

Activity:

“Ciclo de Vida”
“Cómo usarlas”

¿Qué es una Activity? (I)

- ▶ Es un **componente** de la aplicación que provee una **pantalla gráfica para interactuar con el usuario**.
- ▶ Cada actividad se **compone** de:
 - ❏ su **parte gráfica** (*layout-xml*) y
 - ❏ su **parte de lógica** (*JAVA*)
- ▶ **Cada activity tiene una ventana** sobre la que dibuja **su interfaz de usuario**.
 - La ventana **normalmente** llena la ***pantalla entera***, *pero puede ser más pequeña y “flotar” por encima de las otras ventanas*.






¿Qué es una Activity? (II)

- ❏ La **interfaz de usuario** está provista de una **jerarquía de elementos llamados vistas (View)**, **derivados** todos de la **clase View**.
 - ❏ Y por **ViewGroup** o contenedores (layouts)
- ❏ Cada **View** representa un **área rectangular dentro de la ventana (Window) de la activity**.
- ❏ Las View pueden responder a la **interacción del usuario**.
- ❏ También se les llama **“Widgets”** a las vistas.

¿Qué es una Activity? (III)

- ▶ Cada **activity** debe **hacer una cosa en concreto**:
enviar un email, mostrar una pantalla de login,...
- ▶ Una aplicación está formada por una o más “activities” que están relacionadas entre sí.
- ▶ Normalmente una de ellas será la **actividad principal**, *que es la que se presenta al lanzar la aplicación.*
- ▶ **Por ejemplo**: Una aplicación de email tendría:
 - 1 actividad para mostrar correos nuevos
 - 1 actividad para mostrar detalle del correo
 - 1 actividad para escribir un nuevo correo
 - 1 actividad para las preferencias de la aplicación

¿Qué es una Activity? (IV)

- ▶ Una activity puede lanzar otra nueva activity:
 -  de su misma aplicación.
 -  o de otra aplicación.
- ▶ Las actividades tienen 3 diferentes estados:
 -  *Ejecutándose*,
 -  *pausada (paused)*,
 -  *parada (stopped)*,...

¿Qué es una Activity? (V)

▶ Cuando una activity arranca:

- ❏ la anterior activity es “stopped”
- ❏ el sistema guarda la anterior activity en una pila (“back stack”) de activities.
- ❏ la nueva activity es colocada en el top de esa pila “back stack” y toma el foco.
- ❏ “back stack” es una pila LIFO (Last in, First Out).

▶ Cuando el usuario pulsa el botón volver atrás (BACK):

- ❏ la actual activity es “sacada de la pila” y destruida.
- ❏ La activity anterior (*que estaba en esa pila*) rearranca.

Métodos CALLBACK del ciclo de vida de una Activity

Gestión de las Activities por Android:

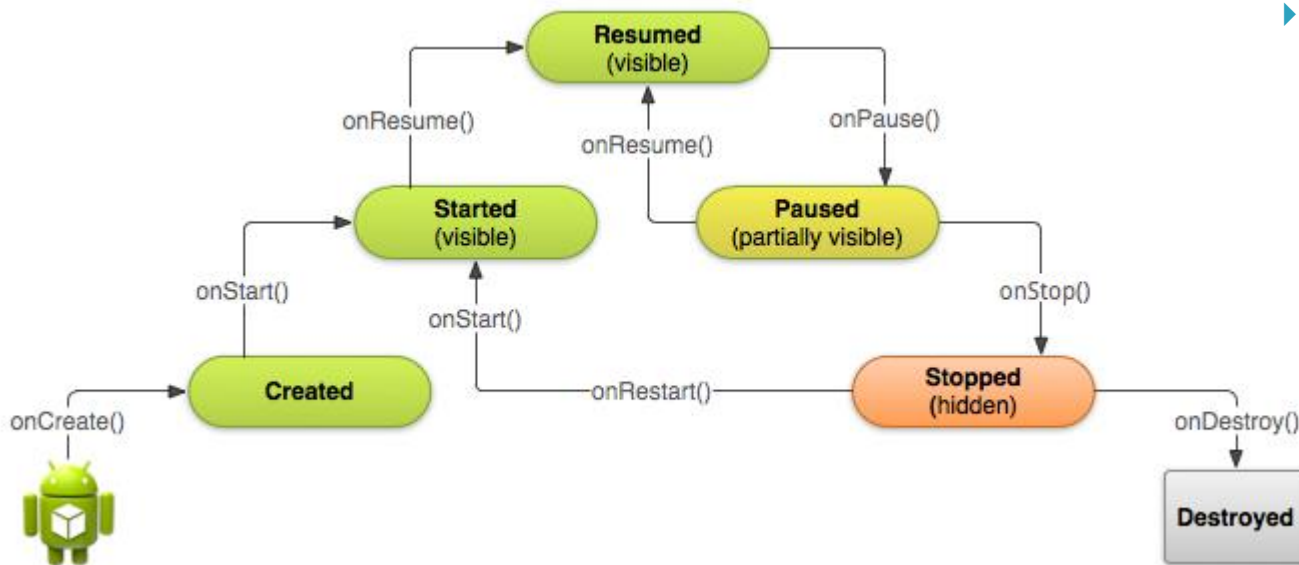
Métodos Callback (I)

- Un **método callback** es un método que implementamos en nuestra clase para que sea **llamado cuando se produzca un determinado evento**.
- Ejemplos de eventos: *nos llaman, abrimos otra aplicación, pulsamos botón volver,...*
- Ejemplos de métodos “callback” relacionados con las activities: *onCreate, onStart, onResume,...*
- Estos métodos **controlan el “ciclo de vida” (lifecycle) de la actividad**.
- **Por ejemplo** se les llama cuando la actividad es creada, destruida, detenida, retomada...

