Contenido

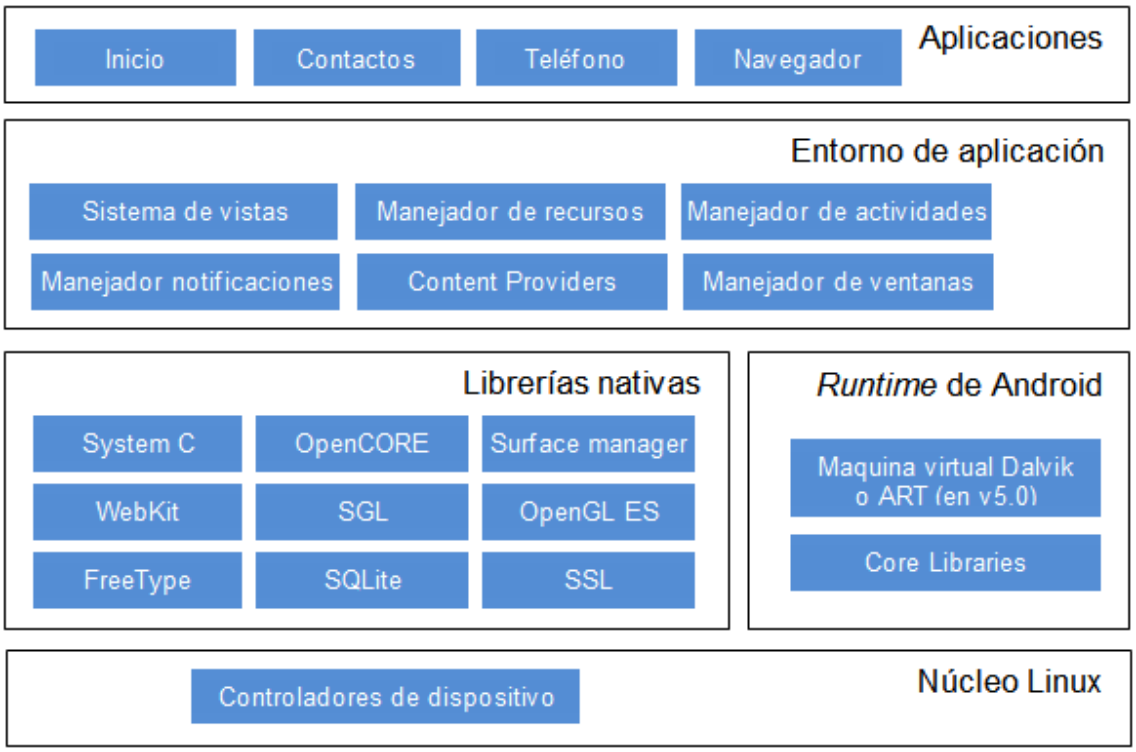
Arquitectura Android

La elección de la arquitectura Android se basa en la idea de controlar los recursos y el consumo. Las aplicaciones Android se ejecutan en un sistema con restricciones (memoria disponible, consumo de batería, diferencias en la visualización, almacenamiento disponible...).

Como desarrollador, deberá prestar especial atención a los siguientes puntos:

- La creación de nuevos objetos.
- El uso de recursos (procesador, RAM, almacenamiento, etc.).
- El consumo de la batería.
- La diversidad de tamaños y resoluciones de pantalla y de configuraciones de hardware.
- La diversidad de versiones de Android disponibles en el mercado.

La arquitectura Android se compone de cinco partes diferenciadas:



- **Aplicación**: representa el conjunto de aplicaciones proporcionadas con Android.
- **Framework Android**: representa el framework que permite a los desarrolladores crear aplicaciones accediendo al conjunto de API y funcionalidades disponibles en el teléfono

(fuentes de contenido, gestor de recursos, gestor de notificaciones, gestor de actividades, etc.).

- **Librerías**: Android dispone de un conjunto de librerías que utilizan los distintos componentes del sistema.
- **Android Runtime**: contiene, entre otros, la máquina virtual ART.
- **Linux Kernel**: el núcleo Linux (2.6) que proporciona una interfaz con el hardware y que gestiona la memoria, los recursos y los procesos Android.

ART (Android RunTime)

Android se basa en una nueva máquina virtual (desde la versión 5.0 Lollipop) particular, denominada **ART** (*Android RunTime*). Remplaza a Dalvik (disponible en las versiones anteriores de Android) y posee varias características particulares:

- AOT Compilation (*Ahead of Time* - compilación anticipada) que permite compilar una aplicación en tiempo de instalación y no en tiempo de ejecución (como hacía Dalvik). Esto permite mejorar enormemente el rendimiento de las aplicaciones, puesto que no se compilan con cada ejecución.
- Mejora del Garbage Collector (recolector de basura).
- Mejora en el desarrollo y depuración de aplicaciones (los fallos ofrecen mensajes más detallados).

