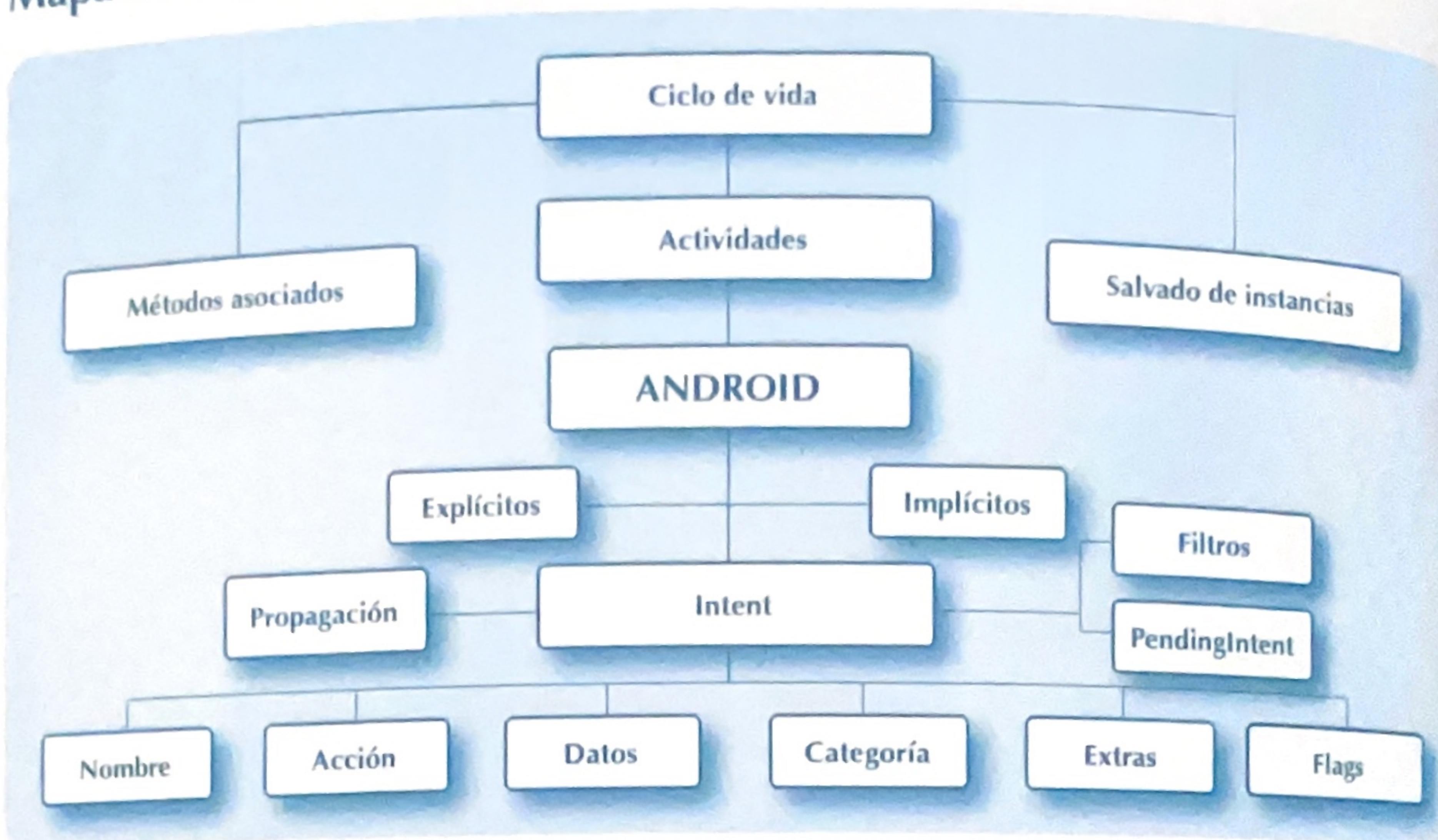


## Mapa conceptual



## Glosario

**API (Application Programming Interface).** Conjunto de protocolos, funciones y comandos informáticos que permiten a los desarrolladores crear programas específicos para determinados sistemas operativos y, además, integrar el software de las aplicaciones.

**Callback (llamada de vuelta).** Es una función que recibe parámetros de una función primaria, que se ejecutará una vez que esta haya terminado de ejecutarse, realizando algún tipo de actividad según la información recibida.

**Foreground.** En términos informáticos, es contrario a **Background**, y hace alusión a aquellos procesos que se ejecutan en primer plano o que están mostrándose a través de la pantalla.

**Layout.** Define la estructura visual en la interfaz de usuario; actúa como contenedor de objetos. Puede declararse en Android mediante XML o añadiendo una instancia en Java en tiempo de ejecución.

**Lenguaje de marcas.** Son aquellos lenguajes o formas de codificación en los que, junto al texto y el código, incorporan etiquetas o marcas que dan información suplementaria sobre la estructura del texto o su presentación.

**Override.** Término utilizado en Java (como lenguaje orientado a objetos) que permite sobrecargar o modificar el comportamiento original de un método heredado.

**URI (Uniform Resource Identifier).** Es un estándar definido que permite identificar y facilitar la localización de recursos en Internet (páginas web, imágenes, videos...).

### 3.1. Introducción

En este capítulo se darán los primeros pasos en programación nativa en Android, para lo que es fundamental conocer las Activity (actividades) y, especialmente, su ciclo de vida y los métodos asociados a cada uno de sus momentos. También se estudiará el funcionamiento la pila y, lo que es más importante, cómo reaparecen las tareas, conservándose las instancias de manera idéntica a como estaban antes de que pasaran a un segundo plano.

En el segundo bloque se abordarán las intenciones (Intent), una potente clase que se ocupa de llamar a una u otra actividad, de manera directa (explícita) o indirecta (implícita), dejando que sea el sistema quien nos aconseje cuál de las utilidades o aplicaciones en él instaladas es más útil para la acción que deseamos realizar.