Gestión de las Activities por Android:

Métodos Callback (II)

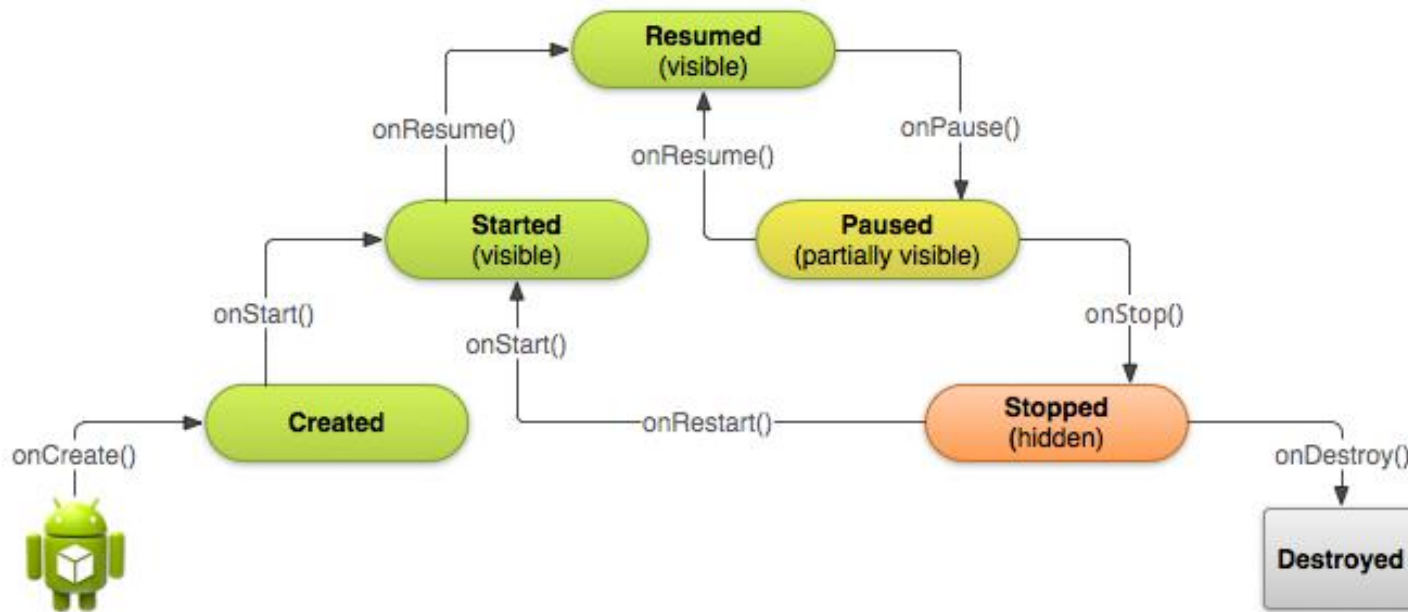
- ▶ Durante la vida de una activity, el sistema llama a una serie de **métodos del ciclo de vida** en una secuencia de **pirámide escalonada**.
- ▶ Cada **estado del ciclo de vida** de la Activity es un **escalón** separado de la **pirámide**.
- ▶ A partir de que el **sistema crea una instancia de una Activity**, cada método callback mueve el estado de la activity un **escalón hacia arriba**.



- ▶ La **cima de la pirámide** es el punto en que la **Activity** esta **“running”** en primer plano e interactuando con el usuario.

Gestión de las Activities por Android:

Métodos Callback (III)



- ▶ Cuando el **usuario deja la activity**:
 - El sistema llama otros métodos del ciclo de vida
 - El **estado de la activity se mueve hacia abajo** en la pirámide.
 - En algunos casos, **la activity se moverá sólo parcialmente hacia abajo** en la pirámide **y espera**. *Por ej. Cuando el usuario cambia a otra app.*
 - En estos casos, *podría volver a subir al top y seguir donde el usuario lo dejó*. Por ej. Si el usuario vuelve a la activity.

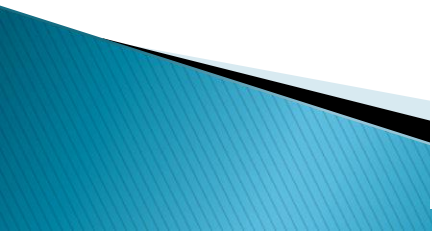
Gestión de las Activities por Android:

Métodos Callback (IV)

- ▶ Dependiendo de la complejidad de nuestra Activity, **probablemente no necesitaremos implementar todos los métodos** del ciclo de vida.
- ▶ Un **método callback** es un método que implementamos en nuestra clase para que sea **llamado cuando se produzca un determinado evento**.

Ciclo de vida de una Activity

Gestión de las Activities por Android: Ciclo de Vida de una Activity (I)



Gestión de las Activities por Android: Ciclo de Vida de una Activity (II): Estados de una Activity (I)

- ▶ El ciclo de vida describe los **estados** y las **transiciones entre esos estados** en los que puede encontrarse una “activity”.

- ▶ Los estados posibles son:

1. RESUMED (a veces llamado “RUNNING” ejecutándose):

- 📄 *Activity en primer plano de la pantalla y tiene el foco del usuario.*

- 📄 *Interactúa con el usuario*

2. PAUSED (pausado):

- 📄 *Activity aún visible en la pantalla, pero parcialmente oscurecida y **no interactúa con el usuario, ni puede ejecutar código.***

- 📄 *Puede ser:*

- 📄 *Porque otra activity transparente esté en estado running (ejecutándose) y tiene el focus.*

- 📄 *Porque aparece un diálogo*

- 📄 *Pantalla del teléfono está bloqueada*

Estados de una Activity: *cont. Paused* (II)

📄 *Una activity pausada puede ser eliminada por el sistema Android sin avisar en cualquier momento.:*

- *Por ejemplo, si se encuentra con problemas de memoria.*

3. STOPPED (detenido):

📄 *Ocorre cuando la **activity** está **completamente oculta** por otra activity.*

📄 *Deja de estar visible en la pantalla, **está en segundo plano** (BACKGROUND).*

📄 *Se guarda la instancia y su estado.*

📄 *Una activity detenida, al igual que la pausada, puede ser eliminada por el sistema Android sin avisar en cualquier momento.:*