Dalvik, la máquina virtual utilizada en las versiones anteriores de Android, se basa en la compilación en tiempo de ejecución (*Just In Time*), lo que significa que su aplicación se compilará con cada ejecución. El paso a ART aumenta el rendimiento, pero también el espacio necesario para almacenar una aplicación (el almacenamiento de la versión compilada de la aplicación).

NDK (Native Development Kit)

Es posible desarrollar de forma nativa en Android utilizando el NDK (*Native Development Kit* - kit de desarrollo nativo), que se basa en C/C++.

También puede utilizar un mecanismo Java que le permitirá llamar a código nativo desde una clase JNI (*Java Native Interface*).

El NDK está disponible en la siguiente dirección: https://developer.android.com/tools/sdk/ndk/index.html

El NDK le da acceso a:

- Un conjunto de herramientas que permiten generar código nativo desde archivos fuente C/C++.
- Una herramienta de creación de archivos apk (véase la siguiente sección: APK (Android Package)) a partir de código nativo.
- La documentación, con ejemplos y tutoriales.

APK (Android Package)

Un APK es el archivo binario que representa una aplicación. Este formato se utiliza para distribuir e instalar aplicaciones.

Para crear un APK, se debe compilar y empaquetar en un archivo una aplicación Android. Este archivo contendrá:

- El código de la aplicación compilada (.dex).
- recursos.
- certificados.
- El archivo de manifiesto.

Para poder publicar su aplicación en **Google Play**, necesita crear un archivo APK.

Para generar su APK desde **Android Studio**, Android proporciona una herramienta que se llama **aapt** (*Android Asset Packaging Tool* - herramienta que permite crear, visualizar y modificar los archivos Android-APK), incluida en el SDK Android e integrada en el entorno de desarrollo.

Puede generar dos tipos de APK:

- Un APK firmado.

- Un APK sin firmar. Este formato puede usarse para probar su aplicación pero no puede publicarse en Google Play.

1. Creación de un keystore

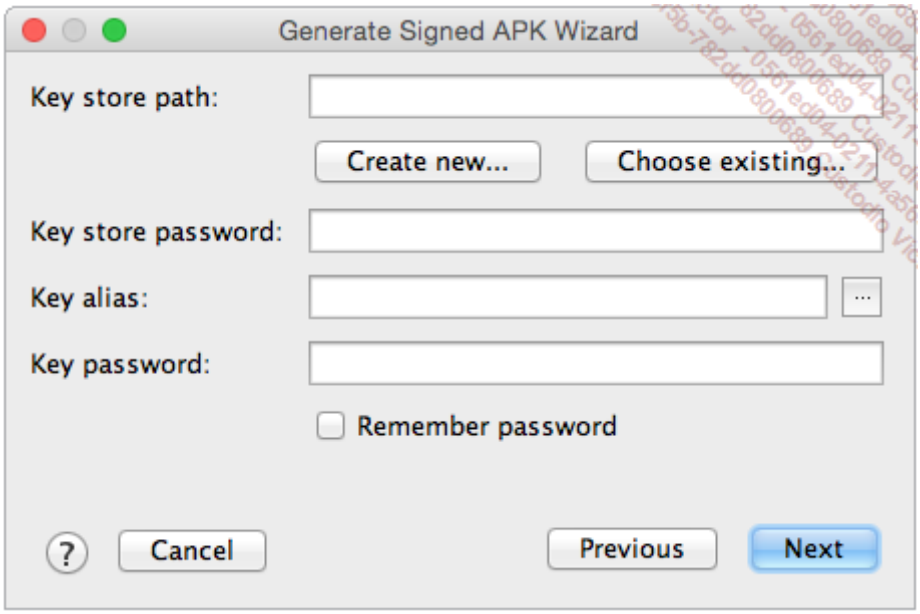
Para generar un archivo APK firmado, necesitará un **keystore** (almacén de claves), que sirve para albergar las claves utilizadas para firmar sus aplicaciones. Puede usar un keystore que ya usaba previamente, o bien crear uno nuevo.

