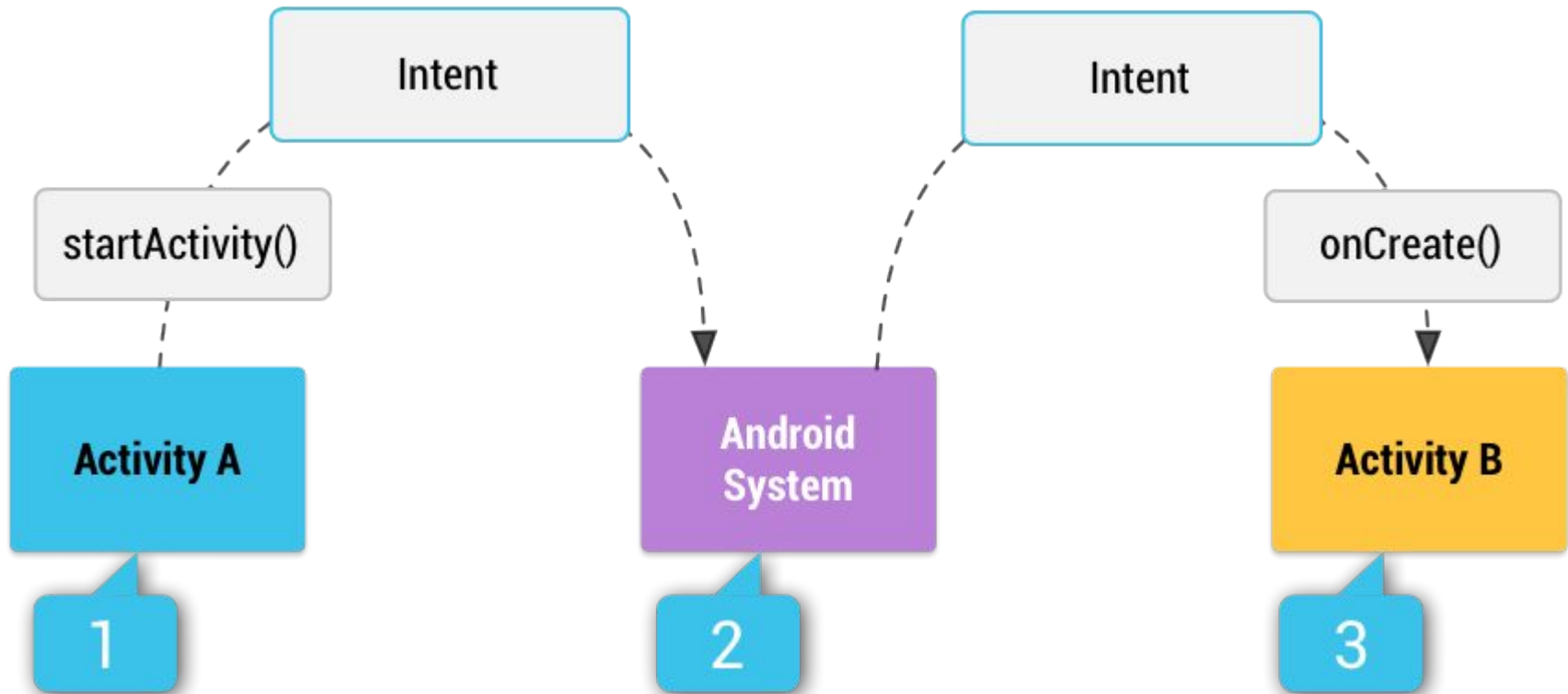
```
val url= "http://www.linti.unlp.edu.ar"
val webIntent = Intent().apply {
    action = Intent.ACTION_VIEW
    data = Uri.parse(url)
}
startActivity(webIntent )
```

Nuestra app
puede arrancar
mediante un
intent una
componente
browser en una
URL dada



Los intents implícitos **describen la acción** que se desea realizar y **proporcionan datos** para la realización de dicha acción.

Intents Implícitos



- (1) El Activity A crea un **Intent** que contiene la descripción de una acción y es enviado al método **startActivity()**.
- (2) El sistema Android busca un **Intent Filter** que coincida con el **Intent**.
- (3) Una vez encontrado, el sistema arranca el **Activity B** invocando al **onCreate()** y enviando el **Intent**.

Intents e IntentFilters

Las app **Android** tienen un estilo web. Esto hace referencia a cómo trabaja la plataforma Android y cómo los usuarios interactúan con los dispositivos móviles. Lo poderoso de Internet es que todo está en un “click”. Intents e IntentFilters proporcionan el paradigma de “click it”, constituyen un mecanismo innovador de navegación y activación.

Un **Intent** es una declaración de necesidad: está formado por múltiples piezas de información que describen la acción o servicio deseado. Ej. visualizar una página web, reproducir un video, enviar un mensaje de texto.

