



Componentes de una aplicación Android



Anatomía de las aplicaciones

- Las **aplicaciones Android se construyen mediante bloques esenciales de componentes**, cada uno de los cuales existe como una entidad propia y desempeña un papel específico, **cada elemento es una pieza única que ayuda a definir el comportamiento general de la aplicación.**
- Es importante mencionar que **algunos de estos elementos son el punto de entrada para que los usuarios interactúen con la aplicación** y en muchos casos veremos que unos elementos dependen de otros.

Anatomía de las aplicaciones

- Consisten en uno o varios componentes definidos en el fichero de manifiesto de la aplicación.

- Tipos de componentes:

- **Activity**

- <https://developer.android.com/reference/android/app/Activity.html>

- **Fragment**

- <https://developer.android.com/guide/components/fragments.html>

- **Service**

- <https://developer.android.com/guide/components/services.html>

- **Broadcast receiver**

- <https://developer.android.com/reference/android/content/BroadcastReceiver.html>

- **Content provider**

- <https://developer.android.com/guide/topics/providers/content-providers.html>



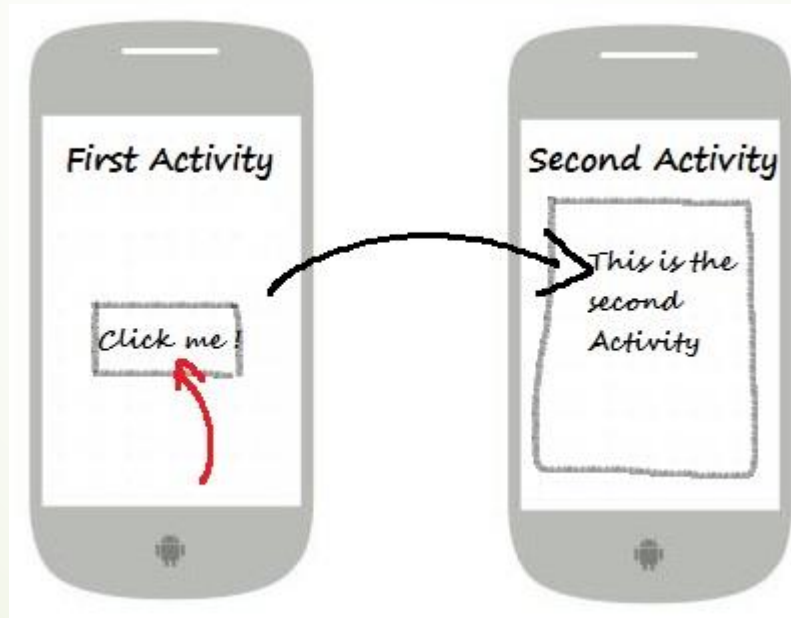
Anatomía de las aplicaciones

■ Actividades

- Cada Actividad (clases java que extienden a Activity) representa **una única pantalla de la aplicación y provee una interfaz de usuario**, es decir, una pantalla sobre la que el usuario puede interactuar. Por lo tanto, una aplicación con tres pantallas distintas, tendrá tres Actividades distintas e independientes.
- De este modo, se puede permitir a otras aplicaciones que inicien cualquiera de las Actividades de nuestra aplicación (siempre y cuando se dé acceso externo a las mismas desde nuestra aplicación).

Anatomía de las aplicaciones

► Actividades





Anatomía de las aplicaciones

➤ Fragmentos

- Un **fragment** podría definirse como una **porción de la interfaz de usuario que puede añadirse o eliminarse de una interfaz de forma independiente al resto de elementos de la actividad**, y que por supuesto puede reutilizarse en otras actividades.
- Nos va a permitir poder **dividir nuestra interfaz en varias porciones** de forma que podamos diseñar diversas configuraciones de pantalla, dependiendo de su tamaño y orientación, sin tener que duplicar código en ningún momento, sino tan sólo utilizando o no los distintos fragmentos para cada una de las posibles configuraciones.



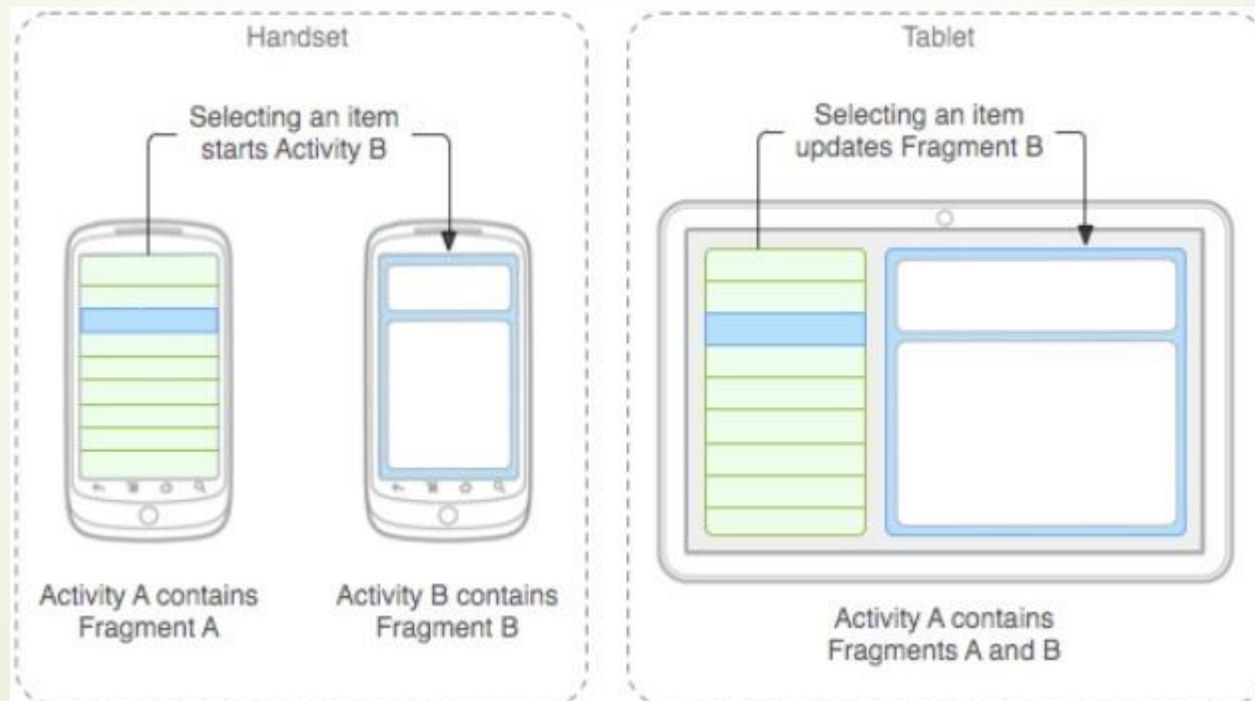
Anatomía de las aplicaciones

■ Fragmentos

- Un **fragmento** representa un comportamiento o una parte de la interfaz de usuario en una Actividad.
- Se pueden combinar múltiples fragmentos en una sola actividad para construir una IU multi-pane y reutilizar un fragmento en múltiples actividades.
- Es como un sección modular de una actividad, que **tiene su propio ciclo de vida, recibe sus propios eventos inputs, y los puede añadir o eliminar mientras la actividad se está ejecutando.**