Para crear un keystore hay que:

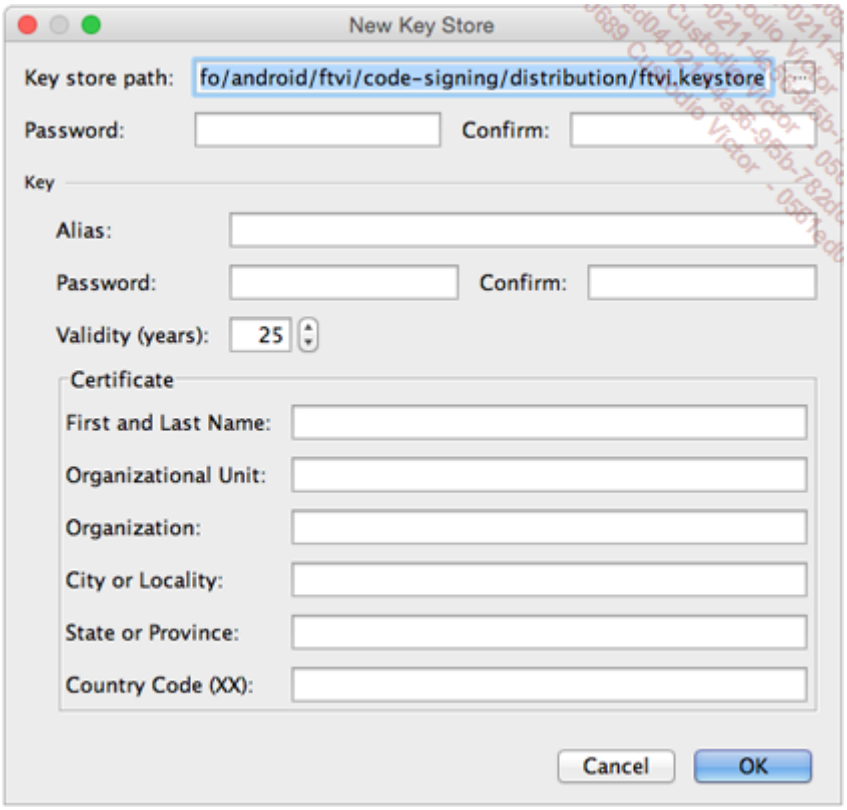
- Elegir una ubicación para guardar el keystore.
- Indicar una contraseña para el uso del keystore.

Un keystore se compone de varias claves de firma, cada clave puede servir, por ejemplo, para firmar una aplicación distinta.

Durante la firma de una aplicación, puede elegir entre utilizar un keystore existente o bien crear uno nuevo.



En la ventana anterior, haga clic en la opción **Create new** (Crear uno nuevo) para crear un nuevo keystore.

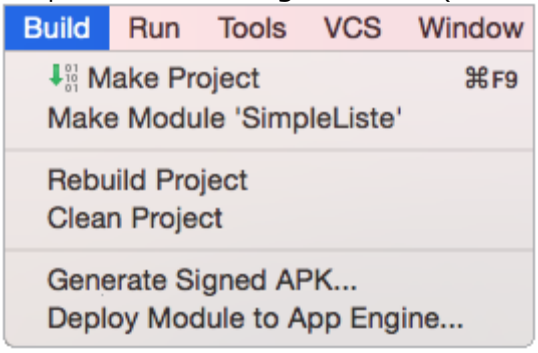


Esta pantalla le permite introducir los datos siguientes sobre su clave:

- **Key store path**: ubicación del keystore.
- **Password**: contraseña del keystore y su confirmación.
- **Alias**: identificador de la clave que servirá para firmar la aplicación.
- **Password**: contraseña de la clave de firma y su confirmación.
- **Validity (years)**: duración de validez, en años.
- Introducir información de al menos uno de los campos adicionales (nombre y apellidos, población, país, etc.).