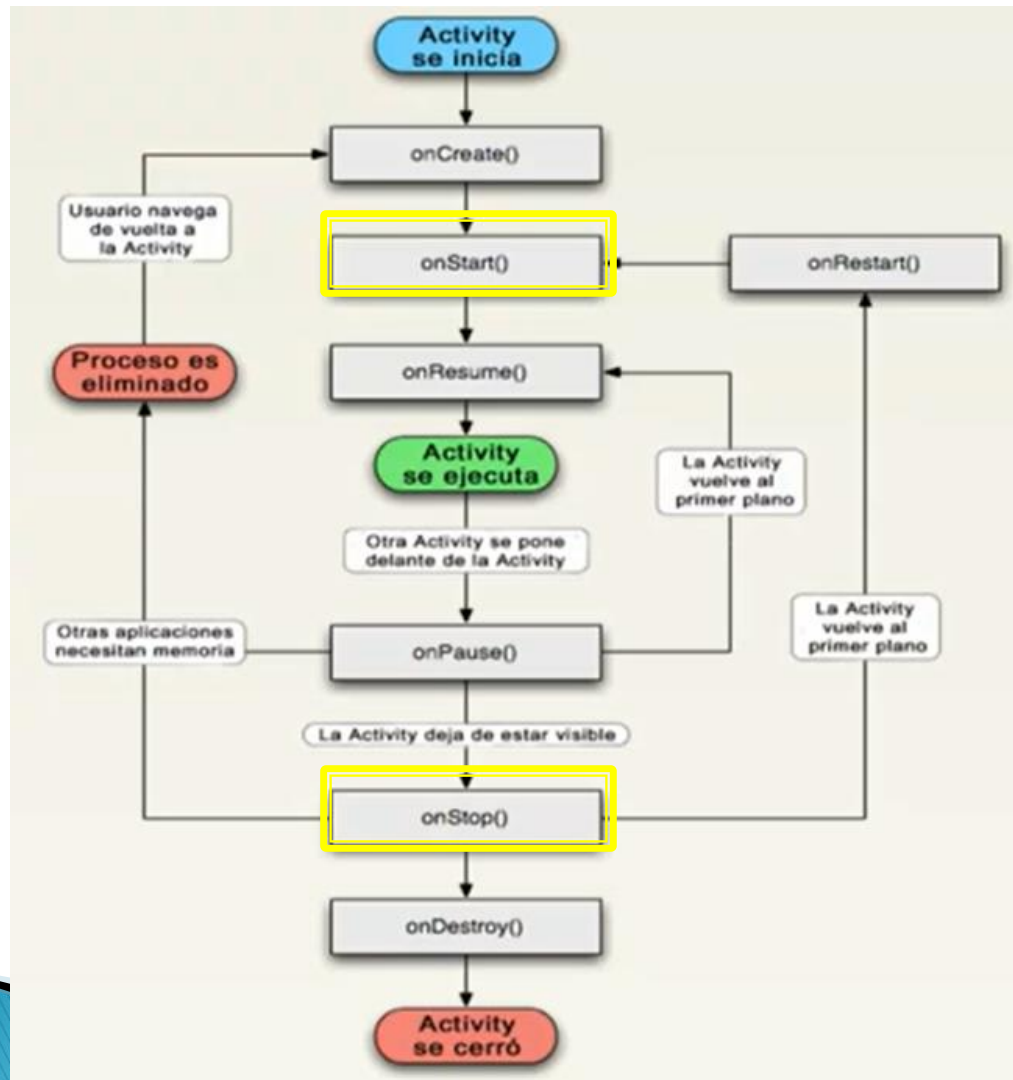
4. CREATED y STARTED:

📄 *Estos dos estados son **temporales** y el sistema rápidamente se mueve desde ellos al siguiente estado llamando al siguiente CALLBACK.*

📄 *Es decir, después que el sistema llama **onCreate()**, rápidamente llama **onStart()** y seguido llama **onResume()**:*

Gestión de las Actividades por Android:
Ciclo de Vida de una Activity (III):
Estados de una Activity: (III)

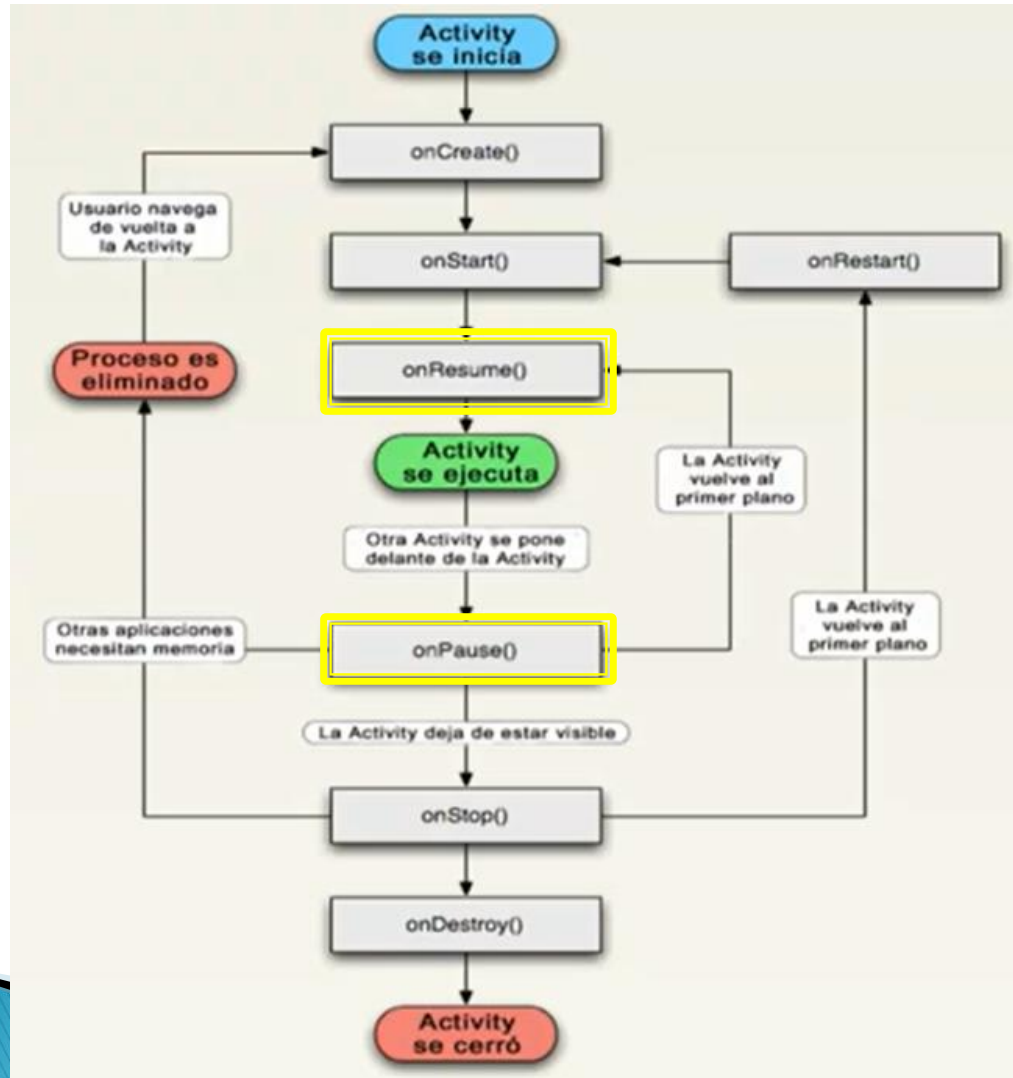
Periodo durante el cual una activity es visible:



VISIBLE
(desde comienzo
método "onStart()" hasta fin método
"onStop()")

Gestión de las Actividades por Android:
Ciclo de Vida de una Activity (III):
Estados de una Activity: (IV)

Periodo durante el cual una activity es visible:



**VISIBLE &
FOREGROUND**
(desde comienzo
método
“`onResume()`”) hasta
fin método
“`onPause()`”)

Gestión de las Activities por Android:
Ciclo de Vida de una Activity (IV):
Estados de una Activity (V)

- ▶ Una activity “**paused**” o “**stopped**” puede **volver al estado “running”** en cualquier momento *(si el sistema no la destruyó)*:
- ▶ Cuando eso ocurre ***sigue siendo la misma instancia Java***. Por tanto: *mismas variables miembro y mismo estado*.
- ▶ Cuando el sistema quita una activity de memoria puede hacerlo llamando al **método *finish()* o *matando su proceso***.

Gestión de las Activities por Android: Ciclo de Vida de una Activity (V): Métodos Callback detallados (I)

- ▶ La clase “**Activity**” cuenta con una serie de **métodos “protected”** que vamos a poder **sobreescribir**.
- ▶ *Son métodos que nos van a informar de los cambios de estado de una activity y permiten responder ante ellos.*
- ▶ Para cada activity, debemos implementar los métodos callback necesarios para cuando nuestra actividad sea **creada**, **running**, **pausada**, **detenida** o **destruida**.
- ▶ Todos estos métodos deben llamar SIEMPRE al método correspondiente de la superclase “super”:

```
public class HelloWorld extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
    }  
}
```

- ▶ Debe ser la primer instrucción del método siempre..

Gestión de las Actividades por Android:
Ciclo de Vida de una Activity (VI):
Métodos Callback detallados(II)

MÉTODO		DESCRIPCIÓN	SEGUIDO DE...	Se suele usar para
onCreate()		<ul style="list-style-type: none"> - Llamado cuando se crea la Activity la primera vez. - Debe implementarse SIEMPRE - Lleva un parámetro Bundle que contiene el previo estado de la Activity guardado (si se guardó). 	<ul style="list-style-type: none"> - onStart() 	<ul style="list-style-type: none"> - Crear vistas. - Crear estructuras de datos. - Enlazar datos a listas, ...
	onRestart()	<ul style="list-style-type: none"> <input type="checkbox"/> Llamado si la activity ha sido parada y está a punto de comenzar de nuevo (<i>después de onStop()</i>). <input type="checkbox"/> NO SE SUELE PROGRAMAR. SE HACE EN onStart(). 	<ul style="list-style-type: none"> - onStart() 	<ul style="list-style-type: none"> - procesamiento especiales necesitados sólo después de haber sido parada.
	onStart()	<ul style="list-style-type: none"> - Llamado justo antes de que la activity llegue a ser visible al usuario. - Puede ser llamado después del onCreate() o bien del onRestart() porque venga después de haber sido detenida 	<ul style="list-style-type: none"> - onResume() si esta arrancando - onStop() si se va a parar 	<ul style="list-style-type: none"> - Peticiones de actualización a los sensores de localización. - Verificar que una característica del sistema requerida está activada - Actualizar la cola de mensajes no leídos para una aplicación lector de correo.

Métodos CallBack detallados(III)

MÉTODO		DESCRIPCIÓN	SEGUIDO DE...	Se suele usar para
	onResume()	<input type="checkbox"/> Se invoca cuando la actividad es visible y va a empezar a interactuar con el usuario. (después de onStart())	– onPause()	– Lanzar animaciones y música
	onPause()	<ul style="list-style-type: none"> – Llamado cuando está a punto de running otra activity. – Debería de hacer su trabajo MUY RÁPIDO; pues la otra Activity no es “Resumed” hasta que este método no termine. <input type="checkbox"/> Puede ser la última notificación que recibamos de nuestra activity porque el <u>sistema puede decidir eliminarla silenciosamente.</u>	<ul style="list-style-type: none"> – onResume() si la activiy vuelve al frente. – onStop() si se hace invisible. 	<ul style="list-style-type: none"> – Salvar datos de edición y otros a almacenamiento persistente. – Parar animaciones, música y otras cosas que pueden consumir CPU. – guardar borradores de emails, – liberar recursos del sistema (broadcast receivers, gestores de sensores, etc),...