### 3.2. Las actividades

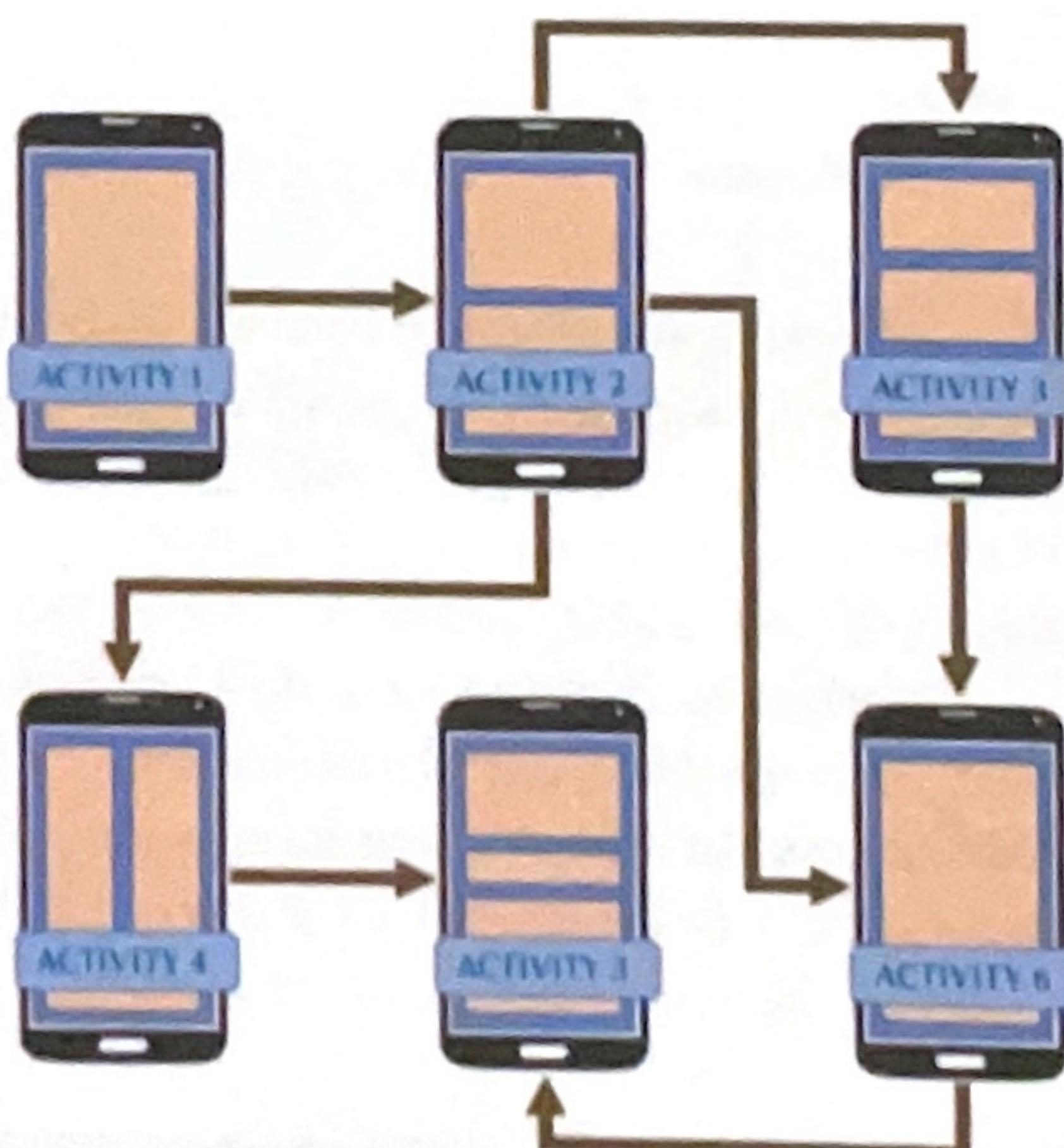
La actividad (Activity) es el principal componente de una aplicación de Android y se encarga de gestionar gran parte de las interacciones con el usuario. A nivel de lenguaje de programación, una actividad hereda de la clase Activity y se traduce en una pantalla. Por tanto, y en términos generales, una aplicación de Android tendrá tantas Activities como pantallas distintas tenga.

Una aplicación generalmente consiste en múltiples actividades vinculadas de forma flexible entre sí. En el diseño de una aplicación, las Activities tendrán una secuencia de aparición según la lógica con la que se haya construido la misma. Aparecerán una u otras pantallas según el flujo del programa y las decisiones que en cada caso tome el usuario.

Desde una pantalla principal (main), se podrán realizar desplazamientos de una a otra Activity y, conforme se va “navegando” por las diferentes pantallas de la aplicación, el dispositivo va recordando la secuencia “visitada”, de tal manera que si se retrocede utilizando los botones del dispositivo o la lógica de la aplicación, irán apareciendo pantallas en sentido inverso a como se habían visionado. Android almacena la posición de cada una de las Activities (pantallas) en una “pila”, Stack (también llamada Back Stack), en este caso, de tipo LIFO (Last In, First Out), acrónimo que significa que el último en entrar será el primero en salir.

El sistema de funcionamiento de esta pila es muy simple: cuando la actividad en curso inicia otra, esta nueva actividad obtiene el foco y es empujada a la parte superior de la pila; la actividad anterior permanece en la pila, pero detenida. El conjunto de actividades que tienen una relación lógica y que se encuentran en la pila se denomina tarea (Task).

Si el usuario regresa (Back), la actividad que tenía el foco se elimina, y la actividad previa y que se encontraba por debajo de ella en la pila se restaura y vuelve a coger el foco.



**Figura 3.1**  
Navegación entre actividades de una aplicación.

Si se continúa presionando Atrás, se irán eliminando sucesivamente tareas hasta llegar a la pantalla de inicio. Cuando todas las actividades se han eliminado de la pila, la tarea deja de existir y desaparece de la memoria.

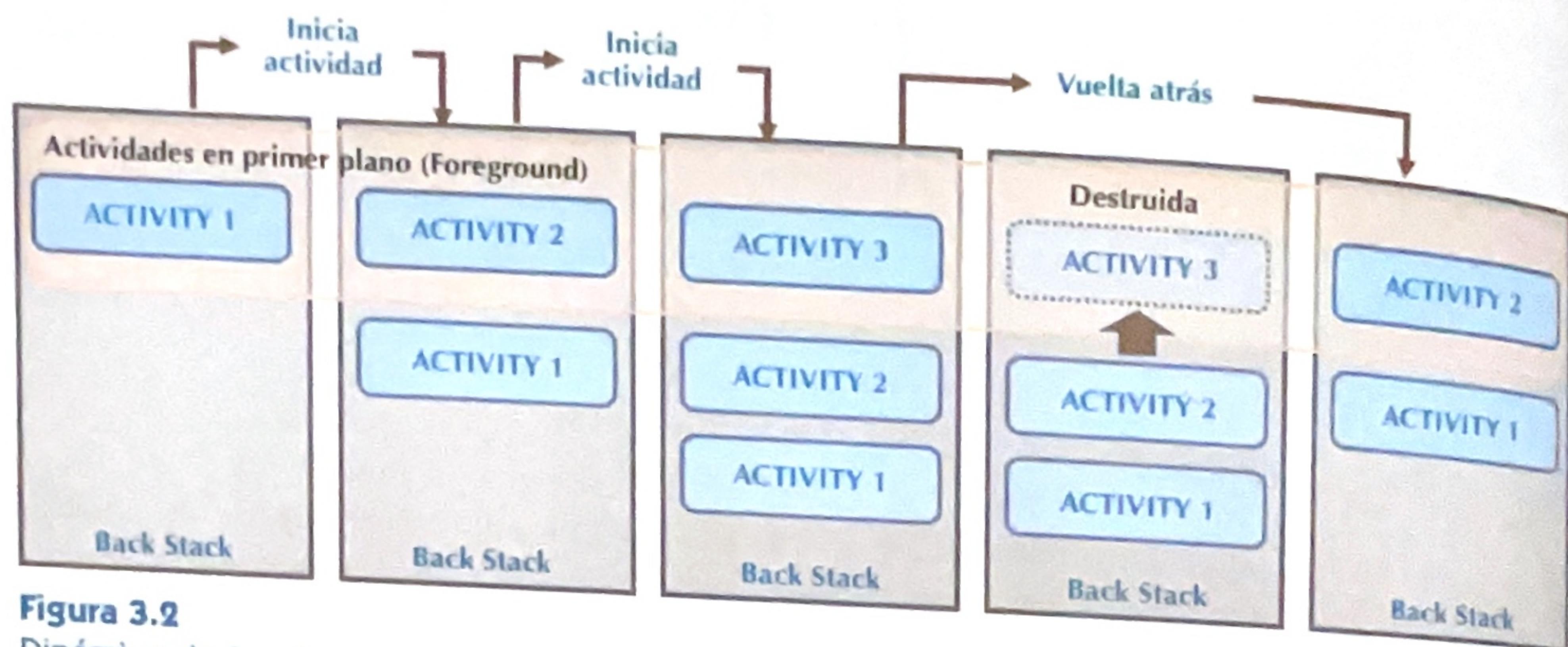


Figura 3.2

Dinámica de funcionamiento del Back Stack.