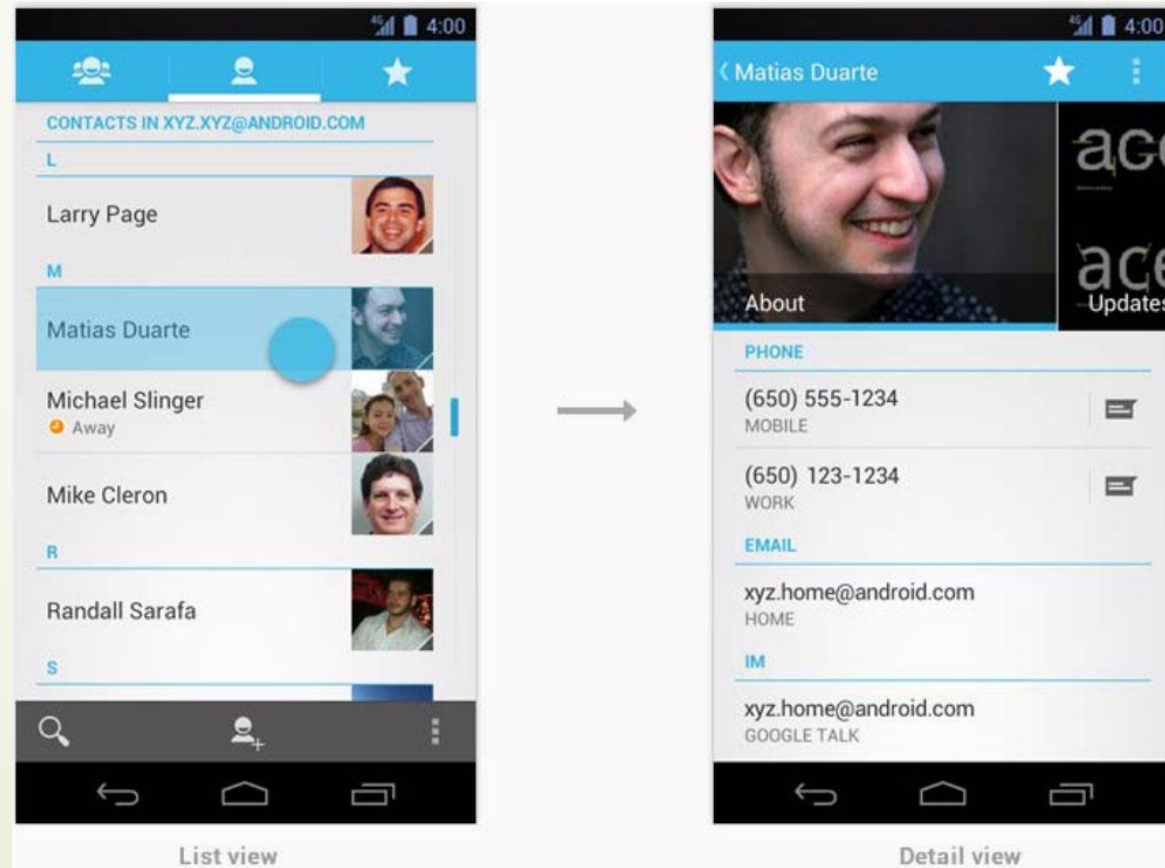
Anatomía de las aplicaciones

Fragmentos



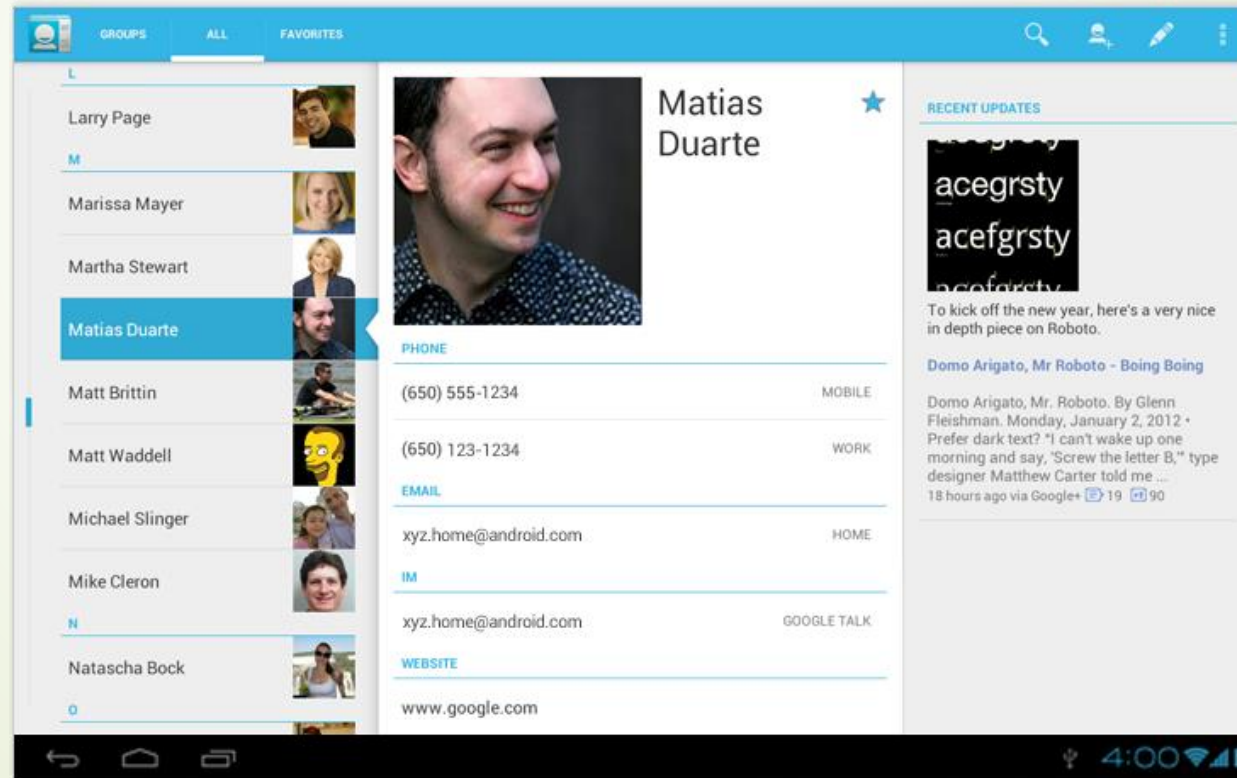
Anatomía de las aplicaciones

Fragmentos



Anatomía de las aplicaciones

Fragmentos





Anatomía de las aplicaciones

■ Intents (Intenciones)

- Un **intento** suele representar la **intención de realizar algo** como realizar una llamada de teléfono, visualizar una página web, etc.
- En muchas ocasiones los intentos no serán inicializados por la aplicación, sino por el sistema, por ejemplo, cuando pedimos visualizar una página web.
- En otras ocasiones será más interesante que la aplicación inicialice su propio Intento.
- 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 (que pasen cierto filtro).**

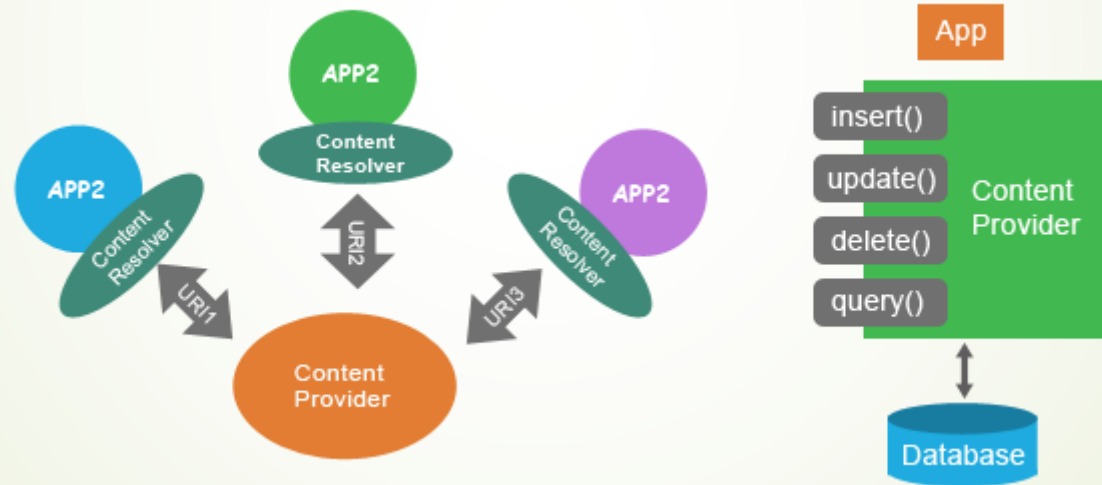


Anatomía de las aplicaciones

- **Content providers (proveedor de contenido)**
 - La **compartición de información entre teléfonos móviles** resulta un tema vital.
 - Android define **un mecanismo estándar para que las aplicaciones puedan compartir datos sin necesidad de comprometer la seguridad del sistema de ficheros.**
 - Con este mecanismo podremos acceder a datos de otras aplicaciones, como la lista de contactos, o proporcionar datos a otras aplicaciones.
 - Android proporciona un conjunto de content providers para los tipos de datos más comunes: audio, vídeo, imágenes, contactos...

Anatomía de las aplicaciones

➤ Content providers (proveedor de contenido)



Anatomía de las aplicaciones

- **Content providers (proveedor de contenido)**

- **Modelo de datos.**

- Provee datos tabulados en forma de tabla.

- Cada fila es un registro

- Cada columna proporciona datos de un tipo y con un significado concreto.

- Por ejemplo al solicitar la información relativa a los contactos existentes en el móvil, cabe esperar recibir información tabulada como sigue:

_ID	NUMBER	NUMBER_KEY	LABEL	NAME	TYPE
13	(425) 555 6677	425 555 6677	Kirkland office	Bully Pulpit	TYPE_WORK
44	(212) 555-1234	212 555 1234	NY apartment	Alan Vain	TYPE_HOME
45	(212) 555-6657	212 555 6657	Downtown office	Alan Vain	TYPE_MOBILE
53	201.555.4433	201 555 4433	Love Nest	Rex Cars	TYPE_HOME



Anatomía de las aplicaciones

- **Content providers (proveedor de contenido)**

- Los Proveedores de Contenido (subclases de `ContentProvider`) **gestionan conjuntos de datos que la aplicación comparte con otras aplicaciones. Los datos compartidos pueden almacenarse en distintos lugares** como, por ejemplo, en una base de datos SQLite, o en un archivo en el sistema.
- **Estos datos compartidos por el Proveedor de Contenido (Content Provider) podrá, ser consultados por otras aplicaciones, e incluso, ser modificados,** siempre y cuando el Proveedor de contenido lo permita (puede que no se permita siquiera el acceso a dicha información y que simplemente se utilice el Proveedor como un almacén privado de datos).



Anatomía de las aplicaciones

► Services (Servicios)

- Un Servicio (clase java que extiende a Service) **es un componente que se ejecuta en segundo plano con el objeto de realizar tareas de larga duración** de forma que no se bloquee la interacción del usuario con la Actividad que esté ejecutándose en ese momento.
- No necesitan interacción con el usuario.
- A diferencia de las Actividades, **un Servicio no provee una interfaz de usuario**. Los servicios pueden ser iniciados por otros componentes quienes, además, podrán interactuar con los mismos.
- Más prioridad que la Actividad estándar.

Anatomía de las aplicaciones

▀ Services (Servicios)

- ▀ Los servicios no poseen IGU.
- ▀ Aunque se ejecutan en segundo plano, como cualquier otro componente, se ejecutan en el hilo principal del proceso que los alberga.
 - ▀ Si realiza una tarea que haga un uso intensivo de recursos (como la reproducción de MP3) o que se bloquee (intercambios RSS o navegación internet) deben crearse hilos para el servicio.
- ▀ Deben aparecer declarados en el manifiesto de sus paquetes utilizando la etiqueta <service>
- ▀ Ejecución/Parada
 - ▀ Context.startService(...)/stopService(...)
 - ▀ Context.bindService(...)/unbidService(...)



Anatomía de las aplicaciones

- **Services (Servicios)**

- En Android disponemos de dos tipos de servicios:

- **Servicios locales:** se pueden utilizar por aplicaciones del mismo terminal

- **Servicios remotos:** Pueden ser utilizados desde otros terminales.



Anatomía de las aplicaciones

■ Receptores Broadcast

- Son unos componentes destinados a detectar y reaccionar ante determinados mensajes o eventos globales generados por el sistema continuamente (“batería baja”, “SMS recibido”, “Tarjeta SD insertada”, “GPS encendido”, “recepción de un mensaje o una llamada”...) o por otras aplicaciones (cualquier aplicación puede generar mensajes broadcast, no dirigidos a una aplicación concreta sino a cualquiera que quiera escucharlo).
- Estas clases **no tienen asociado un interfaz de usuario, pero sí que pueden generar notificaciones en la barra de estado para avisar al usuario.**



Anatomía de las aplicaciones

➤ Diferencia entre Intents y Broadcast.

- Lanzar a ejecución una actividad a través de un Intent es una operación que se efectúa en primer plano y de la que el usuario es consciente.
- Realizar el broadcasting de un Intent es una operación que se lleva a cabo en segundo plano y de la que el usuario no es consciente. Por ejemplo la puede realizar un servicio.



Anatomía de las aplicaciones

- **RECUERDA:** Es importante tener en cuenta que cada Evento se envía como un objeto Intent.

Anatomía de las aplicaciones

➤ Fichero manifiesto AndroidManifest.xml

➤ Fichero XML que debe existir en toda aplicación Android y nos indica:

- Componentes de la aplicación y relaciones entre ellos.
- Permisos que necesita la aplicación para su ejecución