MÉTODO		DESCRIPCIÓN	SEGUIDO DE...	Se suele usar para
	onStop()	<ul style="list-style-type: none"> – Llamado cuando la Activity va a dejar de ser visible para el usuario y no se necesitará durante un tiempo. – Puede ser que esté siendo destruida, o porque otra activity haya sido “resumed” y la cubra. – Siempre va precedido de ONPAUSE() 	<ul style="list-style-type: none"> – onRestart() si esta volviendo a interactuar con el usuario – onDestroy() si se va a destruir 	
onDestroy()		<ul style="list-style-type: none"> – Llamado cuando la activity va a ser destruida. – Puede ser llamado porque alguien llamó a “finish()” o porque el sistema la está destruyendo. – Se puede distinguir entre esos dos escenarios con: isFinishing() 	<ul style="list-style-type: none"> – nada 	<ul style="list-style-type: none"> – Liberar recursos. – Destruir thread que hayamos creado

Gestión de las Activities por Android:
Ciclo de Vida de una Activity (VII):
Métodos CallBack detallados(V)

onCreate()

- Este método debe implementarse **SIEMPRE**, ya que **se llama cuando se crea la actividad**.

- Puede recibir en un parámetro que será:

- **información de estado** de instancia (en una instancia de la **clase Bundle**), por ***sí se reanuda desde una actividad que ha sido destruida y vuelta a crear.***

✍ Esta información se habría **guardado previamente** por **onSaveInstanceState()**.

- **null** si es la **primera vez** que se crea la actividad.

Gestión de las Activities por Android:
Ciclo de Vida de una Activity (VIII):
Métodos CallBack detallados:
cont “onCreate()” (VI)

- En este método, **SIEMPRE** debe llamarse a “**setContentView()**”, para **asignar un layout a la interfaz de usuario** de la activity.
- Si no se hace, la activity no podrá tener interfaz de usuario.

Gestión de las Activities por Android:
Ciclo de Vida de una Activity (XIII):
Métodos CallBack detallados:
onPause() (VII)

IMPORTANTE: *onPause()*

- *Como puede ser que la activity se destruya sin avisar, debemos guardar todos los estados que queramos que persistan en este método.*
- *Porque no hay ninguna seguridad de que los siguientes métodos vayan a ejecutarse.*
 - ▶ *En **onPause** se deberían escribir todos los datos que realmente fuesen importantes (por ej. el trabajo hecho por el usuario).*
 - ▶ *Esto entra en **conflicto** con la rapidez necesaria para **onPause** al ser usado frecuentemente, por lo que **hay que escoger qué es verdaderamente importante guardar.***

Gestión de las Activities por Android: Ciclo de Vida de una Activity (XVIII): Resumen (I)



El método `onResume()` es llamado siempre, antes de que la activity entre en el Estado Running.



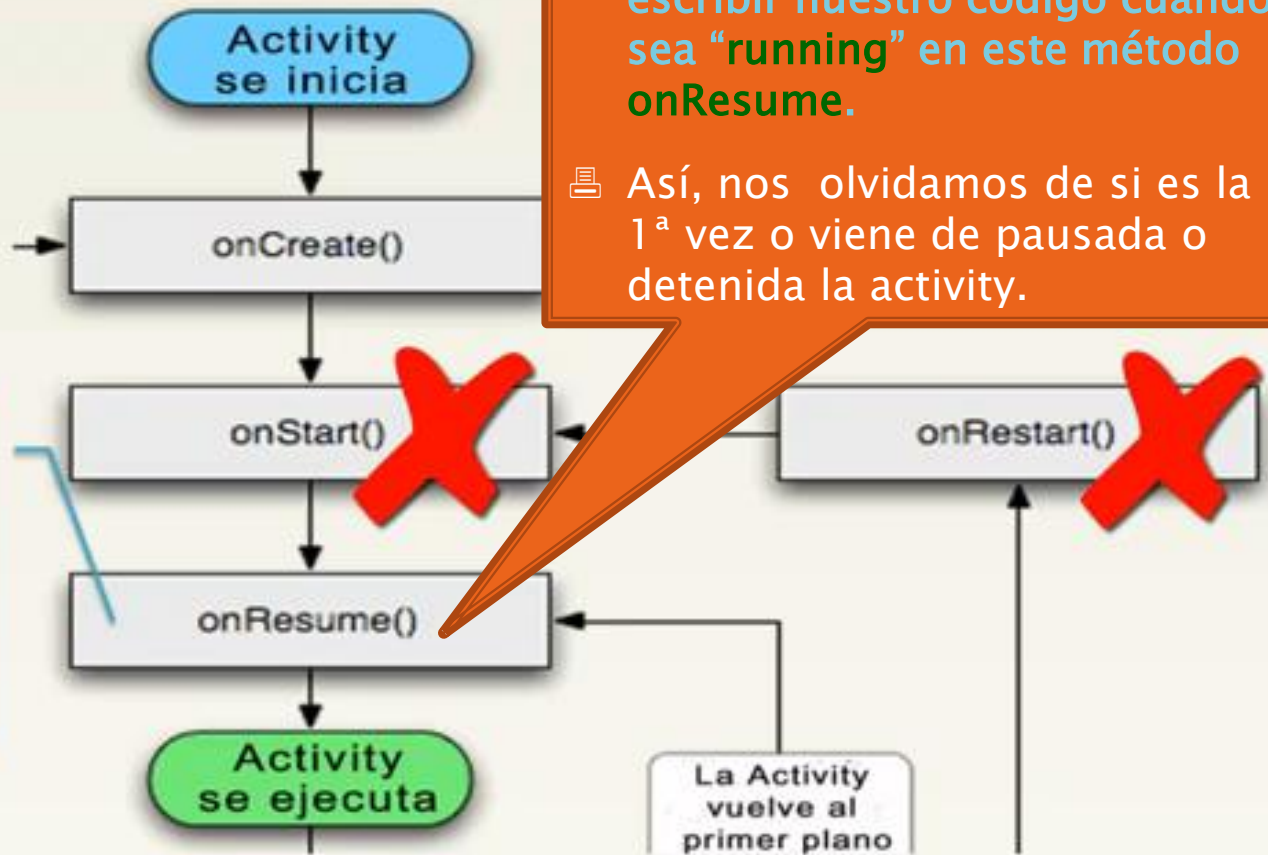
ⓘ Antes de que nuestra activity pase a “RUNNING” el método “`onResume`” es siempre llamado.