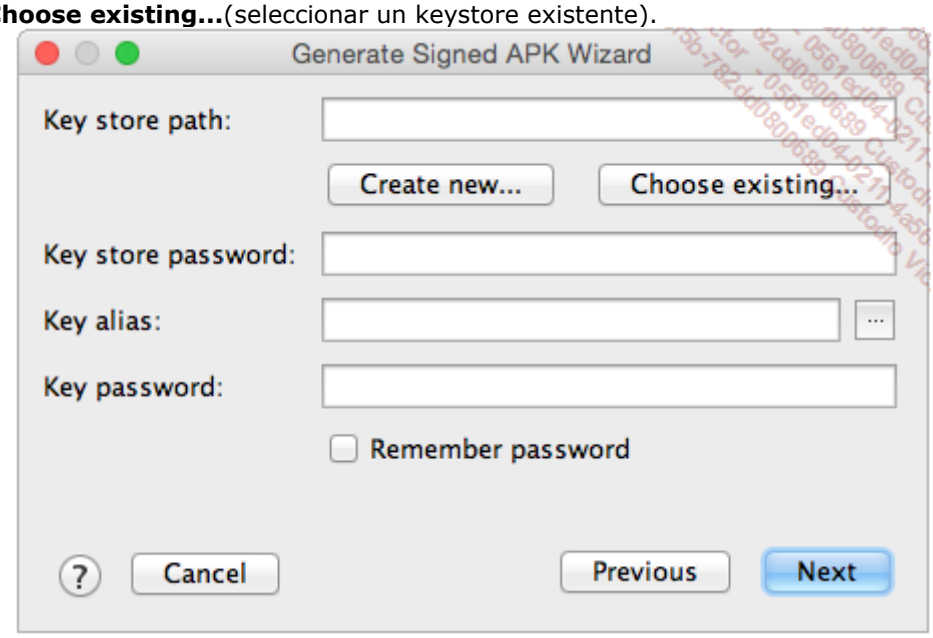
2. Creación de APK firmado

Para generar un APK firmado en Android Studio, haga clic en el menú **Build** (Construir) y, a continuación, seleccione la opción **Generate Signed APK...** (Generar APK firmado).



Seleccione la aplicación correspondiente para generar el APK y, a continuación, haga clic en **Next** (Siguiente).

Seleccione, a continuación, el keystore creado en la sección anterior mediante el botón **Choose existing...**(seleccionar un keystore existente).



Para terminar, seleccione la ubicación del APK generado y haga clic en **Finish** (Finalizar).

Debe conservar su clave de firma cuando publique su aplicación en Google Play. Si la pierde, no podrá actualizar su aplicación nunca más.

También puede crear una clave de firma por línea de comandos mediante el ejecutable **keytool**.

```
[arnau@tulkass ~]$ keytool -genkey -keystore claves -alias "clave firma android" -keyalg RSA -validity 36500
Introduzca la contraseña del almacén de claves:
Volver a escribir la contraseña nueva:
¿Cuáles son su nombre y su apellido?
  [Unknown]:  Arnau
¿Cuál es el nombre de su unidad de organización?
  [Unknown]:
¿Cuál es el nombre de su organización?
  [Unknown]:
¿Cuál es el nombre de su ciudad o localidad?
  [Unknown]:
¿Cuál es el nombre de su estado o provincia?
  [Unknown]:
¿Cuál es el código de país de dos letras de la unidad?
  [Unknown]:  ES
¿Es correcto CN=Arnau, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=ES?
  [no]:  sí

Introduzca la contraseña de clave para <clave firma android>
        (INTRO si es la misma contraseña que la del almacén de claves):
Volver a escribir la contraseña nueva:
[arnau@tulkass ~]$
```

En Windows, el ejecutable keytool se encuentra en la carpeta bin de la carpeta de instalación de Java.

También puede firmar su aplicación a posteriori mediante el ejecutable **jarsigner**.

```
jarsigner -verbose -verify -certs miAplicacion.apk
```

Componentes Android

El framework Android se compone de algunos elementos esenciales para la creación de aplicaciones. En esta sección, se explicarán estos componentes.

1. Activity (actividad)

Una actividad es el componente principal de una aplicación Android. Representa la implementación y las interacciones de sus interfaces.

Tomemos, por ejemplo, una aplicación que enumere todas las imágenes que hay en su teléfono. El proyecto podría descomponerse de la siguiente manera:

- Una vista para mostrar la lista de imágenes.
- Una actividad para implementar la interfaz y las interacciones del usuario con la lista de imágenes.

El framework Android puede decidir "matar" una actividad para liberar recursos para otras aplicaciones (sólo si su actividad ya no está en primer plano) (véase la sección Ciclo de vida de una actividad, más adelante en este capítulo).

2. Fragment (fragmento)

Este concepto se introdujo a partir de la versión 3.0 de Android. Permite construir interfaces más flexibles (smartphone/tableta), dinámicas y fáciles de utilizar.

Un fragmento puede considerarse como una parte de una interfaz. Por lo tanto, una interfaz (actividad) puede componerse de uno o varios fragmentos.

Tomemos una interfaz compuesta de dos vistas. En vez de implementar dos actividades distintas, se podrían utilizar los fragmentos. Esto facilita la creación de dos interfaces diferenciadas para teléfono o tableta.

- **Para teléfono**: los fragmentos se separan en dos actividades diferentes. La selección de un elemento de la actividad A (el fragmento A se incluye en la actividad A) mostrará la actividad B (el fragmento B se incluye en la actividad B).
- **Para tableta**: su interfaz se compondrá de una sola actividad, pero ésta estará separada en dos fragmentos. Cuando el usuario pulse un elemento del fragmento A, actualizará todo el contenido del fragmento B sin iniciar una actividad nueva.

3. Service (servicio)

Un servicio, a diferencia de una actividad, no tiene interfaz pero permite la ejecución de un tratamiento en segundo plano (una operación larga o una llamada remota). Un servicio no se detendrá mientras que no se interrumpa o termine.