Una tarea puede moverse en conjunto a un “segundo plano” cuando el usuario comienza una nueva tarea o va a la pantalla Inicio. Si está en segundo plano, todas las actividades de la tarea se paran, pero el Back Stack de esta tarea se conserva. Si una tarea vuelve a situarse en “primer plano”, esta se volvería a posicionar en el lugar donde se había dejado.

**RECUERDA**

- ✓ Una tarea es un conjunto de actividades con las que el usuario interactúa para realizar un determinado trabajo.

Cuando se detiene una de las actividades, y teniendo en cuenta las características del dispositivo (especialmente la memoria), es posible que el sistema, si necesita recursos de memoria, elimine esa actividad completamente, pero aun así mantiene su lugar en la pila de actividades para cuando necesite volver a primer plano.

WWW

**Recurso web**

En este código QR encontrarás un vídeo de Android Developer sobre el funcionamiento del Back Stack.



Cuando esto suceda, el sistema debe volver a crearla (en lugar de reanudarla), y para evitar que se pierda el trabajo que tenía en curso, se implementa el método `onSaveInstanceState()`, que asegura la recuperación de variables y otros valores ligados a ella.

Por lo que se ha visto hasta ahora, las actividades funcionan dentro del dispositivo como procesos que pueden estar en diferentes estados o momentos de ejecución.

- a) *Foreground Process*. Es el proceso en que se encuentra la actividad que se está mostrando en pantalla y con la que el usuario puede interactuar.
- b) *Visible Process*. La actividad es visible, pero aún no se puede interactuar con ella. Aunque este estado parezca tener poco sentido, su uso tendrá especial utilidad para el programador, ya que es una situación considerada importante por el sistema operativo.
- c) *Service Process*. Proceso que lleva una actividad que está funcionando como un servicio. Hace cosas en segundo plano y generalmente importantes.
- d) *Background Process*. Aquel que tiene una actividad no visible y situada en un nivel inferior de la pila.
- e) *Empty Process*. Se trata de aquellos procesos que no contienen nada y que Android suele usar para crear sobre ellos un proceso nuevo.

### Investiga



Son muchas las posibilidades que te ofrece el conocer de manera correcta la gestión de tareas y de la pila de actividades. Visita la página de Android Developer y familiarízate con su manejo:



### 3.3. Ciclo de vida de una aplicación

Como se ha visto en el apartado anterior, una actividad puede estar en diferentes períodos de funcionamiento que denominaremos *estados*, y que van a depender del flujo de la aplicación, de la interacción con el usuario y de las necesidades de recursos del sistema.

La actividad debe relacionarse con el sistema operativo y con otros programas que pueden estar en diferentes estados en el dispositivo móvil. Teniendo en cuenta el estado de funcionamiento de la misma, para Android, la actividad puede ser:

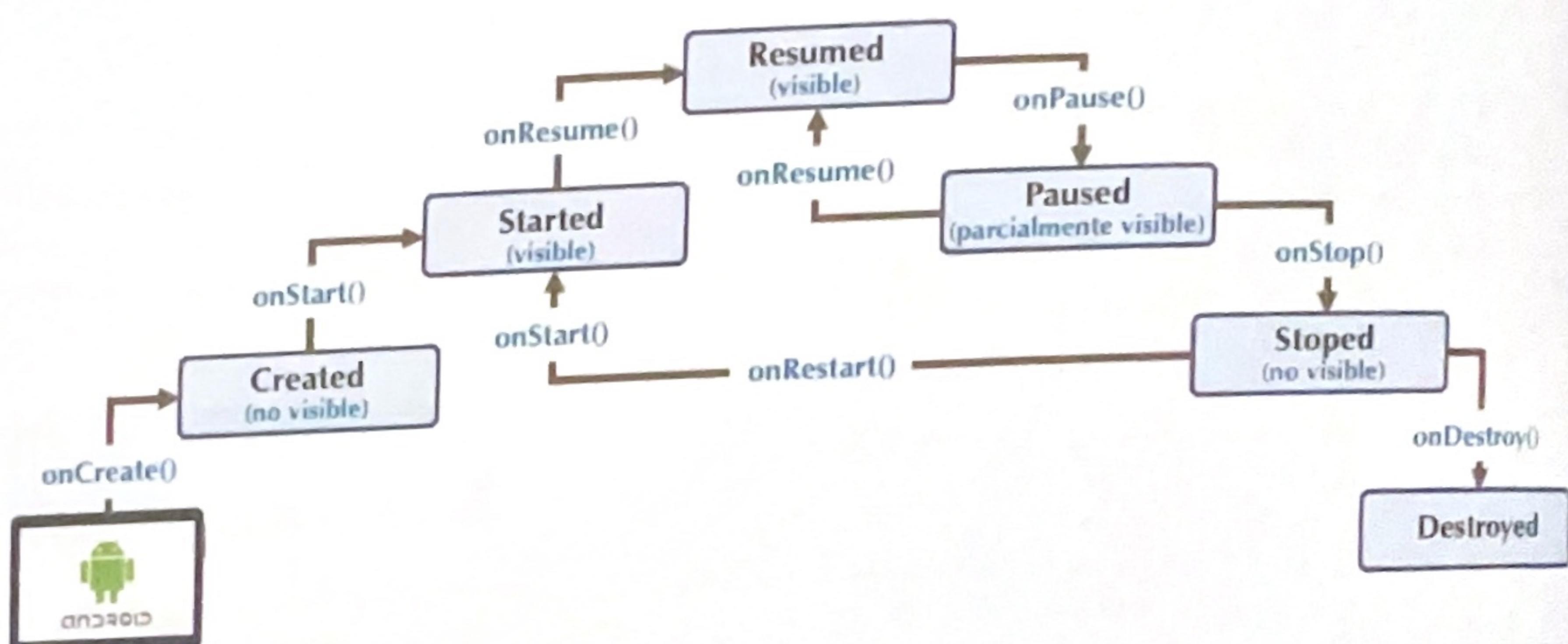
- *Inexistente*. Cuando no ocupa memoria, por lo que, evidentemente, no se puede interactuar con ella.
- *Detenida*. Sí ocupa memoria, aunque no es visible y, por tanto, tampoco es accesible.
- *Pausada*. Es visible, pero no se puede interactuar con ella.
- *Activa*. Está en primer plano del dispositivo y el usuario puede interrelacionarse con ella.

Estos estados suelen ir en sucesión creciente cuando la actividad se pone en marcha, y en sentido decreciente cuando desaparece y se destruye, por lo que el conjunto de estados en el que se puede encontrar una actividad es comúnmente denominado *ciclo de vida*.

Los cambios de estado pueden ser controlados a través de métodos, llamados por Android tipo *callback*. En la figura 3.3 pueden verse los nombres de dichos métodos en el proceso que transcurre desde la construcción hasta la destrucción de la actividad.

Es necesario aprender a controlar el ciclo de vida de una aplicación y saber manejar cada uno de estos métodos para hacer que la aplicación sea lo más robusta posible.

1. `onCreate()`: es llamado cuando la actividad es invocada por primera vez. Es el momento en que se crean las vistas (el interfaz de usuario), se hace una reserva de memoria para los datos, se obtienen las referencias e instancias de componentes mediante `findViewById()`, se suelen hacer las consultas iniciales en las fuentes de datos y se guardan las variables del entorno, como anteriormente se ha comentado (Bundle).