ⓘ No importa si se inicia por primera vez o se reinicia desde un estado anterior.

Gestión de las Actividades por Android: Ciclo de Vida de una Activity (XIX): Resumen (II)



El método `onResume()` es llamado siempre, antes de que la activity entre en el Estado Running.



Por tanto, podemos ignorar los métodos `onStart` y `onRestart`, y escribir nuestro código cuando sea "running" en este método `onResume`.

Así, nos olvidamos de si es la 1ª vez o viene de pausada o detenida la activity.

Gestión de las Activities por Android: Ciclo de Vida de una Activity (XX): Resumen (III)



La Activity puede ser destruida silenciosamente después de onPause()



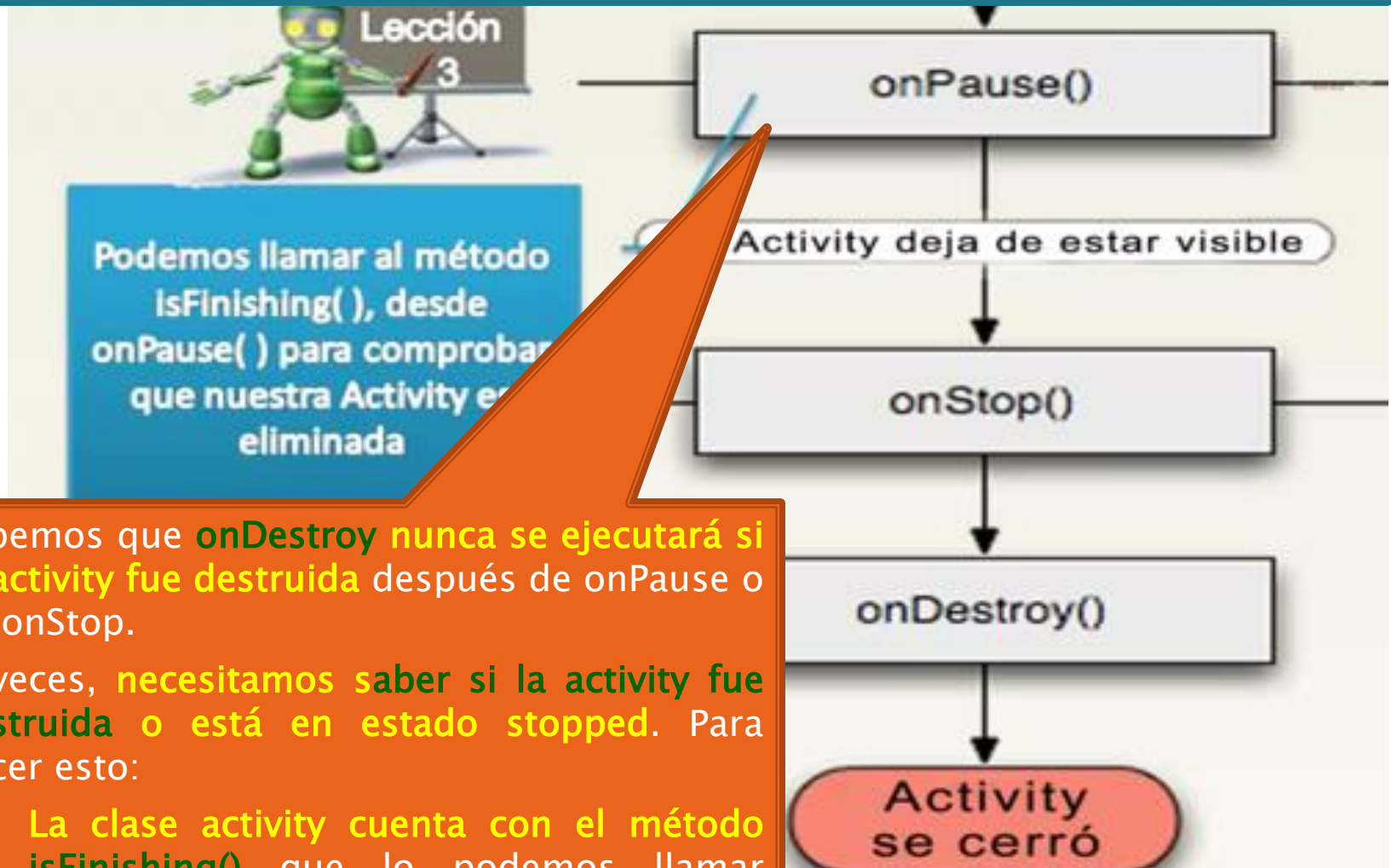
📄 No podemos asumir nunca que después de un onPause viene un onStop y un onDestroy.

📄 Puede ser que se hubiese **destruido** la activity.

📄 Por tanto, **podemos ignorar los métodos onStop y onDestroy**, y escribir nuestro código en el método onPause.