Por ejemplo, se puede utilizar un servicio para escuchar música en segundo plano en su teléfono.

4. Broadcast receiver (receptor de eventos)

Un broadcast receiver es un componente que reacciona con un evento de sistema (consulte la sección Intent (intenciones)). Permite, por ejemplo, saber cuándo:

- El teléfono recibe un SMS.
- El teléfono se enciende.
- La pantalla está bloqueada, etc.

Los broadcast receivers no tienen interfaz de usuario y deben realizar tareas ligeras. Si necesita ejecutar una tarea pesada en la recepción de un broadcast receiver, puede, por ejemplo, iniciar un servicio.

Un broadcast receiver no puede realizar modificaciones en sus interfaces, sólo puede mostrar una notificación, iniciar una actividad o un servicio, etc.

5. Content provider (proveedor de contenido)

Un content provider permite compartir los datos de una aplicación. Estos datos pueden estar almacenados en una base de datos SQLite (consulte el capítulo Persistencia y compartición de datos - Almacenamiento en base de datos), en archivos o en la web. El objetivo es permitir a las aplicaciones consultar estos datos.

El sistema Android ofrece content providers (disponibles por defecto) que pueden usarse en sus aplicaciones:

- Contactos.
- Agenda.
- Multimedia, etc.

6. Intent (intenciones)

Los componentes Android (actividad, servicio y broadcast receiver) se comunican mediante mensajes de sistema que se denominan intents. Los emite el terminal para avisar a cada aplicación cuando se activa un evento.

También puede crear sus propios intents para, por ejemplo, iniciar otras actividades (consulte el capítulo Navegación y gestión de eventos).

Hay dos tipos de intents:

- **Explícito**: para invocar, por ejemplo, a una actividad conocida para que ejecute una acción en concreto.
- **Implícito**: para solicitar al sistema qué actividad puede ejecutar una acción específica (por ejemplo, abrir un archivo PDF).

a. Intent-filter (filtros de intención)

Los filtros de intención sirven para indicar a una actividad, servicio o broadcast receiver su comportamiento o el tipo de intents que pueden tratar de forma implícita. Con ello, todo componente que desee ser avisado por intenciones debe declarar filtros de intención.

La manera más común de declarar filtros de intención consiste en utilizar la etiqueta **intent-filter** en el manifiesto (AndroidManifest.xml, consulte la sección Manifiesto) de su aplicación.

En la creación de un proyecto Android, la actividad principal del proyecto contiene filtros de intención.

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

Estos dos filtros de intención significan, respectivamente, que la actividad:

- Es la actividad principal de la aplicación.
- Pertenece a la categoría Launcher, es decir, estará disponible desde el menú principal de inicio de aplicaciones de Android.

También puede crear filtros de intención dinámicamente gracias a la clase **IntentFilter**.

b. pendingIntent

Un pendingIntent corresponde a la descripción de un intent asociado a una acción. Se crea mediante uno de los métodos siguientes:

- **getActivity(Context, int, Intent, int)**: permite generar un PendingIntent que apunte a una actividad.
- **getBroadcast(Context, int, Intent, int)**: permite generar un PendingIntent que apunte a un Broadcast Receiver.
- **getService(Context, int, Intent, int)**: permite generar un PendingIntent que ejecute un servicio.

Permite, a un mecanismo externo a su aplicación (NotificationManager, AlarmManager, etc.), tener los privilegios necesarios para desencadenar en un momento dado un intent en el interior de su aplicación.

El ejemplo más común es el PendingIntent que sirve para desencadenar una notificación tras la recepción de un mensaje (consulte el capítulo Notificaciones).

7. La clase Application

Cada aplicación Android tiene una clase Application. Esta clase se mantiene instanciada a lo largo de todo el ciclo de vida de su aplicación. Permite mantener el estado global de su aplicación. Puede sobrecargar esta clase para:

- Salvaguardar variables globales de su aplicación.
- Gestionar los recursos que utiliza su aplicación.
- Declarar constantes globales para su aplicación.

Una vez se haya sobrescrito esta clase, debe declararla en su archivo de manifiesto para que se tome en consideración.

```
<application android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:name=".MyApplication">
```

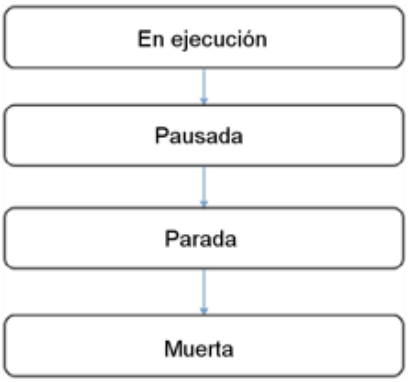
La clase Application se instanciará en el arranque de la aplicación.