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3    package="com.android.curso.helloworld">
4
5    <application
6      android:allowBackup="true"
7      android:icon="@mipmap/ic_launcher"
8      android:label="HelloWorld"
9      android:supportsRtl="true"
10     android:theme="@style/AppTheme">
11      <activity android:name=".MainActivity">
12        <intent-filter>
13          <action android:name="android.intent.action.MAIN" />
14
15          <category android:name="android.intent.category.LAUNCHER" />
16        </intent-filter>
17      </activity>
18    </application>
19
20  </manifest>
```


Anatomía de las aplicaciones

- Algunos elementos del manifiesto:
 - Nivel 1: elemento manifest:
 - Atributo package: nombre del paquete java que es la base de la aplicación
 - Nivel 2:
 - Elemento uses-permission: permisos que necesita la aplicación para funcionar correctamente.
 - Elemento permission: permisos de seguridad para limitar el acceso a componentes específicos de la aplicación
 - Elemento instrumentation: permite monitorizar la interacción de la aplicación con el sistema.
 - Elemento application: declaración de la aplicación:
 - Elemento activity (activities).
 - Elemento service (services).
 - Elemento receiver (broadcast receivers).
 - Elemento provider (content providers).



Actividades (Activity)


- Su función es la **creación del interfaz de usuario**.
- **Representan una pantalla de la aplicación.**
- Toda Actividad tiene una vista asociada (suele ser un layout)
- Una **aplicación estará formada por un conjunto de actividades independientes, aunque todas trabajando para un objeto común.**
- Toda actividad ha de ser una **subclase de Activity**.



Creación de nuevas actividades



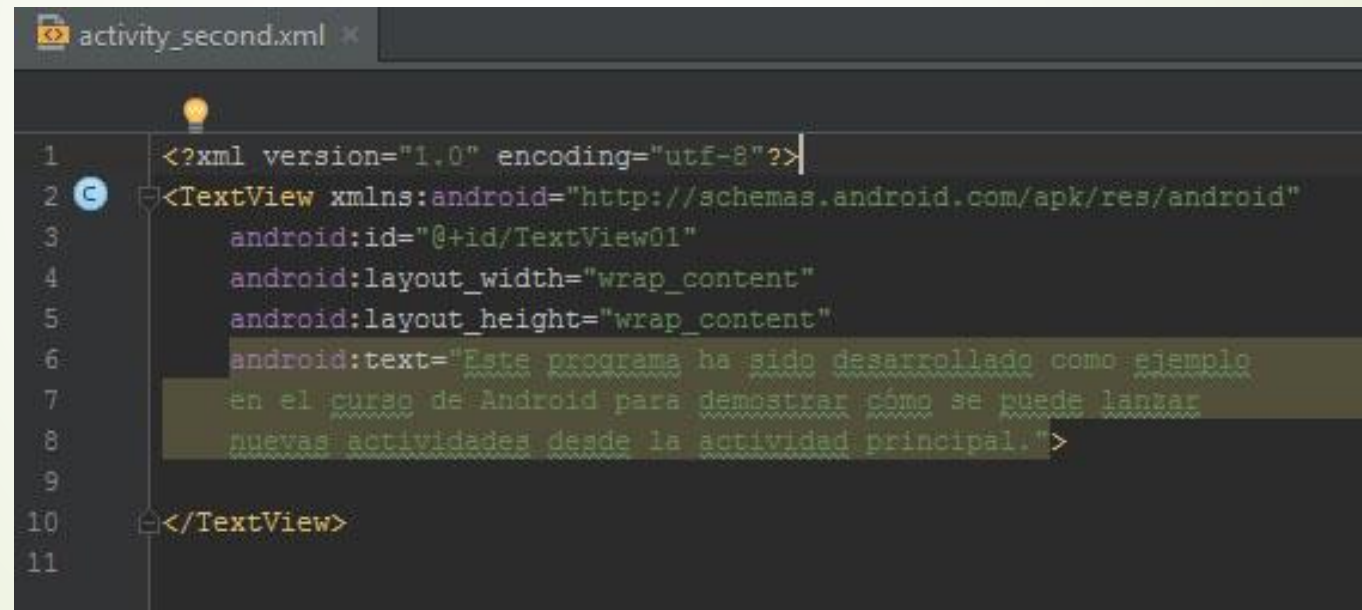
Pasos para añadir una nueva Actividad:

1. Crear un nuevo Layout para la actividad.
2. Crea una nueva clase descendiente de Activity.
 -  En esta clase se indicará el layout a visualizar.
3. Desde otra actividad lanza la nueva actividad.
4. Registra la nueva actividad en el AndroidManifest.xml

Creación de nuevas actividades

1. Creación nuevo Layout

- En la carpeta layout seleccione New/Layout resource file y creamos activity_second.xml



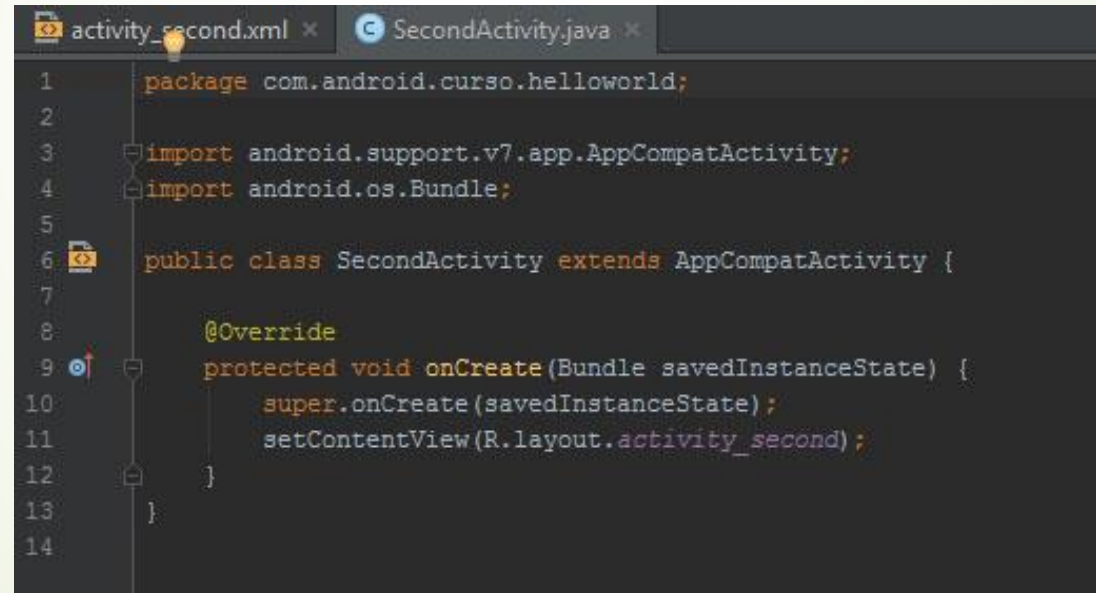
```
1 <?xml version="1.0" encoding="utf-8"?>
2 <TextView xmlns:android="http://schemas.android.com/apk/res/android"
3     android:id="@+id/TextView01"
4     android:layout_width="wrap_content"
5     android:layout_height="wrap_content"
6     android:text="Este programa ha sido desarrollado como ejemplo
7     en el curso de Android para demostrar cómo se puede lanzar
8     nuevas actividades desde la actividad principal.">
9
10 </TextView>
11
```

Realmente no es un Layout sino un TextView que visiona el contenido.

Creación de nuevas actividades

2. Crea una nueva clase

- ➡ Botón derecho sobre java/paquete del proyecto New/Java class y creamos SecondActivity.java

A screenshot of an IDE window showing the code for SecondActivity.java. The window has two tabs: 'activity_second.xml' and 'SecondActivity.java'. The code is as follows:

```
1 package com.android.curso.helloworld;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5
6 public class SecondActivity extends AppCompatActivity {
7
8     @Override
9     protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_second);
12     }
13 }
14
```

Con setContentView asociamos el layout creado anteriormente.



Creación de nuevas actividades

2. Crea una nueva clase

- En el método `onCreate()` se deberán inicializar los componentes esenciales de la actividad entre los que destaca el propio interfaz de usuario, layout, que se inicializará por medio del método `setContentView()`.
- A esa nueva actividad le asignamos un layout definido ya, gracias a la función `setContentView()`.

Creación de nuevas actividades

3. Registra la actividad en AndroidManifest.xml

- Todas las actividades han de registrarse en AndroidManifest.xml
- Puedes editarlo visualmente o por código.

```
<activity  
    android:name=".SecondActivity"  
    android:label="Second Activity"> </activity>
```

Label: si queremos que en la barra de arriba de la actividad aparezca un determinado nombre.



Creación de nuevas actividades

3. Registra la actividad en AndroidManifest.xml

- Cada una de las actividades de la aplicación deberá estar definida en el archivo de manifiesto de la misma.
- Este archivo, también XML, declara, entre otras cosas, las actividades, como elementos `<activity>` anidados en el elemento `<application>`.
- Entre los atributos del elemento `<activity>` destacan propiedades de la actividad tales como el nombre (de la clase de la actividad), o el icono que aparecerá encima de la actividad.



Creación de nuevas actividades

3. Registra la actividad en AndroidManifest.xml

- En general, **la actividad principal contendrá un filtro (intent filter)** que declara que la actividad responderá a la **acción principal** (“main”, es decir, la actividad principal de la aplicación) y que **estará ubicada en la categoría “launcher”**, es decir, que dicha actividad aparecerá en el menú principal del teléfono.

Creación de nuevas actividades

3. Registra la actividad en AndroidManifest.xml

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



Creación de nuevas actividades

3. Registra la actividad en AndroidManifest.xml

- **<action>** especifica que la actividad es el punto principal de entrada en la aplicación (aunque en general no será el único).
- **<category>** especifica que la actividad deberá mostrarse en la lista de aplicaciones del dispositivo (en el launcher).
- Para ver más sobre todas las acciones y categorías que nos brinda Android, pincha [aquí](#).
- Cualquier actividad que no se quiera poner a disposición de otras aplicaciones no tendrá ningún filtro de intents y sólo podrá ser invocada a través de un Intent explícito

Creación de nuevas actividades

3. Registra la actividad en AndroidManifest.xml

- Es decir, **existen sólo tres formas de iniciar una actividad:**
 - O bien invocándola a través de un Intent explícito.
 - O bien invocándola a través de un Intent implícito.
 - O bien a través de un Intent Filter que describa qué tipo de acción se quiere realizar.
 - En este caso, el sistema buscará entre todas las actividades de todas las aplicaciones aquellas que encajen, a través de los filtros de intents que tengan declarados, con lo descrito en el Intent.



Creación de nuevas actividades

3. Registra la actividad en AndroidManifest.xml

- Para iniciar una actividad, por lo tanto, se invocará al método **startActivity()** pasando como parámetro el objeto Intent creado.



Fragmentos en Android.

- **Ejemplo:** Supongamos una **aplicación de correo electrónico**, en la que por un lado debemos mostrar la lista de correos disponibles, con sus campos clásicos De y Asunto, y por otro lado debemos mostrar el contenido del correo seleccionado.
- En un **teléfono móvil lo habitual** será tener una primera actividad que muestre el listado de correos, y cuando el usuario seleccione uno de ellos navegar a una nueva actividad que muestre el contenido de dicho correo.
- Sin embargo, **en una Tablet puede existir espacio suficiente para tener ambas partes de la interfaz en la misma pantalla**, por ejemplo en una Tablet en posición horizontal podríamos tener una columna a la izquierda con el listado de correos y dedicar la zona derecha a mostrar el detalle del correo seleccionado, todo ello sin tener que cambiar de actividad.



Fragmentos en Android.

- Un **fragmento** debería ser un **componente modular y reutilizable** en la aplicación.
- Como **un fragmento define su propio layout y su propio comportamiento utilizando sus propios callbacks de ciclo de vida**, se puede incluir un fragmento en múltiples actividades.
- Esto es especialmente importante ya que permite adaptar el uso del usuario a diferentes tamaños de pantalla. Por ejemplo, se puede incluir múltiples fragmentos en una actividad sólo cuando el tamaño de la pantalla es suficientemente grande, y cuando no lo es, se lanzan diferentes actividades que utilizan diferentes fragmentos.



Fragmentos en Android.

- Para crear un fragmento, se debe crear una subclase de **Fragment** (o una subclase ya existente).
- La clase **Fragment** tiene código que se parece mucho a una Activity. Contiene métodos callback similares a una actividad, como el onCreate(), onStart(), onPause() y onStop().
- De hecho, si se está modificando una aplicación Android ya existente para que utilice fragmentos, simplemente hay que mover código de los métodos callback de la actividad a los respectivos métodos callback del fragmento.



Fragmentos en Android.

- Normalmente, se deberían implementar al menos los siguientes métodos del ciclo de vida:

- onCreate():

El sistema lo llama cuando crea un fragmento. Dentro de la implementación, se debería inicializar componentes esenciales del fragmento que se quiere retener cuando el fragmento está en pausa o parado, y después reanudado.

- onCreateView():

El sistema lo llama cuando es el momento de que el fragmento dibuje la IU por primera vez. Para dibujar una IU para el fragmento, hay que devolver un View desde este método que es la raíz del layout del fragmento. Se puede devolver null si el fragmento no suministra una IU.

- onPause():

El sistema llama a este método como primera indicación de que el usuario está abandonando el fragmento (esto no siempre significa que el fragmento esté siendo destruido) . Aquí es donde normalmente se deberían comitear/guardar los cambios que quieren permanecer más allá de la actual sesión de usuario (ya que el usuario puede no volver).



Fragmentos en Android.

- La mayoría de las aplicaciones deberían implementar al menos estos tres métodos para cada fragmento.
- También existen algunas subclases de las que se puede extender, en vez de la clase base **Fragment**:
 - `DialogFragment`
 - `ListFragment`
 - `PreferenceFragment`



Fragmentos en Android.

- Para **aportar un layout al fragmento**, se debe implementar el método callback **onCreateView()**, al que **llama el sistema Android cuando es el momento de que el fragmento dibuje su layout**.
- La implementación de **este método debe devolver un View que es la raíz del layout del fragmento**.
- Para devolver un layout desde **onCreateView()**, se puede inyectar desde un recurso del layout definido en el XML. Para ayudar con esto, **onCreateView()** suministra un objeto **LayoutInflater**.

Fragmentos en Android.

➡ Ejemplo:

```
public class ExampleFragment extends Fragment{  
    @Nullable  
    @Override  
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {  
        return inflater.inflate(R.layout.example_fragment, container, false);  
    }  
}
```




Fragmentos en Android.

- **Añadir un fragmento a la actividad.**
 - Normalmente, un fragmento aporta una porción de IU a la actividad que lo hospeda (host), la cual está insertada como parte de la jerarquía global de la actividad.
 - Existen dos maneras de añadir un fragmento al layout de la actividad:
 - Declarar el fragmento dentro del archivo del layout de la actividad.(RECOMENDADA)
 - Añadir el fragmento programáticamente a un ViewGroup existente.



Fragmentos en Android.

- **Declarar el fragmento dentro del archivo del layout de la actividad.**
 - En este caso, se puede especificar propiedades del layout para **el fragmento como si fuera un view.**
 - Como ejemplo, tenemos el archivo de un layout para una actividad con dos fragmentos.

Fragmentos en Android.

- Declarar el fragmento dentro del archivo del layout de la actividad.

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

    <fragment
        android:name="com.android.curso.helloworld.ArticleListFragment"
        android:id="@+id/list"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent"></fragment>

    <fragment
        android:name="com.android.curso.helloworld.ArticleReaderFragment"
        android:id="@+id/viewer"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent"></fragment>

</LinearLayout>
```



Fragmentos en Android.

- **Declarar el fragmento dentro del archivo del layout de la actividad.**
 - El atributo **android:name** en el **<fragment>** especifica el nombre de la clase Fragment a instanciar en el layout.
 - **IMPORTANTE:** Cuando el sistema crea este layout de la actividad, instancia cada fragmento especificado en el layout y llama al método `onCreateView()` para cada uno, para recuperar cada layout del fragmento. El sistema inserta el objeto de tipo View devuelto por el fragmento directamente (gracias al método `onCreateView()`) en el lugar del elemento **<fragment>**.



Fragmentos en Android.

- Declarar el fragmento dentro del archivo del layout de la actividad.
 - **Nota:** Cada fragmento necesita un identificador único que el sistema puede usar para restablecer el fragmento si la actividad es reanudada.
 - Existen tres maneras de suministrar un ID para un fragmento:
 - Suministrar el atributo `android:id` con un ID único.
 - Suministrar el atributo `android:tag` con un string único.
 - Si no se aporta ninguno de estos dos, el sistema utiliza el ID del container view.

Fragmentos en Android.

- Finalmente, la actividad que queramos que aloje todos los fragments y sea la principal, tiene que extender de `FragmentActivity` (esta extiende a su vez de `Activity`), y el layout anterior donde habían elementos fragment, sería el layout de este `FragmentActivity`.

```
public class MainActivity extends FragmentActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
}
```

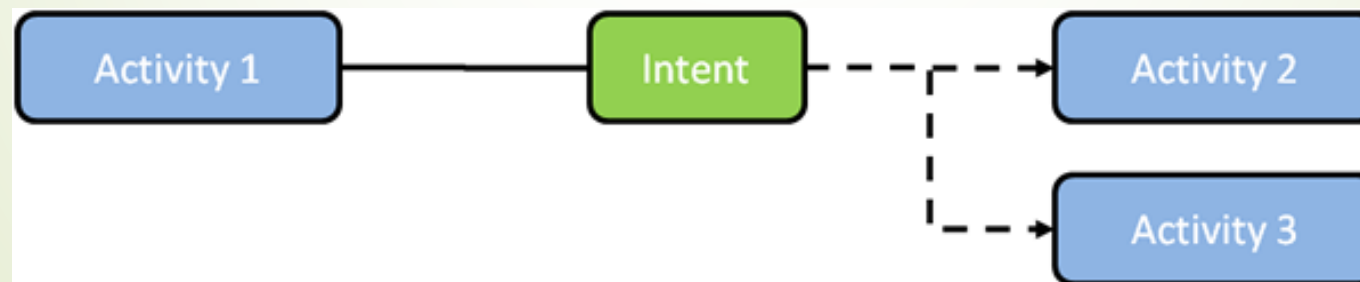



Fragmentos en Android.

- **Antes** de existir los fragments podríamos haber hecho esto implementando diferentes actividades con **diferentes layouts para cada configuración de pantalla**, pero esto nos habría obligado a **duplicar gran parte del código en cada actividad (código de la lógica de la aplicación)**.
- Tras la aparición de los **fragments**, **colocaríamos el listado de correos en un fragment y la vista de detalle en otro**, cada uno de ellos acompañados de su **lógica de negocio asociada**, y tan sólo nos quedaría **definir varios layout para cada configuración de pantalla** incluyendo (o no) cada uno de estos fragments. Aquí no duplicamos código de la lógica de las actividades, ya que las actividades solo alojarán en sus layouts fragmentos que ya tendrán dicha lógica. Sólo habrá que definir varios layouts para diferentes configuraciones de pantalla.

Las intenciones en Android

- Introducir el concepto de **intención** en Android.
- Diferenciar entre intenciones explícitas e implícitas
- Enumerara la información que incorpora una intención.
- Mostrar ejemplos de usos de intenciones





Las intenciones en Android

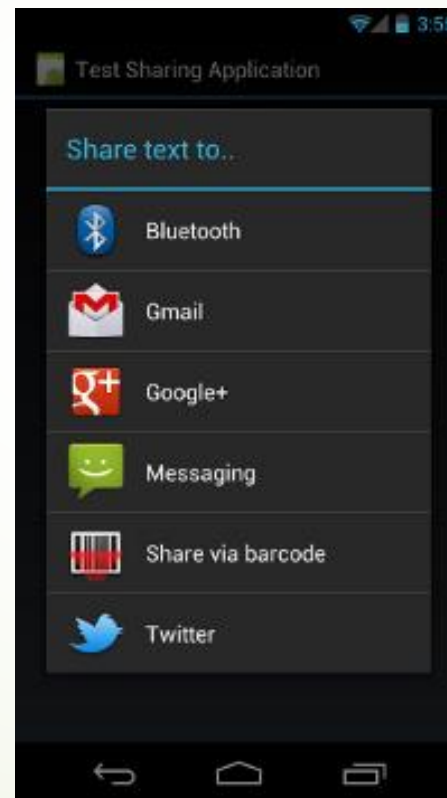
- Son intenciones del usuario de hacer algo.
- Representan la voluntad de realizar una acción o tarea.
- Los objetos de tipo [android.content.Intent](#) se utilizan para enviar mensajes asíncronos dentro de una aplicación o entre varias aplicaciones.
- Así, los Intents permiten enviar o recibir información desde y hacia otras actividades o **servicios**. También, permiten lanzar mensajes de tipo Broadcast (por ejemplo notificaciones en la barra de estado) para identificar cuando ciertos eventos han ocurrido.
- Los Intents son un concept poderoso ya que permiten a la comunicación entre cualquiera de los componentes de la aplicación instalados en el dispositivo.



Las intenciones en Android

- Un objeto **Intent** puede contener información para el componente receptor y de esta manera realizar determinada tarea.
- Por ejemplo, si nuestra aplicación llama a un browser a través de un Intent, este deberá contener una URL para que el browser (el componente receptor) pueda mostrarle esta página al usuario.
- Por otro lado, un **Intent** también puede contener información para el sistema Android para que este pueda determinar bajo algunas situaciones qué aplicación puede atender al solicitud de la aplicación. Esto ocurre con los **Intent Filter**.
 - Ejemplo: Cuando queremos reproducir una canción, y al darle para reproducir, nos aparece un listado de las aplicaciones que podemos usar para reproducir dicha canción.

Las intenciones en Android





Las intenciones en Android

- Utilizaremos **intenciones** cada vez que queramos
 - Lanzar una actividad (**startActivity()**)
 - Lanzar un servicio (**startService()**)
 - Lanzar un anuncio de tipo broadcast (**sendBroadcast()**)
 - Comunicarnos con un servicio (**bindService()**)



Las intenciones en Android

- **Las intenciones permiten indicar acciones genéricas**, como quiero mandar un mensaje, en cuyo caso se lanzará la aplicación registrada por defecto para este propósito.
- Aquí actuará el sistema Android de la siguiente manera:
 - Mira los intent-filter de todas las aplicaciones que tiene instaladas y las que pasen el filtro y sean adecuadas para la acción que se pide, las muestra en un diálogo como el siguiente:



Las intenciones en Android

- Tipos de intenciones.

- **Intenciones explícitas:** se indica exactamente el componente a lanzar. Su utilización típica es la de ir ejecutando los diferentes componentes internos de una aplicación. Por ejemplo, desde un actividad de nuestra aplicación lanzamos otra.
- **Intenciones implícitas:** pueden solicitar tareas abstractas, como “quiero tomar una foto” o “quiero enviar un mensaje”. Además **las intenciones se resuelven en tiempo de ejecución**, de forma que el sistema mirará cuantas aplicaciones han registrado la posibilidad de ejecutar es tipo de acción. Si encuentra varias, el sistema puede preguntar al usuario la actividad que prefiere utilizar.



Las intenciones en Android

- Tipos de intenciones.
 - Los **explícitos** son aquellos que nombran el componente que se necesita, ejemplo, la clase Java específica que se necesita para realizar alguna tarea.
 - Los intent **implícitos** son aquellos que le preguntan al sistema qué servicio o componente es el más adecuado para realizar la petición, es decir no especifica un componente en especial.



Las intenciones en Android

- Tipos de intenciones.