**Figura 3.3**  
Ciclo de vida de una actividad en Android.

2. `onStart()`: este método sucede al anterior. La actividad se hace visible para el usuario, pero aún no se puede interactuar con ella. Suele ser el lugar donde ubicar instrucciones asociadas al interfaz de usuario.
3. `onResume()`: un método que es llamado cuando la actividad puede interactuar con el usuario. Sigue al anterior y precede al `onPause()`. En este periodo se suelen activar sensores, cámara, ejecutar animaciones, actualizar información...
4. `onPause()`: ocurre cuando nuestra actividad pasa a un segundo plano, bien porque está en proceso de destrucción o porque otra actividad pasa a ocupar el primer plano. Es cuando se detienen procesos, animaciones... En este periodo la actividad es parcialmente visible. Puede ocurrir que vuelva a primer plano (`onResume`) o que sea parada (`onStop`), haciendo invisible para el usuario.
5. `onStop()`: este método es invocado cuando la actividad ya no es visible al usuario y otra actividad ha pasado a primer plano. Se pausan animaciones, determinados servicios se detienen (GPS) y se minimizan los recursos consumidos por la actividad, aunque la actividad aún está en memoria. Si queremos reactivarla, se utiliza `onRestart()`, volviendo a primer plano, o `onDestroy()`, si queremos eliminarla definitivamente.
6. `onRestart()`: llamado cuando una actividad parada vuelve a primer plano. Siempre es seguido de un `onStart()`.
7. `onDestroy()`: es llamado cuando la actividad es definitivamente destruida y eliminada de memoria. En su interior se suelen limpiar y liberar recursos, por ejemplo, la cámara.

Por lo que se ha visto anteriormente, se puede decir que el *ciclo de vida completo* de una actividad transcurre entre la llamada a `onCreate()` y la llamada a `onDestroy()`.

El *ciclo de vida visible* de una actividad es aquel en que el usuario puede ver la actividad en pantalla y que transcurre entre la llamada a `onStart()` y la llamada a `onStop()`.

Por último, el *ciclo de vida en primer plano* de una actividad transcurre entre la llamada a `onResume()` y la llamada a `onPause()`. Es el momento que la actividad tiene el foco en el dispositivo y se encuentra por encima de todas las demás actividades que están activas en memoria.

### 3.4. ¡Hola mundo!, de Android

En este apartado se creará la primera aplicación nativa en Android. Será simple, pero marcará la pauta de cómo se ha de proceder para desarrollar el trabajo práctico propuesto a lo largo de este libro.

Para ello, siguiendo los pasos del capítulo anterior, se ha de tener instalado el entorno de desarrollo con el que trabajaremos (aconsejable Android Studio). Tanto este como el sistema operativo deben reconocer el dispositivo móvil físico o alternativamente tener un emulador configurado con el que visionar las aplicaciones que se irán desarrollando.



#### Actividad propuesta 3.1

Para crear un nuevo proyecto debes introducir el nombre del proyecto (Proyecto1), el dominio (`android.ejemplo.es`) y la ubicación (es aconsejable que en estos primeros ejemplos utilicen la raíz de alguna de tus unidades). El cuadro de diálogo te muestra cómo será el nombre del paquete de tu aplicación, en este caso, y si has seguido el nombre y dominio propuesto, será `es.ejemplo.android.proyecto1`. A continuación, se deben definir los dispositivos destino, seleccionando el nivel más bajo de API para el que funcionará esta aplicación. Es recomendable seleccionar API 21: Android 5 (Lollipop), con lo que será funcional en un 85% de dispositivos.

En el siguiente paso, se añadirá una Activity a la aplicación y se configurará el aspecto que esta tendrá. Inicialmente, se debe seleccionar una actividad vacía (Empty Activity). Por último, se ha de introducir el nombre de la actividad (Actividad1) y el nombre de su vista (activity\_actividad1).

Si se han seguido los pasos anteriores, el entorno habrá generado todos los elementos necesarios de la aplicación, las carpetas estudiadas en el capítulo anterior, incluida Grandle Script, con las librerías y referencias necesarias para la compilación y ejecución de la aplicación.

También estará listo el `AndroidManifest.xml` y los ficheros de código `Actividad1.java` (código de la actividad principal) y dentro de res/layout estará `activity_actividad1.xml` (la vista de la actividad principal).

Antes de estudiar el contenido de cada una de las carpetas, y teniendo conectado un dispositivo móvil al equipo y debidamente configurado, es aconsejable ejecutar la aplicación (pulsando la flecha

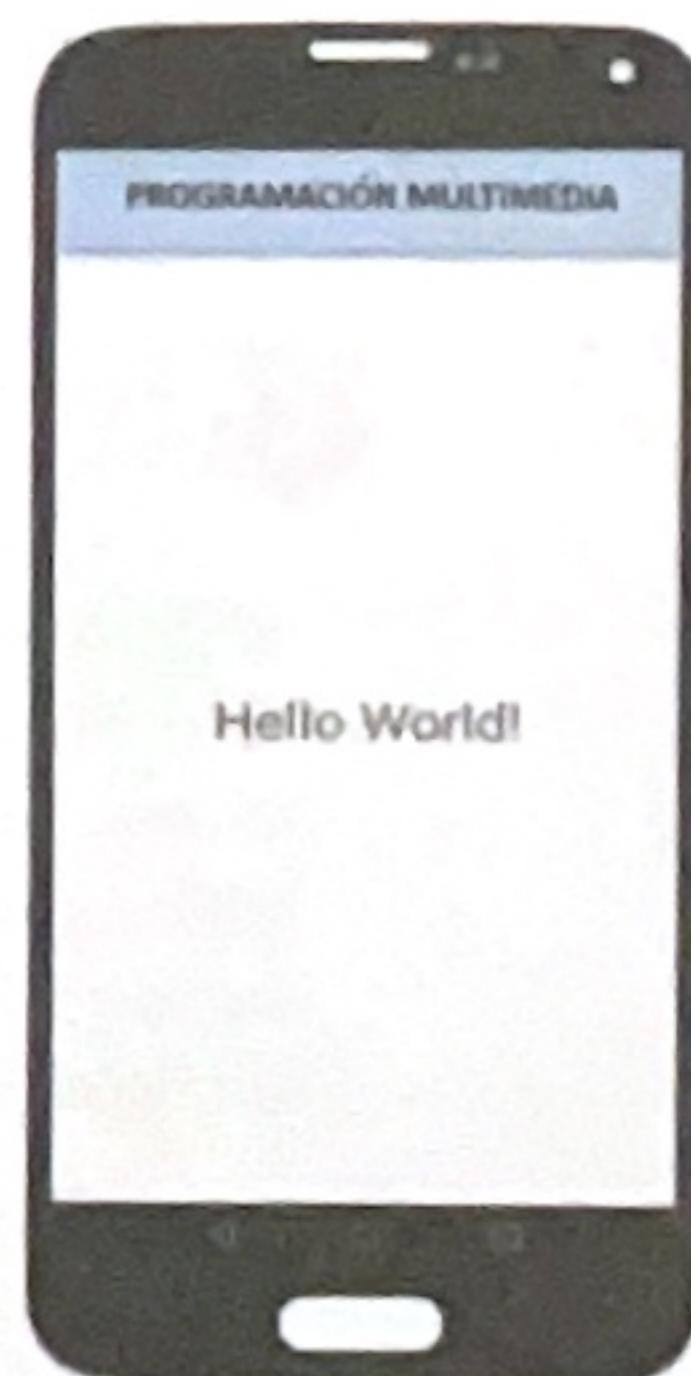


Figura 3.4  
¡Hola Mundo!

verde o Mayús+F10). Tras lo cual, debe de haber aparecido en pantalla algo similar a la imagen mostrada en la figura 3.4.

En un breve análisis de los dos ficheros que soportan el código de esta primera aplicación se puede ver en el fichero XML, responsable de la vista de la actividad (ubicado en res/layout/activity\_actividad1.xml), que en el panel del mismo Android Studio ofrece dos posibilidades de trabajo, vista diseño (Design) y modo texto. Es recomendable, como desarrollador, acostumbrarse a esta segunda, construyendo la vista mediante código, aunque en algunos casos parezca más cómodo la vista gráfica.