Ciclo de vida de una actividad

Por defecto, cada aplicación Android se ejecuta en un proceso separado. Android gestiona los recursos disponibles en el dispositivo y puede, si fuera necesario, cerrar aplicaciones para liberar recursos (excepto aplicaciones en ejecución).

La elección de la aplicación que se cerrará depende fundamentalmente del estado del proceso en el que se encuentra. Si Android debe elegir entre dos aplicaciones que se encuentran en el mismo estado, elegirá la que se encuentre en este estado desde hace más tiempo.

1. Estado de una actividad



Una actividad puede encontrarse en cuatro estados distintos:

- **En ejecución**: la actividad se encuentra en primer plano y recibe las interacciones del usuario. Si el dispositivo necesita recursos, se matará la actividad que se encuentra en el fondo del back stack (consulte la sección Back stack).
- **Pausada**: la actividad está visible pero el usuario no puede interactuar con ella (oculta por un cuadro de diálogo, por ejemplo). La única diferencia con el estado anterior es la no recepción de eventos de usuario.
- **Parada**: la actividad ya no es visible pero sigue en ejecución. Todos los datos relativos a su ejecución se conservan en memoria. Cuando una actividad pasa a estar parada, debe guardar los datos importantes y detener todos los tratamientos en ejecución.
- **Muerta**: se ha matado la actividad, ya no está en ejecución y desaparece de la **back stack**. Todos los datos presentes en caché que no se hayan guardado se pierden.

2. Back stack

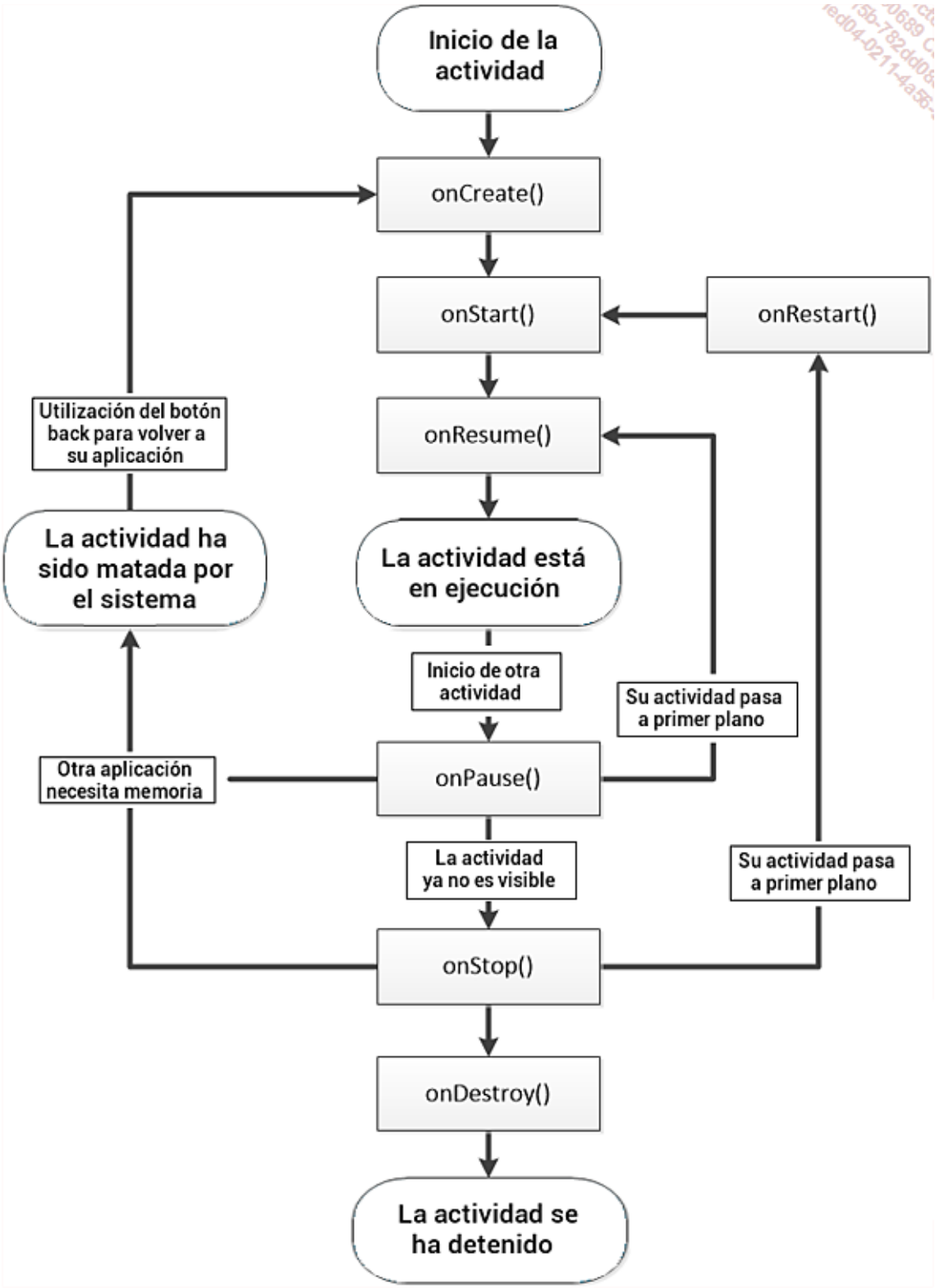
Como ocurre con un sitio de Internet, en una pantalla de Android se muestra una vista cada vez. La sucesión de páginas se almacena en una pila llamada back stack, lo que permite volver a la página anterior cuando se utiliza el botón "volver" del teléfono (similar al botón de volver atrás en un navegador web).