- En la **construcción de un Intent implícito** debemos especificar la acción que debe ser realizada y opcionalmente una URI que regularmente es usada por esta acción.
- Por ejemplo, podemos decirle al sistema que necesitamos ver (acción) una página web (URI). Cuando se inicie la llamada al Intent para que se realice esta acción, el sistema tratará de encontrar una aplicación que esté registrada para este evento, tal es el caso de los browsers.
- También es posible **agregar más información al Intent por medio de “extras”** que son parámetros que se les pasarán al Intent. Estos últimos son pares de key/valores.



Las intenciones en Android

- Partes de una intención:
 - **Nombre del componente:** Con intenciones explícitas, **identificamos el componente que queremos lanzar** con la intención. Hay que indicar el nombre de la clase
 - **Acción:** Con intenciones implícitas, hay que indicarle mediante una cadena de caracteres **la acción a realizar**. La clase Intent define una serie de constantes para acciones genéricas que son listadas a continuación. No obstante, además de estas podemos definir nuevas acciones.

Las intenciones en Android

➤ Algunas acciones genéricas (Para Intenciones implícitas)

Constante	Componente a lanzar	Acción
ACTION_CALL	Actividad	Inicializa una llamada de teléfono.
ACTION_EDIT	Actividad	Visualiza datos para que el usuario los edite
ACTION_MAIN	Actividad	Arranca como actividad principal de una tarea (sin datos de entrada y sin devolver datos)
ACTION_SYNC	Actividad	Sincronizar datos en un servidor con los datos de un dispositivo móvil
ACTION_BATTERY_LOW	Receptor de anuncios	Advertencia de batería baja
ACTION_HEADSET_PLUG	Receptor de anuncios	Los auriculares han sido conectados o desconectados

➤ Más acciones en:

<https://developer.android.com/reference/android/content/Intent.html>

Las intenciones en Android

Partes de una intención

- Categoría: Complementa a la acción con información adicional. Se definen las siguientes categorías:

Constante	Significado
CATEGORY_BROWSABLE	La actividad lanzada puede ser con seguridad invocada por el navegador para mostrar los datos referenciados por un enlace – por ejemplo, una imagen o un mensaje de correo electrónico.
CATEGORY_GADGET	La actividad puede ser embebida dentro de otra actividad que aloja gadgets.
CATEGORY_HOME	La actividad muestra la pantalla de inicio, la primera pantalla que ve el usuario cuando el dispositivo esta encendido o cuando la tecla HOME es presionada.
CATEGORY_LAUNCHER	La actividad puede ser la actividad inicial de una tarea y se muestra en el lanzador de aplicaciones de nivel superior.
CATEGORY_PREFERENCE	La actividad a lanzar es un panel de preferencias

Ej. ACTION_MAIN Y CATEGORY_LAUNCHER: primera actividad a lanzar



Las intenciones en Android

- ▀ Partes de una intención

- ▀ **Categoría**

- ▀ Se trata de **un String con información adicional sobre el tipo de componente que debería manejar el Intent lanzado.**
 - ▀ Se pueden almacenar tantas descripciones de categorías como se requiera, dentro del Intent.
 - ▀ Al igual que en el caso de las acciones, dentro de la clase Intent existen diversas constantes que definen diferentes categorías, ya nombradas antes.

Más información: <https://developer.android.com/reference/android/content/Intent.html>

Las intenciones en Android

▀ Partes de una intención

▀ Datos

- ▀ Los datos sobre los cuales se actuará se pasan en forma de URI y de tipo MIME.
- ▀ En función de las especificaciones de los tipos de datos, se adoptarán diferentes acciones.
 - ▀ Ejemplo:
 - ▀ Si la acción es ACTION_EDIT, el campo data contendrá la URI de un documento que será mostrado en modo edición.
 - ▀ En el caso de ACTION_CALL, los datos serán una URI tipo tel:XXXXXXX, con el número al cual llamar.
 - ▀ Si la acción es ACTION_VIEW y los datos son una URI tipo http:, la actividad receptora deberá descargar y mostrar los datos a los cuales se refiera la URI (sean una página web, un documento descargado, un video de youtube o cualquier otro tipo de archivo)



Las intenciones en Android

- ▶ Partes de una intención

- ▶ **Datos**

- ▶ Para establecer y para leer la URI y el tipo MIME, se utilizarán los métodos accesorios setData(Uri), setType(String), setDataAndType(Uri, String), getData y getType().
 - ▶ **URI**: identifica un recurso en internet
 - ▶ **MIME**: son los tipos de los recurso que hay en internet (text/html, image/gif,...)

Las intenciones en Android

► Partes de una intención

► MIME

- MIME es un acrónimo de extensiones multipropósito de correo de internet (Multipurpose Internet Mail Extensions).
- Se trata de un standard que especifica como debe un programa (inicialmente un programa de correo o un navegador web) transferir archivos multimedia (video, sonido, por extensión cualquier archivo que no esté codificado en US_ASCII).
- Con anterioridad al desarrollo de las extensiones MIME, cualquier archivo que no se limitase a texto ascii debía ser codificado a estos caracteres (uuencode uudecode).
- MIME adjunta un archivo de cabecera a cada archivo, especificando el tipo y el subtipo del contenido del archivo principal. Gracias esta información tanto el servidor como el navegador pueden manejar y presentar correctamente los datos.



Las intenciones en Android

- ▀ Partes de una intención

- ▀ **MIME**

- ▀ En el uso diario de internet estamos beneficiándonos (y a veces sufriendo) los MIME TYPES. Cada vez que solicitamos una página de internet se abre un diálogo entre nuestro navegador y el server. Nuestro navegador pide la página. El servidor, antes de enviarla, nos confirma que ese recurso existe, y el tipo de datos que contiene. Esto último, mediante referencia al tipo MIME al que corresponde. Este diálogo, oculto al usuario, es parte de las cabeceras HTTP, protocolo que se sigue en la web.



Las intenciones en Android

- ▶ Partes de una intención

- ▶ **URI**

- ▶ Un **Uniform Resource Identifier** o **URI** (Identificador uniforme de recurso) **es una cadena de caracteres corta que identifica inequívocamente un recurso** (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc). Normalmente estos recursos son accesibles en una red o sistema.

Las intenciones en Android

- **La clase Intent.** Depende qué tipo de Intent usemos usaremos un constructor u otro.

Public constructors

`Intent()`

Create an empty intent.

`Intent(Intent o)`

Copy constructor.

`Intent(String action)`

Create an intent with a given action.

`Intent(String action, Uri uri)`

Create an intent with a given action and for a given data url.

`Intent(Context packageContext, Class<?> cls)`

Create an intent for a specific component.

`Intent(String action, Uri uri, Context packageContext, Class<?> cls)`

Create an intent for a specific component with a specified action and data.

Intent Implícitos

Intent Explícitos

Las intenciones en Android

➤ Intenciones explícitas:

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

➤ Intenciones implícitas:

```
** Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.androidcurso.com/"));  
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("geo:41.653613,-0.877351"));  
Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:962849347"));  
Intent intent = new Intent("android.media.action.IMAGE_CAPTURE");
```

➤ Para arrancar la actividad:

```
startActivity(intent);
```

*** public Intent (Contexto en el que se lanza, clase a lanzar)

** El dispositivo intentará visualizar esa página web.

Último Intent: capturará una imagen a través de la cámara.

Finalmente, después de crear todo esto, hay que llamar a `startActivity(intent)` para que lance la actividad o la acción relacionada con el Intent.

Las intenciones en Android

➤ Intenciones implícitas:

```
Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:962849347"));
```

➤ A este Intent (llamar por teléfono) hay que darle el siguiente permiso en el manifiesto:

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

➤ Y a este otro intent (realizar una foto)

```
Intent intent = new Intent("android.media.action.IMAGE_CAPTURE");
```

➤ Hay que darle el siguiente permiso en el manifiesto:

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



Las intenciones en Android

- Intenciones implícitas:

- Las intenciones implícitas hay que verlas como la intención de lanzar alguna aplicación que está instalada en el dispositivo móvil. Si hay varias que pueden realizar dicha acción para ese dato, nos mostrará un diálogo para elegir con qué aplicación queremos realizar dicha acción. Si hay solo una aplicación, se ejecutará automáticamente dicha acción, y si no hay ninguna aplicación que pueda ejecutar dicha acción para esos datos que nos pasan, el dispositivo móvil nos lanzará un mensaje de error.
- Los datos son el parámetro URI. Si te fijas tiene un sufijo (mailto:, http://, tel:, geo: ,...) que nos indica el tipo de dato que es y con qué aplicación podrían verse. Hay muchos tipos diferentes.

Las intenciones en Android

- Para enviar un correo electrónico:

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_SUBJECT, "asunto");
intent.putExtra(Intent.EXTRA_TEXT, "texto del correo");
intent.putExtra(Intent.EXTRA_EMAIL, new String[] { "roberto@android.com" });

startActivity(intent);
```

- La API de Android, de la clase Intent dice:

Intent

putExtra(String name, int value)

Add extended data to the intent.

- Agregamos diferentes valores preestablecidos que está esperando esta acción. Luego a las constantes les damos un valor elegido por nosotros.



Las intenciones en Android

- ▶ Para enviar un correo electrónico: (otra forma)

```
Intent i = new Intent (Intent.ACTION_VIEW,  
Uri.parse("mailto:dirección@dominio.com"));  
startActivity(i);
```



Las intenciones en Android

- Obteniendo la información del Intent desde una actividad.
 - Para obtener la información del Intent (o mejor dicho el Intent en si) lanzado de una actividad que nos ha llevado a esta nueva actividad, utilizamos el método `getIntent()` de la clase `Activity`.
 - Si la actividad fue llamada a través de un intent implícito, podemos recibir la información y la URL desde el Intent utilizando los métodos `getAction()`, `getData()` y `getStringExtra()` (también puede ser utilizado para Intent explícitos).




Las intenciones en Android

- Obteniendo la información del Intent desde una actividad
 - Bundles
 - No son un elemento visible al usuario.
 - Es una clase que contiene valores en forma de <clave,valor>.
 - **Los utilizamos para pasar información**, por ejemplo, cuando rotamos el dispositivo Android (pasamos a modo apaisado), o para dar información a una Actividad que vamos a ejecutar.
 - A los Bundle se les puede ver como “maletas” que inicialmente están vacías, y mediante el método putExtra() o putTipoDato(), vamos añadiéndole información, y una vez relleno de información, se lo asignamos al Intent, el cual podemos considerarlo un pasajero con su maleta (bundle).
 - Para asignar un Bundle a un Intent se usa el método putExtras().




Comunicación entre actividades

- La actividad y la necesidad de comunicarse.
 - Las **actividades** en Android son las responsables de crear el interfaz de usuario.
 - Representar una pantalla de la aplicación.
 - Cada actividad se implementa en una clase independiente.
 - Incluso desde nuestra aplicación podemos llamar a actividades de otras aplicaciones.
 - Por lo tanto, **las distintas actividades no comparten sus variables y necesitan un mecanismo de comunicación.**
 - Este mecanismo está basado en **intenciones**.



Comunicación entre actividades

- Las intenciones
 - Representan la voluntad de realizar una acción.
 - Básicamente nos permiten lanzar una actividad o un servicio
 - Según como lo configuremos podemos conseguir que se lancen una u otra actividad.
 - Por ejemplo, si queremos enviar un mensaje, podremos crear una intención para que...
 - Una actividad en concreto realice el trabajo
 - El sistema escoja la actividad más adecuada.
 - La decisión la tome el usuario.



Comunicación entre actividades

- **Intenciones como mecanismo de comunicación:**

- En una intención podemos incorporar información para que sea recogida por la actividad lanzada en:
 - **Datos:** Datos con los que trabajaremos. Normalmente se indica la URI (tel:963572123, <http://www.google.com>,...) y el tipo MIME (text/xml, image/jpeg, audio/mp3, ...).
 - **Extras:** Información adicional. Está formada por un conjunto de pares variable/valor almacenados en un objeto de tipo Bundle.

Comunicación entre actividades

➤ Pasar datos a la actividad lanzada.

➤ En la actividad que lanza a otra:

```
Intent intent = new Intent(this, SecondActivity.class);  
intent.putExtra("usuario", "Pepito Perez");  
intent.putExtra("edad", 27);  
startActivity(intent);
```

➤ En la actividad lanzada:

```
Bundle extras = getIntent().getExtras();  
String nombre = extras.getString("usuario");  
int edad = extras.getInt("edad"); Nombre de la clave
```

getIntent() obtiene el Intent con el que hemos sido lanzados.

putExtra(clave,valor) añade información extra a la intención. Esta información se extrae con el método getExtras():

Comunicación entre actividades

- **Pasar datos a la actividad lanzada.**

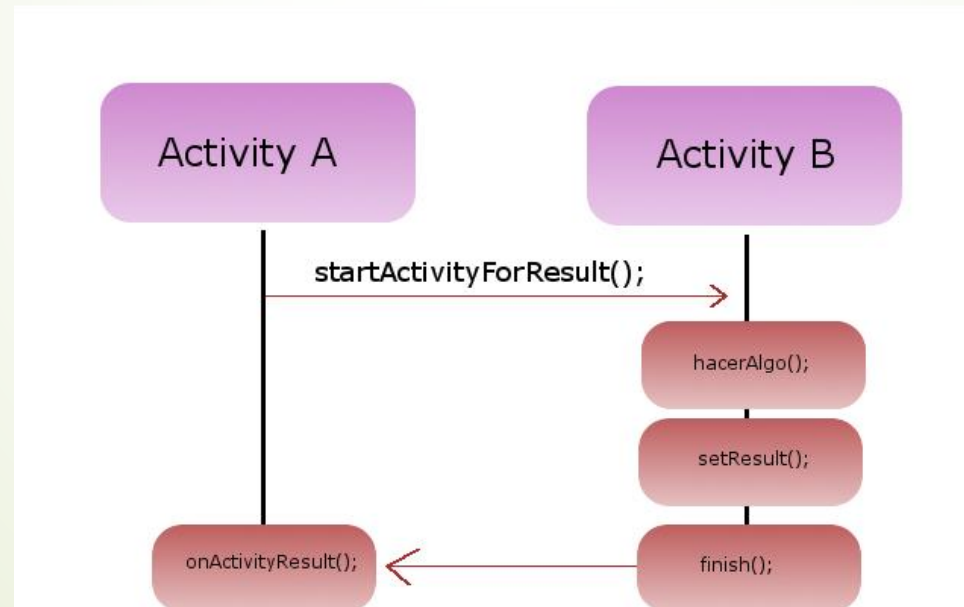
- Otra alternativa a la anterior es

```
Intent intent = new Intent(this, SecondActivity.class);
Bundle bundle = new Bundle();
bundle.putString("usuario", "Pepito Perez");
bundle.putInt("edad", 27);
intent.putExtras(bundle);
startActivity(intent);
```

Comunicación entre actividades

➤ Pasar datos a la actividad lanzada.

- `startActivityForResult()`: A diferencia de `startActivity`, `startActivityForResult` espera una que la actividad lanzada nos devuelva un determinado resultado.



Comunicación entre actividades

- Pasar datos a la actividad lanzada.

- startActivityForResult:

- En la actividad que lanza a otro:

```
Intent intent = new Intent(this, SecondActivity.class);
startActivityForResult(intent, REQUEST_CODE);

private static final int REQUEST_CODE = 1234;
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if(requestCode == REQUEST_CODE && resultCode == RESULT_OK){
        String res = data.getExtras().getString("resultado");
    }
}
```

- En la actividad lanzada

```
Intent intent = new Intent();
intent.putExtra("resultado", "valor");
setResult(RESULT_OK, intent);
finish();
```



Los Filtros de Intent

- El sistema **Android** determinará qué aplicaciones son las adecuadas para responder a un **Intent implícito** y en caso de que haya más de una aplicación que pueda responder a la **necesidad**, se lanzará al usuario un cuadro de diálogo en el cuál podrá elegir qué aplicación es la más conveniente para él.
- Gracias a los **Filtros** (Intent Filter), cuando se lance a ejecución una aplicación que por ejemplo reproduzca música (en el manifiesto de dicha aplicación, la actividad principal tendrá un filtro donde se indica que es para escuchar música) Android mostrará un diálogo para que podamos elegir entre varios reproductores para reproducir dicho audio, incluyendo éste que tiene el filtro, ya que cuando **Android ejecuta algo, busca en los filtros de todas las actividades, y aquellas que lo cumplan, te las sugiere como aplicación para reproducir dicho audio.**



Los Filtros de Intent

- Esta selección de aplicaciones se basa en el uso de **Intent Filters** que se definen en el archivo AndroidManifest.xml
- Para reaccionar a un determinado Intent implícito, un componente de la aplicación **debe estar registrado a través de un Intent Filter en el archivo AndroidManifest.xml** para este evento.
- En caso de que el componente no defina un Intent Filter únicamente podrá ser llamado a través de un Intent explícito.
- **Definen qué tipo de Intent lanzará la actividad en la que se definen.**
- **Se resuelven en tiempo de ejecución.**



Los Filtros de Intent

- A un Intent podemos asociarle una acción, unos datos y una categoría.
- Las actividades pueden declarar el tipo de acciones que pueden llevar a cabo y los **tipos de datos** que pueden gestionar.
 - Las **acciones** son cadenas de texto estándar que describen lo que la actividad puede hacer.
 - Por ejemplo, `android.intent.action.VIEW` es una acción que indica que la actividad puede mostrar datos al usuario. Esta acción viene predefinida en la clase Intent, pero es posible definir nuevas acciones para nuestras actividades.
 - La misma actividad puede declarar que el **tipo de datos** del que se ocupa es, por ejemplo, `"vnd.android.cursor.dir/person"`.
 - También puede declarar una **categoría**, que básicamente indica si la actividad va a ser lanzada desde el lanzamiento de aplicaciones, desde el menú de otra aplicación o directamente desde otra actividad.

Los Filtros de Intent

- ▶ En el AndroidManifest.xml quedaría algo así:

```
<intent-filter>
  <action android:name="android.intent.action.VIEW"/>
  <category android:name="android.intent.category.DEFAULT"/>
  <data android:mimeType="vnd.android.cursor.dir/person"/>
</intent-filter>
```

Así, para llamar implícitamente a una actividad a través de un Intent, en vez de asignar el nombre de la clase le asignamos una de las acciones que esta llevar a cabo, con el tipo de datos adecuado.



Los Filtros de Intent

- Cada **filtro** describe un conjunto de intents que el componente está dispuesto a recibir y gestionar.
- **Los filtros evitan la recepción de intents implícitos no deseados**, ya que solo son enviados al componente si pueden pasar a través de los filtros que dicho componente haya definido



Los Filtros de Intent

- Sin embargo, **los intents explícitos siempre son enviados a su objetivo**, sin considerar su contenido, ya que **no se consultan los filtros**.
- 



Los Filtros de Intent

- Para resolver un intent implícito, el sistema consultará los filtros de intents de los componentes registrados, y los comparará con la acción, datos (URI, tipo) y categoría definidos en el intent.
- En dicha consulta, el sistema no tiene en cuenta ni los flags ni los extras del intent.
- Para que el intent implícito sea finalmente entregado a un componente que haya declarado un filtro, deberá cumplir todas las condiciones declaradas en el mismo para poder pasar a través de dicho filtro (acción, datos y categoría).
- Si alguna de las áreas no las cumple, el intent no será capaz de pasar el filtro, por lo que el sistema no entregará dicho intent al componente que declaraba dicho filtro.

Los Filtros de Intent

■ Ejemplo:

```
<intent-filter>
  <action android:name="android.intent.action.VIEW"/>
  <action android:name="android.intent.action.EDIT"/>
  <action android:name="android.intent.action.PICK"/>
  <category android:name="android.intent.category.DEFAULT"/>
  <data android:mimeType="vnd.android.cursor.dir/vnd.google.note"/>
</intent-filter>
```

- Este filtro nos indica que la actividad está disponible para ver, editar o seleccionar elementos del tipo 'vnd.android.cursor.dir/vnd.google.note'.

Los Filtros de Intent

■ Ejemplo:

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

- Este filtro nos indica que la actividad que esta es la actividad principal de la aplicación (MAIN), y que aparecerá en el menú de aplicaciones del sistema (LAUNCHER).

Los Filtros de Intent

■ Coincidencia de intents

- Los intents son contrastados contra los diversos filtros de la aplicación no solo para descubrir qué componentes pueden ser activados sino también para extraer información del tipo de componentes instalados en el dispositivo.
- Por ejemplo, para obtener la lista de aplicaciones que se mostrarán en el launcher, el sistema busca todas las actividades que especifiquen el filtros:

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



Los Filtros de Intent

- Coincidencia de intents
 - Del mismo modo, el sistema “descubre” la pantalla de inicio, buscando la actividad que contenga un filtro con “android.intent.category.HOME”



Los Filtros de Intent

- Si un intent cumple más de un filtro de más de una actividad (es decir, cualquier aplicación del móvil) o servicio, el sistema mostrará un listado al usuario con todos los componente capaces de recibir el Intent.
- Si el Intent implícito no cumple ningún filtro de ningún componente, se lanzará una excepción.



Ciclo de vida de una aplicación Android

- La gestión del ciclo de vida de las actividades es crucial al desarrollar una buena aplicación.
- Para ello, en cada actividad se deberán implementar los **métodos callback que serán invocados cuando la actividad cambien de estado**.
- **El sistema operativo es el encargado de pausar, parar o destruir nuestra aplicación según las necesidades de recursos del dispositivo.**
- Nosotros como desarrolladores debemos aprender a controlar todos estos eventos para hacer nuestras aplicaciones robustas y mejorar el rendimiento de los teléfonos.
- El **ciclo de vida** de una actividad representa cada una de las pantallas de nuestra aplicación



Ciclo de vida de una aplicación Android

- Cada aplicación Android se ejecuta en su propio proceso Linux y posee su propia instancia de la VM Dalvik.
- Cuando una repetición necesita la intervención de un componente de la aplicación:
 - Android se asegura de que el proceso asociado a la aplicación esté en ejecución, arrancándolo en caso de necesidad.
 - También verifica que la instancia adecuada del componente esté disponible y si no lo está, lo crea.
- El proceso Linux que encapsula a toda aplicación se crea cuando el código de la misma debe ejecutarse y permanece en ejecución hasta que:
 - No se necesita más
 - El sistema reclama los recursos ocupados para otras aplicaciones



Ciclo de vida de una aplicación Android

- El **ciclo de vida** de las aplicaciones Android **no se controla desde las aplicaciones, sino desde la plataforma.**
- En base a:
 - Las partes de la aplicación que el sistema sabe que están en ejecución.
 - La importancia de dichas partes para el usuario.
 - La cantidad de memoria disponible en el sistema.

Ciclo de vida de una aplicación Android

- El ciclo de vida de los componentes tiene:
 - Un inicio, cuando Android los instancia en respuesta a un Intent.
 - Un fin, cuando su instancia se destruye.
 - Entretanto, su vida transita entre los estados activo e inactivo, o en el caso de las Activities, el de visible e invisible para el usuario

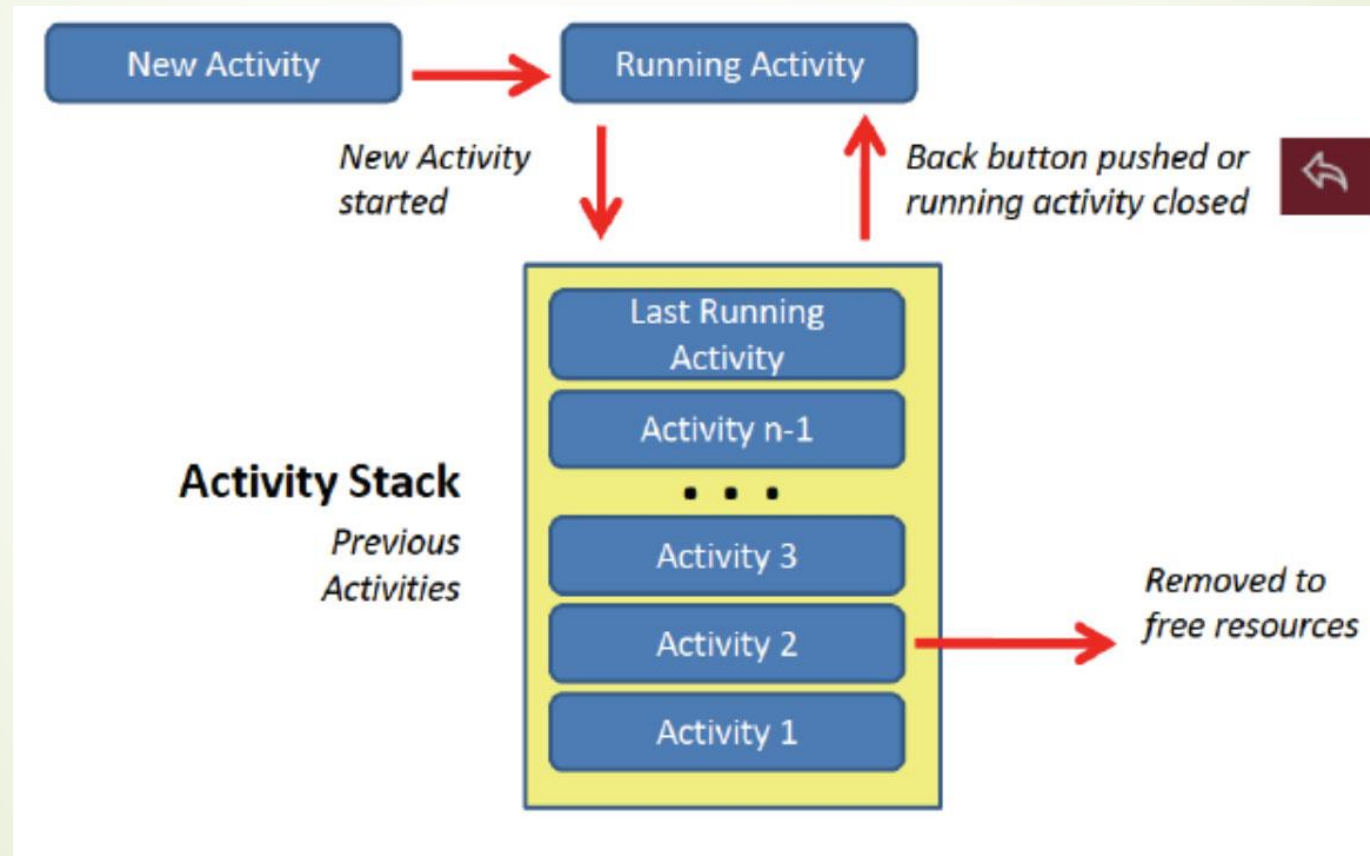




Ciclo de vida de una aplicación Android

- Las **Activities** se gestionan a través de una **pila**.
- Cuando una nueva actividad se lanza a ejecución, pasa a ocupar la cima de la pila.
- La cima de la pila anterior pasa a segundo plano hasta que la nueva actividad termine.
- Cuando el usuario pulsa el botón de vuelta atrás (back button), la siguiente actividad en la pila pasa a ocupar la cima y se activa (volviendo a su ejecución en primer plano)

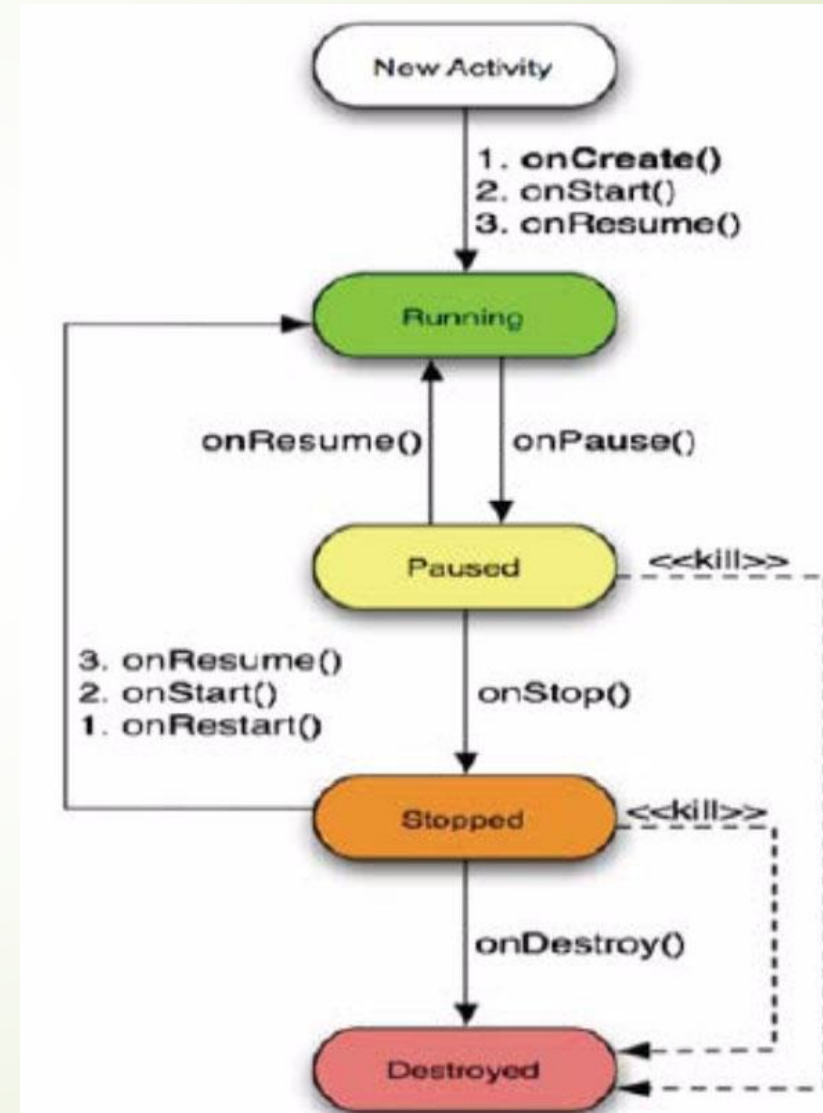
Ciclo de vida de una aplicación Android



Ciclo de vida de una aplicación Android

➤ Básicamente **3 estados**:

- En ejecución
- En pausa
- Parada



Ciclo de vida de una aplicación Android

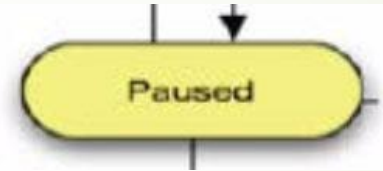
- Estado "Running"



- La Activity se encuentra en este estado **cuando se ejecuta en primer plano** (se encuentra en la cima de la pila).
- La actividad interactúa entonces con el usuario

Ciclo de vida de una aplicación Android

➤ Estado "Paused"



- La **Activity** está visible pero ya no se encuentra en la cima de la pila, con lo que no interactúa con el usuario.
- Está en un **segundo plano**, visible pero parcialmente oculta por otra actividad que ha obtenido el foco.
- La actividad **sigue viva**, manteniéndose su información en memoria, y sigue unida al gestor de ventanas, pero podría ser finalizada en casos de necesidad extrema de memoria por parte del sistema.
- La actividad que se encuentra en la pila es transparente o no cubre toda la pantalla. Si se cubre toda la pantalla está en estado parado.

Ciclo de vida de una aplicación Android

➤ Estado "Stopped"



- La Activity está completamente **oculta** por otra.
- **Mantiene su estado**, es decir, sigue manteniéndose en memoria.
- Ya no está visible y **previsiblemente será matada** por el sistema cuando se necesite algo de memoria.



Ciclo de vida de una aplicación Android

- Una vez pausada o parada la actividad, el sistema podrá eliminarla de la memoria bien
 - Invocando a `finish()`
 - O bien matando directamente su proceso.
- En cualquiera de los dos casos, cuando la actividad se vuelva a iniciar, deberá ser creada de nuevo (invocando a su método callback `onCreate()`)
- Nota: No se debe invocar explícitamente a los métodos `finish()` o `finishActivity()` ya que se pueden producir comportamientos extraños. Sólo se utilizarán en el caso de que no se desee que el usuario vuelva a la misma instancia de la actividad, es decir, volver a la pantalla de dicha actividad.

Ciclo de vida de una aplicación Android

- ▶ Eventos entre estados. Cada uno de los métodos:
 - ▶ **onCreate():**
 - ▶ Método invocado al crearse la actividad por primera vez.
 - ▶ Aquí es donde se inicializa la parte estática de la actividad (layout, datos...)
 - ▶ Puede recibir el estado previo de la actividad si se grabó, vía Bundle.
 - ▶ Actividad no visible aún.
 - ▶ No se puede eliminar la actividad después de este método
 - ▶ **onRestart():**
 - ▶ Método invocado después de haberse parado la actividad (oculta).
 - ▶ Actividad no visible aún.
 - ▶ No se puede eliminar la actividad después de este método.

Ciclo de vida de una aplicación Android

➤ Eventos entre estados. Cada uno de los métodos:

➤ **onStart():**

- Se ejecuta cuando la Activity se está mostrando apenas en la pantalla del dispositivo del usuario.
- Método invocado justo antes de que la actividad se haga visible.
- No se puede eliminar la actividad después de este método.

➤ **onResume():**

- Se ejecuta una vez que la Activity ha terminado de cargarse en el dispositivo y el usuario empieza a interactuar con la aplicación.
- Cuando el usuario ha terminado de utilizarla es cuando se llama al método onPause().
- Método invocado justo antes de que la actividad interactúe con el usuario.
- La actividad es la primera en la pila de actividades.