📄 En este método **onPause es donde debemos salvar toda la información que queremos que persista.**

Gestión de las Activities por Android: Ciclo de Vida de una Activity (XXI): Resumen (IV)

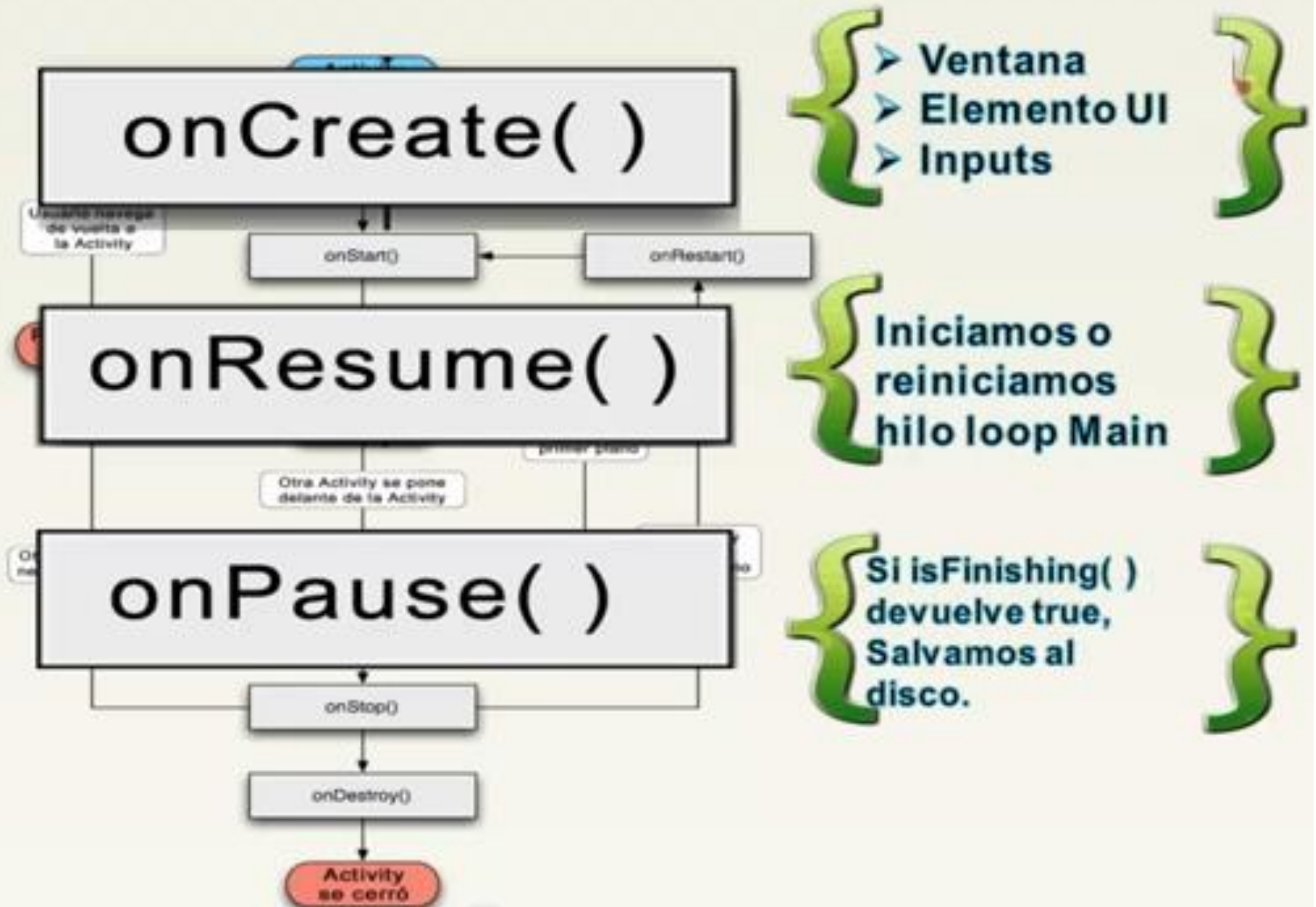


📄 Sabemos que **onDestroy nunca se ejecutará si la activity fue destruida** después de `onPause` o de `onStop`.

📄 A veces, **necesitamos saber si la activity fue destruida o está en estado stopped**. Para hacer esto:

- **La clase activity cuenta con el método `isFinishing()`** que lo podemos llamar desde **onPause** para saber si la activity va a ser eliminada o no.

Gestión de las Activities por Android: Ciclo de Vida de una Activity (XXII): Resumen (V)



**¿Cuál es la Activity que
lanza mi APP desde la
pantalla HOME:
la PRINCIPAL?**

Launcher Activity (I)

- ▶ Launcher Activity es:
 - la actividad que se muestra en la pantalla principal del dispositivo móvil y,
 - La que arranca nuestra app al pulsar sobre ella (se ejecuta su `onCreate()`).
- ▶ ¿Cómo sabe Android cuál es la “Launcher Activity”?
 - Debemos indicarlo en el **AndroidManifest** con un `<intent-filter>` que incluye:
 - La acción **MAIN**
 - La categoría **LAUNCHER**.
 - Por defecto, al crear un nuevo Proyecto, Android lo hace ya él.

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

CREAR UNA ACTIVITY

¿Cómo crear una Activity? (I)

- 1) Crear una **subclase de Activity** (*o de una subclase ya existente*).
- 2) **Implementar los métodos callback necesarios** para cuando nuestra actividad sea: **creada**, detenida, reanudada,...
- 3) Obligatorio implementar el método **onCreate()**.
- 4) Los dos **callbacks más importantes** son:

 **onCreate()**

 **onPause()**

¿Cómo crear una Activity? (II)

- 5) Debéis **declarar la Activity en el manifest**:
- *Si es la **activity** que aparece al crear un nuevo proyecto, **no hará falta**. Android lo hace por ti.*
 - *En los demás casos:*
 - *Si se crea la nueva Activity con: **FILE**→ **NEW ACTIVITY**, Android la declarará por nosotros.*
 - *Si creáis manualmente una clase **JAVA** y luego un **LAYOUT**, la tendréis que **declarar vosotros**.*
 - **RECORDAD CÓMO SE HACÍA EN la presentación de “AndroidManifest–Declarar componentes”**

Usando “Intent-Filters” (I)

- ▶ Una actividad, como la mayoría de componentes, puede declarar varios **<intent-filter>**, para declarar cómo otros componentes pueden activarlo.
- ▶ Al crear una nueva aplicación (usando las herramientas del SDK), la actividad creada automáticamente ya incluye el intent-filter que responde a la acción MAIN (*punto de entrada de la app*) y que debe colocarse en la categoría “launcher” (*que aparezca en el lanzador*).

```
<intent-filter>
```

```
    <action android:name="android.intent.action.MAIN"/>
```

```
    <category android:name="android.intent.category.LAUNCHER"/>
```

```
</intent-filter>
```


Usando “Intent-Filters” (II)

- ▶ Si la aplicación no va a permitir a otras aplicaciones acceder a sus actividades, no hace falta añadir más filtros.
- ▶ En este caso, las actividades de la misma aplicación pueden usar intents explícitos que se dirijan a otras actividades de la misma aplicación *por su nombre*.
- ▶ En caso contrario, se deberán incluir otros <intent-filter> (intents implícitos) que contengan un elemento <action> y puede que un <category> y/o <data>.