El código contenido en la vista se estudiará más adelante, pero ahora basta con conocer que consta de un contenedor (en este caso, un ConstraintLayout) y un objeto de texto en su interior (TextView); este texto es el que ha aparecido en pantalla. Además, también vienen definidos la codificación y el esquema de Android que le permite reconocer los comandos básicos.

Nos detendremos un poco más en el estudio del fichero Proyecto1.java (se encuentra dentro de la carpeta java). Si se han utilizado los parámetros propuestos, esto es lo que debe aparecer.

```
package es.ejemplo.android.proyecto1;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class Actividad1 extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_actividad1);
    }
}
```

Vamos a analizar este código, ya que seguirá un modelo de uso común en la programación en Android. Lo primero que aparece es el paquete en el que irá envuelta la aplicación que, en este caso, es una simple clase de Java, pero con la peculiaridad de que hereda (extends) de la superclase AppCompatActivity, lo que proporciona a esta clase la posibilidad de “entender” el código específico de Android.

Esta es la superclase que se suele utilizar desde la API 15, aunque, como se verá más adelante, en algunas ocasiones se necesitará hacer que la clase herede de otras superclases.



#### PARA SABER MÁS

La gran superclase en la que se basa Android es Activity. Esta es la clase base y todas las actividades deben heredar de ella, directa o indirectamente. La evolución posterior de las diferentes versiones de Android ha hecho que esta clase haya ido relegando funciones en otras superclases. Siendo la relación entre ellas la siguiente y significando en este caso el símbolo < que hereda de...

Activity < FragmentActivity < AppCompatActivity < ActionBarActivity

Por tanto, se puede entender que `android.support.v7.app.AppCompatActivity`, uno de los import que precede a la clase, facilita a la aplicación las librerías que contiene esta superclase.



### Actividad propuesta 3.2

Elimina la extensión de la superclase y observa el efecto que tiene sobre el código. Verás que deja de reconocer los métodos ligados al ciclo de vida de una actividad de Android. Y si eliminas o comentas el import de la superclase, además de no reconocer los métodos, tu aplicación tampoco reconocerá la superclase.

Dentro de la clase Actividad1, tan solo se incluye un método, sobrescrito (@Override), que recordando el apartado anterior, responderá en el momento que la aplicación se cree y entre en funcionamiento en la memoria del teléfono.

Los otros momentos del ciclo de vida de la aplicación sucederán a este, aunque no aparece ninguno de ellos, al no haberse escrito código para los métodos que responden a cada uno de los demás estados.

En el momento que se crea la aplicación, ocurren dos hechos importantes. Por un lado, se prepara un Bundle (paquete de almacenaje) donde guardar las instancias y estados de la aplicación para ser recuperados en caso de necesidad. Para ello también existe una librería (import) que suministra este paquete.

Evidentemente, en este ejemplo hay muy poco que guardar, pero es fácil imaginar lo que puede ocurrir si no existiera esta posibilidad y en pleno funcionamiento de una aplicación, especialmente si tiene formularios de datos, entra una llamada de teléfono y la aplicación se pone en pausa.

Preparado el almacenaje, se invoca `savedInstanceState`, la llamada a la superclase que se ocupa de gestionar el salvado. Más adelante, se profundizará en el guardado de estados en la actividad.

El otro hecho importante que ocurre, cuando se crea la actividad, viene marcado con la línea de código `setContent View(R.layout.activity_actividad1)`. Esta lo que hace es asociar el fichero de la vista a la clase Java, utilizando para ello la clase R como gestora. Y tan solo con eso se “inflará” el contenido visual definido dentro del fichero XML antes visto.

Estos dos puntos son principios muy elementales, pero imprescindibles para dar los primeros pasos en el conocimiento de Android.

Si se añade el siguiente código a la aplicación, se consigue un Log informativo que avisa del paso por el momento del ciclo de vida Startet (para que reconozca este comando, Android Studio importará la librería `android.util.Log`).

Es recomendable filtrar, en el entorno de desarrollo, los Log por tipo (Verbose, Debug, Info, Warn, Error, Assert), además de establecer un filtro personalizado por Tag (en este caso, el Tag definido es “EJEMPLO”).

```
@Override
protected void onStart(){
    super.onStart();
    Log.i("EJEMPLO", "Estoy onStart");
}
```

### Actividad propuesta 3.3



Siguiendo el ejemplo anterior, implementa Logs en tu aplicación que informen del paso por cada uno de los estados posibles del ciclo de vida de la misma. El resultado debe ser algo similar a lo siguiente captura de pantalla. No olvides que una aplicación pasa al estado de parado cuando pasa a segundo plano.

El logcat muestra los siguientes mensajes:

- 2019-08-04 21:32:52.572 24933-24933/es.ejemplo.android.proyecto I/EJEMPLO: Estoy onCreate
- 2019-08-04 21:32:52.577 24933-24933/es.ejemplo.android.proyecto I/EJEMPLO: Estoy onStart
- 2019-08-04 21:32:52.592 24933-24933/es.ejemplo.android.proyecto I/EJEMPLO: Estoy onResume
- 2019-08-04 21:32:58.157 24933-24933/es.ejemplo.android.proyecto I/EJEMPLO: Estoy onPause
- 2019-08-04 21:32:58.542 24933-24933/es.ejemplo.android.proyecto I/EJEMPLO: Estoy onStop
- 2019-08-04 21:33:02.998 24933-24933/es.ejemplo.android.proyecto I/EJEMPLO: Estoy onRestart
- 2019-08-04 21:33:03.001 24933-24933/es.ejemplo.android.proyecto I/EJEMPLO: Estoy onStart
- 2019-08-04 21:33:03.002 24933-24933/es.ejemplo.android.proyecto I/EJEMPLO: Estoy onResume
- 2019-08-04 21:33:06.580 24933-24933/es.ejemplo.android.proyecto I/EJEMPLO: Estoy onPause
- 2019-08-04 21:33:06.931 24933-24933/es.ejemplo.android.proyecto I/EJEMPLO: Estoy onStop
- 2019-08-04 21:33:06.932 24933-24933/es.ejemplo.android.proyecto I/EJEMPLO: Estoy onDestroy

### 3.5. Conservar el estado de una aplicación

Como ya se ha visto en el ciclo de vida, cuando se pausa o se detiene una actividad, se conserva el estado de la misma. Esto ocurre porque el objeto Activity aún se conserva en memoria, y toda la información de sus objetos y su estado actual permanece. Por tanto, cuando la actividad regrese a primer plano, esa información aún estará ahí.

Puede ocurrir que, estando parada, el sistema destruya una actividad para recuperar recursos de memoria; el objeto Activity también se destruye, lo que es desconocido para el usuario y, si este quiere volver a pasarla a primer plano, el sistema debe volver a crear el objeto Activity, para aparecer con el mismo aspecto e información que el usuario la dejó. Todo esto se consigue mediante la implementación de un método de callback, que permite guardar información acerca del estado de nuestra actividad: onSaveInstanceState().