- Actividad visible.
- No se puede eliminar la actividad después de este método.
- Siempre le sigue onPause().
- Se debería usar para inicializar componentes que hemos liberado en el método onPause()

Ciclo de vida de una aplicación Android

- Eventos entre estados. Cada uno de los métodos:

- **onPause():**

- Método invocado justo antes de que otra actividad se visualice.
 - Actividad visible.
 - Se deberán liberar recursos y grabar datos rápidamente.
 - La siguiente actividad no se iniciará hasta que este método finalice.
 - Sí se puede eliminar la actividad después de este método.
 - Después de esta llamada puede venir un `onResume()` si la Activity que haya ejecutado el `onPause()` vuelve a aparecer en primer plan o un `onStop` si se hace invisible para el usuario.
 - Ejemplo: Nuestra aplicación usa la cámara, en el método `onPause()` es bueno para liberar la cámara, ya que no la vamos a usar una vez usada ya, y otras aplicaciones pueden necesitar usarla. También para guardar opciones que hemos seleccionado en la actividad de opciones.

Ciclo de vida de una aplicación Android

- Eventos entre estados. Cada uno de los métodos:

- **onStop():**

- Método invocado cuando la actividad ya no es visible.
 - Sí se puede eliminar la actividad después de este método.
 - Si vemos el diagrama, después de que se ha ejecutado este método nos quedan tres opciones: ejecutar el `onRestart()` para que la Activity vuelva a aparecer en primer plano, que el sistema elimine este proceso porque otros procesos requieran memoria o ejecutar el `onDestroy()` para destruir la aplicación.

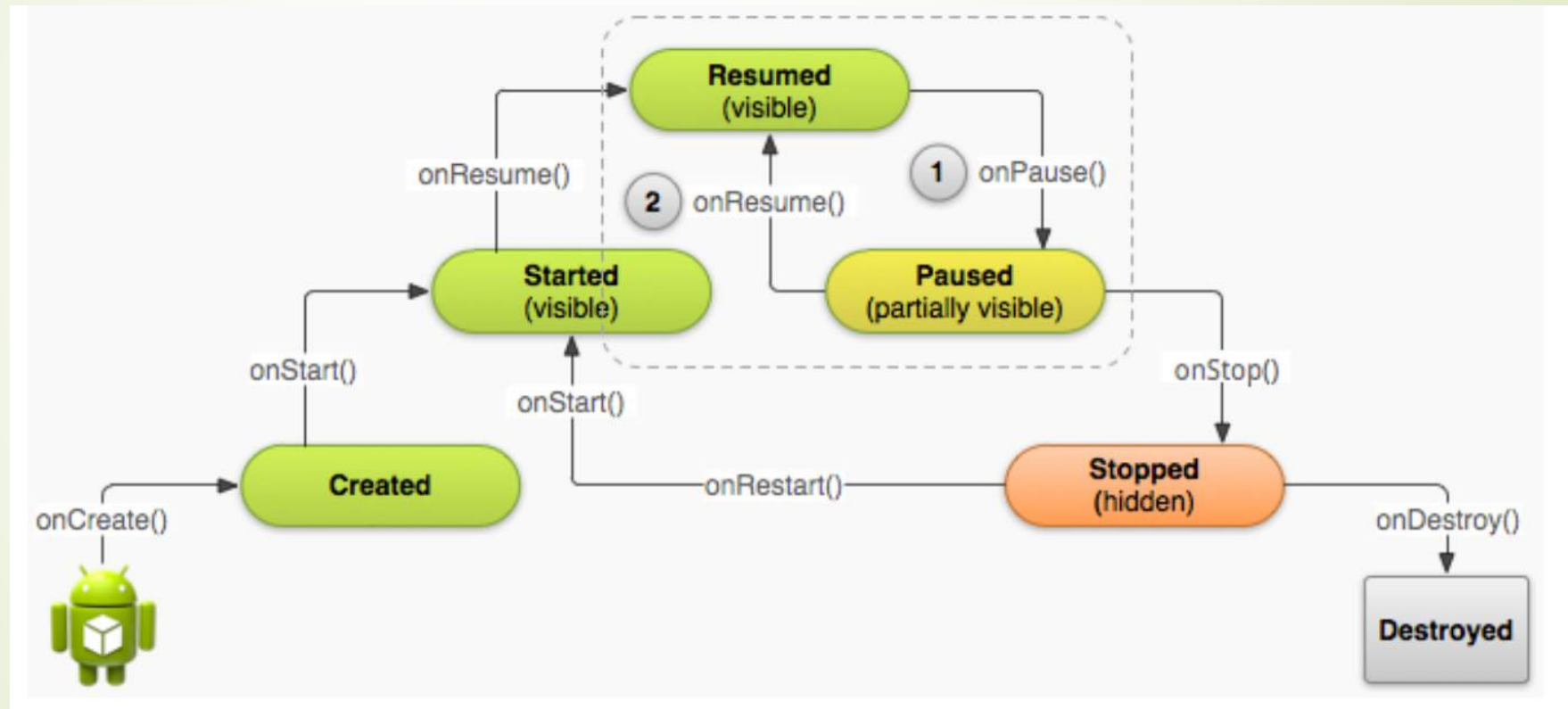
- **onDestroy():**

- Esta es la llamada final de la Activity, después de ésta, es totalmente destruida.
 - Esto pasa por los requerimientos de memoria que tenga el sistema o porque de manera explícita el usuario manda a llamar este método (`finish()`).
 - Si quisiéramos volver a ejecutar la Activity se arrancaría u nuevo ciclo de vida.

Ciclo de vida de una aplicación Android

- ▶ Eventos entre estados. Cada uno de los métodos:
 - ▶ Una actividad puede mantenerse en tres estados:
 - ▶ **Resumed**: La actividad está en el primer plano de la pantalla y el usuario la está utilizando. Este estado también se denomina “running”.
 - ▶ **Paused**: La actividad es visible, pero hay otra actividad en primer plano que tiene el foco. La actividad pausada sigue “viva” ya que se mantiene en memoria y conserva toda la información de estado, si bien el sistema operativo puede eliminarla en caso de memoria disponible muy baja.
 - ▶ **Stopped**: La actividad se oculta completamente por una nueva actividad. Una actividad detenida también se mantiene en memoria con toda la información de estado. Sin embargo, el usuario ya no la ve visible y el sistema operativo puede eliminarla cuando se necesita memoria para otra tarea.

Ciclo de vida de una aplicación Android



Ciclo de vida de una aplicación Android

- Eventos entre estados. Cada uno de los métodos:
 - Es importante recordar que, **cuando se implementen todos o algunos de estos métodos, será necesario invocar a la implementación del correspondiente método de la superclase (`super.onMetodo()`), por ejemplo `super.onCreate()` para que el sistema pueda gestionar correctamente el ciclo de vida.**
 - Todas nuestras operaciones se deben poner después de la llamada al método de la clase superior **super**.

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



Ciclo de vida de una aplicación Android

- ▶ Eventos entre estados

- ▶ Todos los métodos de transición pueden sobrescribirse para implementar las acciones pertinentes.

- ▶ SE DEBE sobrescribir siempre el método `onCreate()` para inicializar correctamente la Actividad la primera vez

- ▶ ALTAMENTE RECOMENDABLE que se sobrescriba el método `onPause()` para asegurar que los cambios en los datos se almacena correctamente y se gestiona correctamente el fin de la interacción con el usuario.

Ciclo de vida de una aplicación Android

➤ Coordinación de actividades

- Cuando una actividad invoca a otra en el mismo hilo, sus ciclos de vida se solapan durante cierto tiempo.
- Mientras que la primera actividad se pausará y puede que se pare (si se hace completamente invisible), la segunda actividad será creada.
- La secuencia de transiciones de estados es el siguiente:



Ciclo de vida de una aplicación Android

➤ RESUMEN

Método	Descripción	Kill	Siguiente método
onCreate()	Se invoca cuando la Actividad se crea por primera vez. Aquí es donde se reserva la memoria necesaria, se crea la interfaz de usuario, se recupera el estado de sesión anterior, etc.	No	onStart()
onRestart()	Se invoca cuando la Actividad está parada, justo antes de iniciarse de nuevo	No	onStart()
onStart	Se invoca justo antes de que el usuario pueda ver la Actividad. A continuación, se puede invocar el método onResume() si la Actividad vuelve al primer plano o onStop() si se oculta	No	onResume() o onStop()

Ciclo de vida de una aplicación Android

► RESUMEN

Método	Descripción	Kill	Siguiente método
onResume()	Se invoca justo antes de que el usuario comience a interactuar con la Actividad.	No	onPause()
onPause()	<p>Se invoca cuando el sistema está a punto de comenzar otra Actividad. Es recomendable usar este método para confirmar con el usuario si quiere guardar los cambios, desactivar animaciones y cualquier código que consuma CPU, etc. Estas sentencias deben ser muy rápidas porque la nueva Actividad no se inicia hasta que finaliza este método.</p> <p>A continuación, se puede invocar el método onResume si la Actividad vuelve al primer plano o onStop() si se oculta.</p>	Sí	onResume() O onStop()

Ciclo de vida de una aplicación Android

RESUMEN

Método	Descripción	Kill	Siguiente método
onStop()	<p>Se invoca cuando la actividad ya no es visible al usuario. Esto puede ocurrir porque se vaya a destruir la Actividad o porque otra Actividad (existente o nueva) se ha reanudado y vuelve al primer plano.</p> <p>A continuación, se puede invocar el método <code>onRestart()</code> si la Actividad vuelve al primer plano o <code>onDestroy()</code> si se destruye.</p>	Sí	<code>onRestart()</code> O <code>onDestroy()</code>
onDestroy()	<p>Se invoca antes de que se destruya una Actividad, se trata del último método.</p> <p>El sistema operativo puede invocar este método porque el usuario decide finalizar la aplicación (método <code>finish()</code>) o porque es necesaria memoria libre. Se puede distinguir entre estos dos escenarios con el método <code>isFinishing()</code></p>	Sí	ninguno



Ciclo de vida de una aplicación Android

➤ RESUMEN

- La columna “kill” de esta tabla indica si el sistema puede matar (kill) el proceso de la Actividad cuando finaliza la ejecución del método sin ejecutar ninguna sentencia más de la Actividad. Hay tres métodos así: onPause(), onStop() y onDestroy().
- Los métodos que se han marcado con “No” en la columna “kill” están, a priori, protegidos. El sistema operativo sólo los “mata” en una situación de inestabilidad con falta de memoria.



Ciclo de vida de una aplicación Android

➤ RESUMEN

- onPause() es el método que se ejecuta siempre en caso de que el sistema operativo mate una Actividad.
- Sin embargo, no es posible asegurar que el sistema invoque los métodos onStop() y onDestroy() porque se haya ejecutado onPause() y la Actividad haya acabado.
- Por lo tanto, **se debe utilizar el método onPause() para guardar los datos importantes y persistente de la Actividad.**
- No obstante, **hay que ser selectivo sobre qué información se debe guardarse durante onPause(),** ya que este método bloquea el inicio de una nueva Actividad y el usuario podría notar que el teléfono se enlentece.