LANZAR UNA ACTIVITY SENCILLA

Lanzar una Activity (I)

- ▶ Se puede lanzar otra “activity” **llamando a “startActivity()”, pasándole un “objeto Intent”** que *describe la actividad que queremos lanzar*.
- ▶ **Hay 2 tipos de “Intents”** según la “activity” que queramos lanzar:

1. Si la “activity” es conocida, usaremos “**Intents explícitos**”:

📄 Para ello, se crea un “**Intent**” usando el **nombre de la clase que define la “activity”** que queremos lanzar

📄 Este caso suele darse cuando trabajamos con nuestras ***propias aplicaciones***.

📄 **Ejemplo**: una activity lanza otra llamada “***SignInActivity***”:

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

Lanzar una Activity (II)

2. Si no queremos **lanzar ninguna activity en concreto**, sino realizar una acción concreta, como :

- *mandar un email, un mensaje de texto*,... usando datos de nuestra activity y no tenemos ninguna activity que lo haga en nuestra aplicación

En este caso, podemos **usar activities de otras aplicaciones**. Usaremos “**intents implícitos**”.

📄 Para ello, podemos crear un intent que **describa la acción que se quiere realizar** y **el sistema lanza la actividad apropiada desde otra aplicación**.

Lanzar una Activity (II)

📄 Si existen **múltiples actividades** que pueden manejar el intent, **el usuario podrá elegir** la que quiere usar.

📄 Porque el sistema le mostrará un menú con las app disponibles.


📄 **Ejemplo:** si queremos enviar un e-mail:

```
Intent intent = new Intent(Intent.ACTION_SEND);  
intent.putExtra (Intent.EXTRA_EMAIL, recipientArray);  
startActivity(intent);
```

📄 **Intent.ACTION_SEND** indica que la intención es **enviar datos** desde esta activity a otra.

📄 El **extra EXTRA_EMAIL** añadido al intent es un **array strings** de direcciones de email a los que se debería mandar el email.

Lanzar una Activity (III)

- Cuando una aplicación de email responde a este intent:
 - lee el array de strings suministrado en el extra y lo mete en el campo "to" del formulario de composición del email.
 - En esta situación, la activity del email de la aplicación se lanza y cuando el usuario ha terminado se relanza la actividad inicial.
 - Además indica que estamos enviando un e-mail.
-  **Android busca entre las aplicaciones y sus activities cuáles pueden manejar este intent.**

**LANZAR UNA ACTIVITY
QUE DEVUELVA
RESULTADOS**

Lanzar una Activity que devuelva resultados (IV)

- ▶ A veces, se quiere **recibir un resultado de la actividad** que se lanza.
- ▶ En este caso, se ejecuta la actividad llamando al método **“startActivityForResult()”**.
- ▶ **Para recibir el resultado de la actividad**, hay que implementar el **método** callback **“onActivityResult()”**.
- ▶ **Cuando la actividad es realizada, retorna un resultado en un Intent al método “onActivityResult()”,** que obviamente, deberemos implementar para que haga alguna cosa con el resultado.

Ejemplo de `startActivityResult()`

```
private void pickContact() {  
    // Intent para escoger un contacto, tal y como sea definido en la URI  
    intent = new Intent(Intent.ACTION_PICK, Contacts.CONTENT_URI);  
    startActivityForResult(intent, PICK_CONTACT_REQUEST);  
}  
  
@Override  
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
    if (resultCode == Activity.RESULT_OK && requestCode == PICK_CONTACT_REQUEST)  
    {  
        // Consultar el nombre en el content provider  
        Cursor cursor = getContentResolver().query(data.getData(), new String[]  
        {Contacts.DISPLAY_NAME}, null, null, null);  
        if (cursor.moveToFirst()) {  
            // Verdadero si el cursor no es vacío  
            int columnIndex = cursor.getColumnIndex(Contacts.DISPLAY_NAME);  
            String name = cursor.getString(columnIndex);  
            // Aquí podemos hacer algo con el nombre...  
        }  
    }  
}
```

Explicación del ejemplo de `startActivityResult()`

- ▶ **Intent.ACTION_PICK** especifica que se quiere lanzar una activity que muestre una lista de objetos (un conjunto de datos) a seleccionar para que el usuario elija uno de ellos. Una vez elegido, la activity devuelve la URI del elemento elegido.
- ▶ **Contacts.CONTENT_URI** es la URI donde se guarda la información de contactos (*todos los contactos*).
- ▶ **PICK_CONTACT_REQUEST** es una constante definida en la clase y que identifica nuestra petición o solicitud. Se usa como “requestCode” (*codigo de solicitud*) para comprobar si lo que hemos recibido corresponde con el código que enviamos en el “startActivityResult”. Es para distinguir diferentes datos que nos puedan llegar.
- ▶ **Activity.RESULT_OK** nos llegará en resultCode si todo ha ido bien.

Explicación del ejemplo de `startActivityForResult()`

Se verá con más detalle y se harán
ejercicios sobre ello en el TEMA DE
INTENTS

FINALIZAR UNA ACTIVITY

Cerrar o finalizar una Activity

- ▶ Se usa el método **finish()**.
- ▶ Se puede **cerrar otra actividad diferente** que ya **se hubiese lanzado por medio de `startActivityForResult()` con `finishActivity()`**
- ▶ Por lo general, **no es aconsejable cerrar nosotros nuestras actividades**, porque Android las administra para nosotros.
- ▶ **Sólo deberían usarse en circunstancias muy concretas en las que queramos sí o sí que el usuario salga de la actividad.**