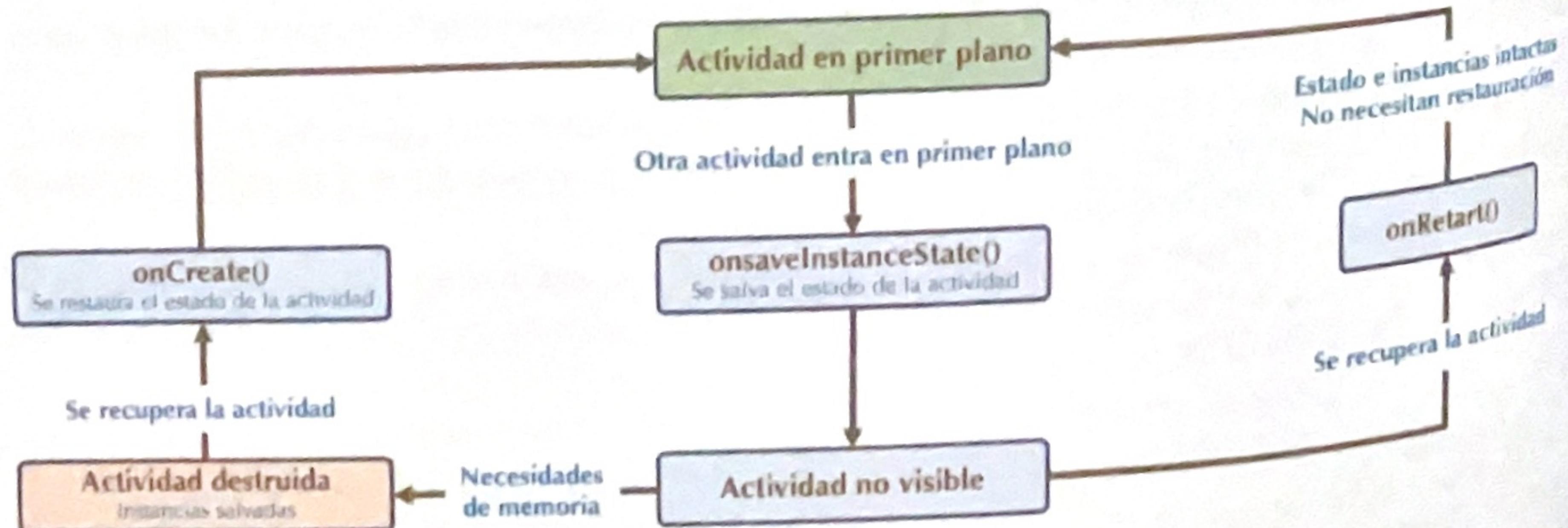
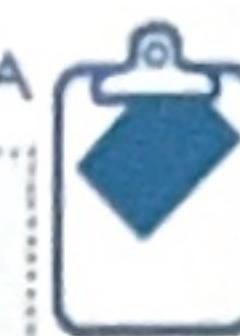


Figura 3.5

Gestión de memoria y salvado de instancias.

Antes de la destrucción, el sistema llama a `onSaveInstanceState()`, método que es pasado a un Bundle, donde puede ser guardada la información del estado, mediante métodos como `putString()` y `.putInt()`. Con la aplicación destruida, cuando el sistema vuelve a crear la actividad se extrae el estado guardado en el Bundle; si no hay información que recuperar (como ocurre cuando se crea por primera vez), el Bundle que se recibe es null.

## TOMA NOTA



La llamada a `onSaveInstanceState()` va ligada de manera específica a cada objeto de la actividad (especialmente de la vista), que conserva el estado de cada uno de ellos, proporcionando en cada vista información sobre sí misma en el momento de la restauración. Para que esto ocurra, es necesario que se proporcione un identificador a cada uno de los objetos que aparezcan en la aplicación (con el atributo `android:id`), de tal manera que si un objeto no tiene ID, el sistema no puede guardar su estado.

Puede ocurrir que cuando el usuario provoque la destrucción de la actividad (saliendo de ella al pulsar el botón de atrás), no se produzca llamada a `onSaveInstanceState()` y, por tanto, no se conserve la información existente. El salvado de estados puede ser deshabilitado, lo que no es aconsejable, mediante el atributo `android:saveEnabled` en `false` o llamando al método `setSaveEnabled()`.

### 3.6. Intents y filtros

Un Intent es un sistema de comunicación que permite interactuar entre componentes de la misma o de distintas aplicaciones de Android. Es un elemento básico de comunicación, tal que con él se puede comenzar una aplicación, iniciar un servicio, entregar un mensaje.

Mediante un Intent se podrá enviar o recibir información desde y hacia otras actividades o servicios, así como lanzar mensajes de tipo Broadcast para identificar cuándo han ocurrido ciertos eventos. Es una manera útil de reutilizar otros componentes que ya desarrollan la funcionalidad que necesitamos en nuestro código, pudiendo, por ejemplo, iniciar una aplicación que nos permita tomar fotografías sin tener que realizar el código de dicha aplicación.

Por sus características y construcción, Android dispone de dos tipos de Intents, explícitos o implícitos.



**Figura 3.6**  
Llamada mediante Intent.

### 3.6.1. Intents explícitos

Los Intents explícitos son aquellos que nombran el componente que se necesita ejecutar, la clase Java específica que se necesita para realizar alguna tarea.

Su construcción es fácil, ya que tan solo se debe instanciar el Intent pasándole como parámetro el contexto y la actividad que se va a lanzar. Posteriormente, lanzarlo mediante el comando `startActivity`.

#### TEN EN CUENTA

- ✓ Todas las actividades que se ejecuten de manera independiente deben de ser previamente declaradas en el Manifest.

#### Actividad propuesta 3.4



En la aplicación creada anteriormente con el nombre de Proyecto1, crea una segunda actividad denominada Actividad2 (te recomiendo que, por rapidez, copies, pegues y renombres Actividad1). Igualmente debes hacer con el fichero XML de la vista (renómbralo con `activity_actividad2.xml`). Ahora asocia en la segunda actividad la vista a este fichero. El código de esta debe quedarte así:

```
public class Actividad2 extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_actividad2);
    }
}
```

Retoca el objeto de texto del segundo XML y sustituye el "Hello World!" por otra cadena distinta. A continuación, añade el siguiente código al método `onDestroy` de la clase Actividad1.

```
Intent ejemplo = new Intent(this, Actividad2.class);
startActivity(ejemplo);
```

Si ahora ejecutas y pulsas el botón Atrás del teléfono, verás que se destruye tu aplicación y da error, esto es debido a que no has declarado esta segunda actividad en el Manifest. Debes hacerlo escribiendo el siguiente código después de la etiqueta de cierre (`</activity>`) de la primera actividad, dentro del fichero manifest.

```
<activity android:name=".Actividad2">
</activity>
```

Si ahora ejecutas y pulsas el botón de Atrás del dispositivo verás que se cierra la Actividad1 y, posteriormente, se abre la Actividad2.

### 3.6.2. Intents implícitos

Los implícitos son aquellos con los que tan solo informamos al sistema la acción que deseamos realizar, y el sistema nos responde con el componente más adecuado para realizar dicha petición, es decir, no se especifica un componente concreto. De tal manera, que si se quisiera ver una página web, sería el sistema el que propondría el navegador con el que hacerlo o preguntaría con cuál hacerlo si tuvieras más de uno instalado en el dispositivo y ninguno configurado como predefinido.



#### Actividad propuesta 3.5

En la aplicación creada anteriormente con el nombre de Proyecto1, añade el siguiente código al método onDestroy de la clase Actividad2. Verás que cuando eliminas esta se te abre el navegador con la página de Google (para cargar las librerías tan solo debes posicionarte sobre los elementos desconocidos y pulsar Alt+Intro).

```
Intent ejemplo= new Intent(Intent.ACTION_VIEW);
ejemplo.setData(Uri.parse("http://www.google.es"));
startActivity(ejemplo);
```

### 3.6.3. Elementos

Si se observa el código anterior a la hora de construir el Intent, se define la acción que se va a realizar y se aportan los datos con los que realizará dicha acción. Además de estos, los elementos que pueden necesitarse para construir un Intent implícito son los siguientes:

#### A) Nombre del componente

Utilizado fundamentalmente en los explícitos. Es donde se identifica el componente que se desea lanzar con el Intent.