Todas las actividades iniciadas se almacenan en la **back stack**. Cuando se inicia una actividad nueva, ésta se añade en la cima de la back stack.

Si el usuario pulsa el botón ”atrás” de su dispositivo, la actividad en curso se cierra y la que se encontraba inmediatamente debajo en la back stack se abre.

3. Ciclo de vida

El ciclo de vida de una actividad es bastante complejo y su comprensión es indispensable en el desarrollo para Android. El siguiente esquema resume este ciclo de vida:



Cuando se inicia la actividad, se invoca al método **onCreate**. En este método, debe inicializar su vista y vincular los datos a una lista. Este método recibe como parámetro un Bundle que contiene el estado anterior de la actividad.

A esta llamada le sigue el método **onStart**, que permite indicar el inicio efectivo de la aplicación (ahora ya es visible).

A continuación, se invoca al método **onResume** para ejecutar todos los tratamientos necesarios para el funcionamiento de la actividad (thread, proceso, tratamiento), inicializar variables y listeners. Estos tratamientos deberán detenerse cuando se invoque al método **onPause** y relanzarse, si fuera necesario, cuando se produzca una futura llamada al método **onResume**.

Después de estas tres llamadas, la actividad se considera utilizable y puede recibir interacciones de los usuarios.

Si otra actividad pasa a primer plano, la actividad en ejecución pasará a estar pausada. Justo antes de la llamada al método **onPause**, se invocará al método **onSaveInstanceState** para permitirle guardar los datos importantes de la actividad. Estos datos podrán aplicarse a futuras ejecuciones de la actividad con la llamada al método **onRestoreInstanceState** o la llamada a **onCreate**.

El método **onPause** permite detener los tratamientos realizados (tratamiento no necesario si la actividad no es visible) por la actividad (tratamiento, thread, proceso).

Si su actividad pasa a estar visible de nuevo, se realizará una llamada al método **onResume**.

El paso de la actividad al estado ”parada” tiene asociado una llamada al método **onStop**. En este método hay que detener todos los tratamientos restantes.

Una vez parada, su actividad puede:

- **Volver a iniciarse**: acción realizada por una llamada al método **onStart** siguiendo el ciclo de vida normal de la actividad.
- **Terminar**: se realiza con una llamada al método **onDestroy**, en el que deberá parar todos los tratamientos restantes, cerrar todas las conexiones a la base de datos, todos los threads, todos

los archivos abiertos, etc. Puede provocar la llamada al método **onDestroy** utilizando el método **finish**.

➔ Es posible sobrecargar todos los métodos del ciclo de vida de una actividad.

Manifiesto

El archivo **AndroidManifest.xml**, que se encuentra en la raíz de cualquier proyecto Android, es lo más parecido a una descripción completa de la aplicación (componentes, actividades, servicios, permisos, proveedores de contenido, etc.).

➔ Cada componente que requiera una declaración en el manifiesto se abordará en la sección dedicada a dicho componente o funcionalidad.

Vamos a describir las secciones generales e indispensables de un archivo de manifiesto. El comienzo de un archivo de manifiesto es, con algunos detalles, casi idéntico al de cualquier otra aplicación.