Un **IntentFilter** es una declaración de capacidad e interés en ofrecer asistencia a dichas necesidades. Se definen en el archivo **AndroidManifest.xml** con el tag **<intent-filter>**.

Mediante **Intents**, las aplicaciones pueden aprovechar la funcionalidad existente en otras aplicaciones solicitando código existente para manejar el **Intent** en lugar de escribir código desde cero.

Los **Intents** se utilizan para lanzar un **Activity**, un **Service**, un **BroadcastReceivers**.

Los **Intents** se resuelven dinámicamente, en ejecución.

Intents Implícitos - Ejemplos

Muestra los contactos del dispositivo al usuario:

```
val intent = Intent().apply {  
    action = Intent.ACTION_VIEW  
    data = Uri.parse("content://contacts/people")  
}  
startActivity(intent)
```

Acción

Datos

Crear una intención y pedir que sea tratada.

El intent es evaluado y pasado al manejador apropiado, en este caso el receptor podría ser la aplicación *built-in* **Contactos**.

Marca un número especificado, permite iniciar un llamado telefónico:

```
val intent = Intent().apply {  
    action = Intent.ACTION_VIEW  
    data = Uri.parse("tel: (+54) 12345789")  
}  
startActivity(intent)
```

Acción

Datos

Provee acceso editable al contacto identificado por 1:

```
val intent = Intent().apply {  
    action = Intent.ACTION_EDIT  
    data = Uri.parse("content://contacts/people/1")  
}  
startActivity(intent)
```

Acción

Datos

El atributo **acción** de un Intent es un verbo, por ej. VIEW, PICK, EDIT, DIAL, etc. La clase Intent declara un conjunto de acciones predefinidas (<http://developer.android.com/reference/android/content/Intent.html>).

Los desarrolladores pueden crear nuevas acciones.

Los **datos** de un Intent son expresados en forma de URI y pueden ser un registro de la bd de contactos, un sitio web, etc.

Intents e IntentFilters

Algunas **constantes estándares de Android** que indican **acciones** a realizar :

Constantes	Componente Destinataria	Acción
ACTION_MAIN	Activity	Arrancar el Activity cuando comienza la aplicación.
ACTION_EDIT	Activity	Mostrar datos al <i>usr</i> para editar.
ACTION_CALL	Activity	Iniciar una llamada telefónica.
ACTION_SYNC	Activity	Sincronizar datos de un servidor con los del dispositivo.
ACTION_PICK	Activity	Tomar y retornar datos.
ACTION_DIAL	Activity	Marcar el número de teléfono especificado.
ACTION_SENDTO	Activity	Enviar un mensaje a alguien.

Es posible definir acciones propias, en ese caso el paquete de nuestra aplicación se usará como prefijo. Por ejemplo: ejemplo.labo.MUESTRA_PUNTUACIONES.

Intents & IntentFilters

- Android instala un .apk, registra las componentes de la aplicación incluyendo los **intents filters**. Una vez registrados, el sistema puede asociar un pedido de **intent** con un Activity, un BroadcastReceiver o un Service. Para localizar el manejador apropiado, Android inspecciona los atributos **acciones**, **datos** y **categoría** del intent.
- Cada **IntentFilter** puede definir cero o más acciones y cero o más categorías. Si no se especifican acciones, coincidirá con cualquier **Intent**, en otro caso coincidirá con el Intent que tenga la misma **acción**. Un **IntentFilter** sin categorías coincidirá solamente con Intents sin categorías, en otro caso deberá tener al menos las que tiene el intent.

Intents & IntentFilters

```
val url= "http://www.linti.unlp.edu.ar"
val webIntent = Intent().apply {
    action = Intent.ACTION_VIEW
    data = Uri.parse(url)
}
startActivity(webIntent )
```

Intent

Una componente puede registrarse asimismo a través de un *intent filter* para realizar una acción específica con datos específicos.

El siguiente código registrará un *Activity* para un *Intent* que será disparado cuando un usuario desee abrir una página web:

```
<activity android:name=".BrowserActivitiy"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http"/>
    </intent-filter>
</activity>
```

Intent Filter

“Puedo ser iniciado, aunque la categoría del Intent que me arranca no tenga asignada ninguna categoría”



Intents & IntentFilters



La plataforma Android determina quién atenderá el intent, para ello usa los datos incluidos en el Intent.

Si varias componentes tienen registrado el mismo intent filter, el usuario decidirá qué componente lanzar.

En nuestro ejemplo, el usuario deberá elegir entre Chrome o el Browser personalizado.

Referencias

Videos:

<https://www.raywenderlich.com/android>

Guías para desarrolladores:

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

<http://www.vogella.com/tutorials/android.html>

Video:

La evolución de ART: <http://bit.ly/2fVEhGv>

Libros (Biblioteca):

Programming Android. 2nd ed. O'Reilly, 2012. Mednieks, Zigurd ; Dornin, Laird; Meike, G. Blake ; Nakamura, Masumi.