#### SABÍAS QUE...

Puedes nombrar un componente de varias maneras:

Usando setComponent()

```
ComponentName componente = new ComponentName(this, Actividad.class);
intent.setComponent(componente);
```

Usando setClass()

```
intent.setClass(this, Actividad.class);
```

Usando setClassName()

```
intent.setClassName("es.ruta", "es.ruta.Actividad");
startActivity(intent);
```

## B) Acción

Mediante `setAction()` se suministra una cadena de texto que informa de la acción que se va a realizar y lo identifica dentro del Manifest. La acción determina, en gran parte, los otros elementos del Intent (datos y extras). Existe una serie de acciones bastante comunes y de frecuente uso, como son:

- ACTION\_CALL
- ACTION\_SET\_WALLPAPER
- ACTION\_DELETE
- ACTION\_VIEW
- ACTION\_SEND

En este ejemplo, se invoca al sistema para realizar una llamada telefónica:

```
Intent ejemplo= new Intent(Intent.ACTION_CALL);
ejemplo.setData(Uri.parse("tel:NÚMERO"));
activity.startActivity(ejemplo);
```

Este otro ejemplo abre la cámara del dispositivo:

```
Intent ejemplo= new Intent(android.media.action.IMAGE_CAPTURE);
activity.startActivity(ejemplo);
```

Las acciones pueden clasificarse en varios tipos:

- Acciones genéricas
- Acciones auxiliares
- Acciones definidas por el usuario

## C) Datos

Contiene la URI (Uniform Resource Identifier) con los datos con los que el componente que reciba el Intent debe trabajar. El tipo de datos determina qué componente usará el sistema para procesarlos. Así, por ejemplo, para construir un Intent implícito que le pida al sistema enviar un correo electrónico, debería hacerse de la siguiente forma.

```
Intent correo= new Intent();
correo.setAction(Intent.ACTION_SEND)
correo.putExtra(Intent.EXTRA_TEXT, mensaje);
correo.setType("text/plain");
```

## D) Categoría

De uso en los implícitos, complementa a la acción, suministrando información adicional sobre el tipo de componente que ha de ser lanzado.

- CATEGORY\_BROWSABLE
- CATEGORY\_HOME
- CATEGORY\_LAUNCHER
- CATEGORY\_PREFERENCE

## E) Extras

Información adicional que será recibida por el componente lanzado. Muy útil en los explícitos, ya que te servirán para mandar información de una Activity a otra. Está formada por un conjunto de pares variable/valor. Estas colecciones de valores se almacenan en un objeto de la clase Bundle. Para llenar el Bundle con información, puede utilizarse alguna de estas posibilidades.

```
putExtra(String name, boolean value);
putExtra(String name, int value);
putExtra(String name, double value);
putExtra(String name, String value);
```

Para recoger la información en destino se utiliza:

```
Bundle b = intent.getExtras();
```

## F) Flags

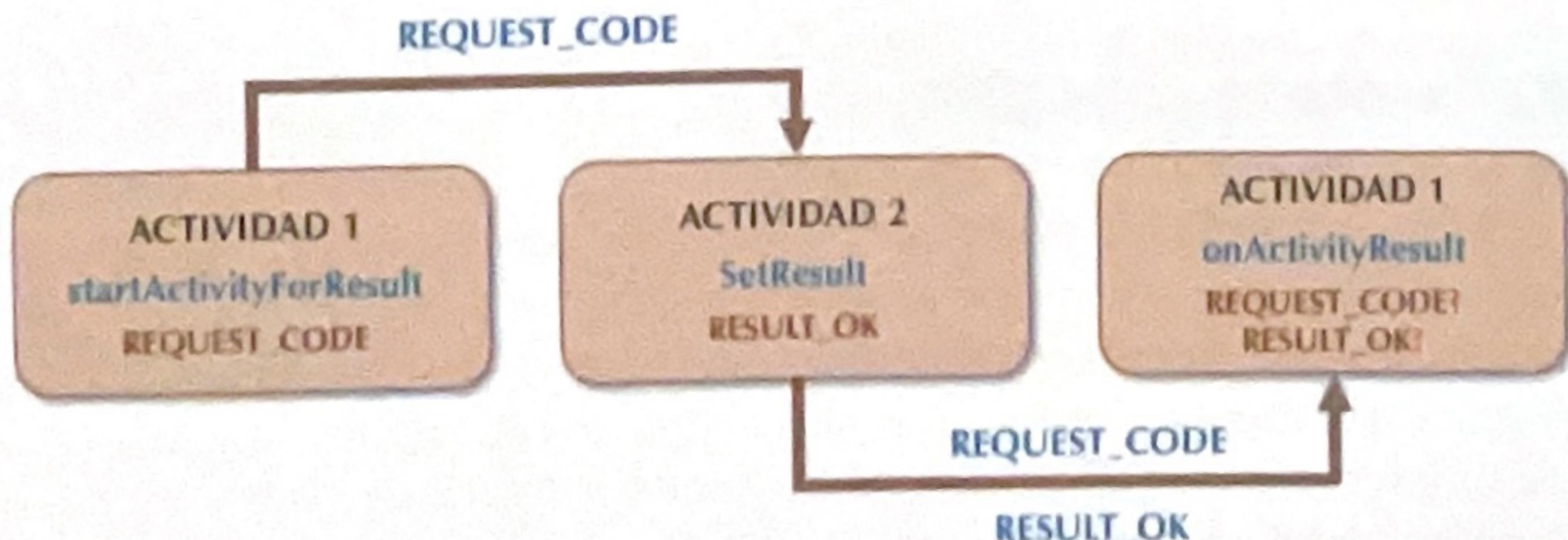
Son metadatos que indican al sistema Android cómo iniciar una actividad y determinar, en parte, el comportamiento del componente que atienda la petición.

### 3.6.4. Propagación

Conocidos los tipos (explícito e implícito), elementos y forma que tienen de construirse los Intents, pueden dársele tres usos fundamentales.

#### A) Para comenzar una actividad

En este caso, además de iniciar la actividad mediante la forma vista, `startActivity()`, puede recibirse el resultado de la actividad cuando esta finalice, mediante `startActivityForResult()`. El proceso se hace utilizando el REQUEST\_CODE (información recibida) y el RESULT\_OK que confirma que la Activity ha finalizado correctamente.



**Figura 3.7**  
Esquema de funcionamiento de `startActivityForResult`.

Para ponerlo en práctica, se puede utilizar el siguiente código en la Actividad 1:

```
private static final int REQUEST_CODE = 10;
-----
Intent ejemplo = new Intent(this, ActividadDos.class);
activity.startActivityForResult(ejemplo, REQUEST_CODE);
```

En el método finish() de la Actividad 2:

```
@Override
public void finish() {
    Intent data = new Intent();
    data.putExtra("RETORNO", "Este es el valor de retorno");
    setResult(RESULT_OK, data);
    super.finish();
}
```