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.eni.android.hello"
...
</manifest>
```

Esta primera parte contiene:

- La declaración xmlns (obligatoria).
- El package (identificador) de su aplicación (elegido en la creación de su proyecto).

El contenido del manifiesto varía en función de la aplicación:

- **permissions** (véase la sección Permissions (permisos) en este capítulo).

- **instrumentations**: permiten activar la supervisión de las interacciones entre la aplicación y el sistema mediante pruebas.
- **uses-configuration**: este atributo permite detallar las especificidades de navegación compatibles con su aplicación, tales como los teclados físicos, trackball, lápiz, etc. Este nodo es opcional.
- **uses-feature**: este nodo permite indicar los elementos de hardware requeridos por la aplicación (audio, bluetooth, cámara, GPS, localización, NFC, etc.). Este nodo es opcional.
- **supports-screens**: este atributo permite especificar las dimensiones de pantalla soportadas por la aplicación (smallScreens, normalScreens, largeScreens, xlargeScreens).
- **La aplicación**: define la estructura y el contenido de la aplicación (actividad, servicio, etc.).

En la sección application del manifiesto tendrá que describir todas las características específicas y todos los componentes de su aplicación:

- Nombre de la aplicación.
- Icono de la aplicación.
- Tema utilizado por la aplicación.
- Los distintos componentes de la aplicación (Actividad, Servicio, Proveedor de contenido, Receptor de eventos) así como las características específicas de cada componente (componente principal, presencia en el inicio de la aplicación, uso de datos).

Permissions (permisos)

Una de las características específicas de Android está relacionada con el sistema de permisos. Permite, a cada aplicación, definir los permisos que desea tener.

Cada *permission* se corresponde con el derecho de acceso a una funcionalidad o a un dato.

Por defecto, las aplicaciones no tienen ninguna *permission* y deben solicitar una *permission* cada vez que lo requieran. Por ejemplo, esto permite al usuario saber la lista de todas las *permissions* que solicita una aplicación antes de instalarla.

También puede crear *permissions* para que otras aplicaciones tengan el derecho de utilizar sus datos o sus servicios, etc.

→ Las *permissions* se declaran fuera de la etiqueta **application** de su manifiesto.

1. Utilizar una permission

Solicitar una *permission* es muy fácil: se añade la etiqueta **<uses-permission>** en el archivo **AndroidManifest.xml**.

Si desea utilizar una conexión a Internet en su aplicación, deberá utilizar la siguiente línea en su manifiesto:

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

Todas las *permissions* de sistema empiezan por **android.permission**, puede encontrarlas en esta dirección:http://developer.android.com/reference/android/Manifest.permission.html

→ Si, por ejemplo, su aplicación utiliza la conexión a Internet sin solicitar la *permission*, la excepción **SecurityException** le informará de que le falta declarar la *permission* en su manifiesto.

2. Declarar sus permissions

También puede declarar sus propias *permissions*. La declaración de una *permission* se realiza en el archivo**AndroidManifest.xml** mediante la etiqueta **<permission>**.

Para declarar una *permission*, necesitará tres datos:

- **El nombre de su permission**: para evitar un conflicto de nombres, prefije su *permission* con el identificador del package de la aplicación.
- **Una etiqueta para su permission**: un texto corto y comprensible que indicará el título de la *permission*.
- **Una descripción de la permission**: un texto más completo para explicar con mayor claridad los datos accesibles mediante la *permission*.

A continuación se muestra un ejemplo:

```
<permission
```

```
android:name="com.eni.android.permission.MI_PERMISSION"

android:label="@string/label_permission"

        android:description="@string/desc_permission "
```

Y, a continuación, las cadenas de caracteres que sirven de etiqueta y de descripción para esta nueva *permission*:

```
<resources>

    <string name = "label_permission"> Etiqueta de

        mi permission </string>

    <string name = "desc_permission"> Descripción

        de mi permission </string>

</resources>
```

También debe detectar las violaciones de seguridad en el uso de *permissions* que haya definido.

Si desea que el uso de un dato, actividad o servicio en particular de su aplicación, requiera la declaración de una *permission*, puede indicarlo en su archivo de manifiesto, en cualquier parte donde se exija la *permission* con el atributo **android:permission**.

Tomemos, por ejemplo, el caso de un servicio que obliga a declarar una *permission* a la aplicación que desea utilizarlo:

```
<service android:name="com.eni.android.MiServicio"

android:permission="com.eni.android.permission.MI_PERMISSION">
```

En este ejemplo, cualquier aplicación que desee utilizar el servicio "MiServicio" debe declarar, obligatoriamente, la *permission* "MI_PERMISSION".

Si su *permission* involucra a un proveedor de contenido (content provider), puede distingir la *permission* de lectura de la *permission* de escritura (**android:readPermission** y **android:writePermission**).

También puede comprobar el uso de su *permission* directamente en el código de su servicio, por ejemplo. Para ello, utilice el método **checkCallingPermission()** que devolverá, respectivamente, PERMISSION_GRANTED, si la *permission* se ha solicitado correctamente, o PERMISSION_DENIED, en caso contrario.

➔ Todos los errores de *permission* aparecen en tiempo de ejecución, y no en la compilación.