Por último, y de nuevo en la Actividad 1, se comprobará que la información que le llega tiene los códigos correctos para que reciba la información.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode == RESULT_OK && requestCode == REQUEST_CODE) {
        if (data.getStringExtra("RETORNO")) {
            //Código para realizar
        }
    }
}
```

www

## Recurso web

En este código QR encontrarás un vídeo tutorial de programación y más de cómo construir una actividad haciendo uso del método onActivityResult.



### B) Para iniciar un servicio

Un Service es un componente que realiza operaciones en segundo plano, sin interfaz de usuario. Mediante un Intent se puede iniciar un servicio, modificar su comportamiento y establecer conexión entre componentes. El Intent describe el servicio que se debe iniciar y contiene los datos necesarios para ello. Se pueden utilizar los comandos startService(), stopService() y bindService(). Este último caso sirve para establecer un enlace con el servicio de otro componente mediante un Intent.

### C) Para entregar un mensaje

Un mensaje es un aviso que cualquier aplicación registrada puede recibir. Cuando se crea un nuevo receptor Broadcast, se informa al sistema acerca de qué tipos de Intents implícitos pueden ser lanzados con este componente. Pueden enviarse mensajes a otras aplicaciones mediante sendBroadcast(), sendOrderedBroadcast(), sendStickyBroadcast() o sendStickyOrdererBroadcast().

#### Investiga



Busca en la página de Android Developer información del uso de cada una de las opciones de los Intents en los casos de Service y Broadcast.

### 3.6.5. Filtros

De utilidad en el Manifest, determinan la acción que puede llevar a cabo un componente y los tipos de Intents que pueden gestionarla. Los filtros se pueden clasificar por acción, por categoría o por tipo de datos.

### 3.6.6. PendingIntent

Es un tipo de Intent especial en el que se especifica una acción que se va a realizar en el futuro. Pasa un Intent futuro a otra aplicación, así como los permisos necesarios, como si la aplicación creadora estuviese aún en vigor. Puede ser ejecutado en un determinado momento programado mediante el AlarmManager o provocar que se ejecute cuando el usuario realice alguna acción. Suelen ser frecuentemente usados con las notificaciones. Ofrecen tres posibilidades:

- PendingIntent.getActivity(): recupera un PendingIntent para comenzar una actividad.
- PendingIntent.getBroadcast(): Recupera un PendingIntent para escuchar una emisión.
- PendingIntent.getService(): recupera un PendingIntent para iniciar un servicio.
- PendingIntent.getForegroundService(): esto iniciará un ForegroundService.

En el siguiente ejemplo se construirá una actividad que lanzará mediante PendingIntent a una segunda actividad después de transcurrir cinco segundos. Para ello, primero se construye un Intent explícito, que lanzará una actividad denominada Actividad3 (se puede utilizar cualquier actividad creada para ello). Luego se le asocia un PendingIntent, al que se le ha de pasar como parámetros el contexto el requestCode (necesario en caso de cancelarlo), el Intent y el Flag.

```
Intent intent = new Intent(this, Actividad3.class);
PendingIntent miPending = PendingIntent.getActivity(
    this, 1, intent, miPending.FLAG_UPDATE_CURRENT);
```



## PARA SABER MÁS

En este caso, se ha utilizado un Flag de tipo FLAG\_UPDATE\_CURRENT (actualiza este PendingIntent en caso de existir). Otras opciones son:

- FLAG\_CANCEL\_CURRENT: si existe el PendingIntent debe cancelarse.
- FLAG\_ONE\_SHOT: el PendingIntent solo puede usarse una vez.
- FLAG\_IMMUTABLE: el PendingIntent no puede cambiar.

El siguiente paso será ligar el PendingIntent al AlarmManager para que responda después de tiempo indicado. Se instancia este servicio y se configura según el tiempo deseado (no hay que olvidar que el tiempo Android lo trata en milisegundos). El código resultante sería el siguiente:

```
AlarmManager alarma= (AlarmManager) getSystemService(ALARM_SERVICE);
alarma.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis() + (5 * 1000), miPending);
```

Si ahora se ejecuta la aplicación, pasados cinco segundos se inicia la aplicación que se utilizada en lugar de Ejemplo.class. Esto ocurrirá de igual manera si se cierra antes de los cinco segundos la aplicación Padre que gestiona el Intent.

## Resumen

- Una actividad (Activity) es el principal componente de las aplicaciones en Android; equivale a una pantalla y se ocupa de las interacciones con el usuario. En caso de tener más de una actividad en memoria, el sistema se ocupa de almacenarlas en una pila (Back Stack), que es de tipo LIFO. El conjunto de actividades que tienen una relación entre ellas se denomina tarea (Task).
- Las actividades funcionan dentro de nuestro dispositivo como procesos que pueden estar en diferentes estados: Foreground, Visible, Service, Background y Empty Process.
- El estado en que se encuentra en cada momento una actividad forma parte del denominado *ciclo de vida*. Los cambios de estado pueden ser controlados a través de los siguientes métodos: onCreate(), onStart(), onResume(), onPause(), onStop(), onRestart() y onDestroy().
- Según la visibilidad y la interacción con el usuario, los estados de una actividad pueden ser incluidos en un ciclo de vida completo (todos los estados), ciclo de vida visible (puede verse la actividad) y ciclo de vida en primer plano (puede interactuarse con ella).
- En caso de necesidad de recursos, el sistema puede provocar la destrucción temporal de una actividad que el usuario tenía parada. Cuando esta vuelve a primer plano, el sistema debe volver a crearla, recuperando íntegramente el estado anterior gracias al método onSaveInstanceState().

- Los Intents (intenciones) son componentes básicos de comunicación que permiten interactuar entre componentes de la misma o de distintas aplicaciones de Android. Pueden ser utilizados para comenzar una actividad, iniciar un servicio o entregar un mensaje.
- Existen Intents de tipo explícito (cuando se nombra el componente a ejecutar) o implícito (cuando se dice al sistema lo que se desea hacer y él decide el componente más apropiado para realizar dicha acción). Tanto uno como otro pueden estar constituidos por una serie de elementos, como son: nombre, acción, datos, categoría, extras y flags. También se le pueden aplicar filtros para determinar la acción que se va a realizar.
- Un tipo especial de Intent es el PendingIntent, aquel en el que se especifica una acción que se va a realizar en el futuro, ejecutándose en un momento programado como si la aplicación creadora estuviese aún en vigor.

## ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Cómo se denomina el proceso cuando el usuario puede interactuar con él?:

- a) Foreground Process.
- b) Visible Process.
- c) Service Process.
- d) Background Process.

2. Cuando una actividad es visible y se puede interactuar con ella, ¿en qué estado se encuentra?:

- a) Created.
- b) Started.
- c) Resumed.
- d) Paused.

3. ¿Qué método suele utilizarse para incluir la creación de las vistas?:

- a) onCreate().
- b) onStart().
- c) onResume().
- d) onDestroy().

4. ¿Cuál de estos métodos no se incluye en el ciclo de vida visible?:

- a) onCreate().
- b) onStart().
- c) onResume().
- d) onPause().

5. ¿Dónde realiza el salvado de instancias el método savedInstanceState?:

- a) En un recurso XML.
- b) En un Bundle en memoria.
- c) En un fichero SQLite.
- d) En un fichero de texto plano.

6. ¿Cómo se denominan los Intents que son llamados por la acción que se va a realizar?:

- a) Explícitos.
- b) Implícitos.
- c) Diferidos.
- d) PendingIntent.

7. ¿Cuál de los siguientes no es incluido entre los elementos de un Intent?:

- a) Nombre del componente.
- b) Categoría.
- c) Package.
- d) Flags.

8. ¿Cuál de los siguientes no es un uso frecuente de un Intent?:

- a) Comenzar una actividad.
- b) Iniciar un servicio.
- c) Almacenar información.
- d) Entregar un mensaje.

9. ¿Cuál de los siguientes métodos es incorrecto para entregar un mensaje mediante Intent?:

- a) sendBroadcast().
- b) sendOrderedBroadcast().
- c) sendStickyBroadcast().
- d) sendOrdererStickyBroadcast().

10. Con un PendingIntent:

- a) Podemos realizar acciones diferidas.
- b) Podemos construir notificaciones.
- c) Podemos iniciar aplicaciones a la espera de resultados.
- d) Podemos realizar acciones sin necesidad de que la actividad creadora esté activa.

### SOLUCIONES:

1. a b c d

2. a b c d

3. a b c d

4. a b c d

5. a b c d

6. a b c d

7. a b c d

8. a b c d

9. a b c d

10. a b c d