Desarrollo Android

Arkaitz Garro

Master en desarrollo de Apps Móviles

Componentes

Manifest, resources, Activity

Una aplicación Android

Consiste en una serie de componentes, ligados por el archivo Manifest, el cual describe su comportamiento y su interacción

Componentes

Activity

Service

Content Provider

Intent

Broadcast receiver

Widgets

Notifications

AndroidManifest.xml

Toda aplicación Android lo incluye, en la raíz del proyecto

Define la estructura, metadatos, sus componentes y requerimientos

Incluye un nodo por cada Activity, Service, Content Provider y Broadcast Receiver

Gracias a los Intent Filters y permisos podemos determinar cono interactúan entre ellos y con otras aplicaciones

Puede incluir metadata como el icono, la versión, el theme, requisitos de ejecución...

AndroidManifest.xml

Es posible definir dónde se va a instalar la aplicación, utilizando el atributo `installLocation`

Si no se especifica la localización, se instala en la memoria interna

Si se instala en la memoria externa, la aplicación es terminada cuando montamos el dispositivo como USB storage

Dependiendo del tipo de aplicación, no es muy recomendable

http://developer.android.com/guide/topics/manifest/manifest-intro.html

AndroidManifest.xml

Un vistazo al Manifiesto

```
<manifest xmlns:android=”http://schemas.android.com/apk/res/android”
    package=“com.paad.myapp“
    android:versionCode=“1“
    android:versionName=“0.9 Beta“
    android:installLocation=“preferExternal“>
    [ ... manifest nodes ... ]
</manifest>
```

AndroidManifest.xml

Posibles atributos

`uses-sdk`: permite definir el mínimo y máxima versión del API a utilizar por nuestra aplicación.

- **`minSdkVersion`**: versión mínima requerida del SDK
- **`maxSdkVersion`**: versión máxima soportada del SDK
- **`targetSdkVersion`**: especifica la plataforma contra la que se ha desarrollado y testeado la aplicación

```
<uses-sdk android:minSdkVersion=”6” android:targetSdkVersion=”15”/>
```

AndroidManifest.xml

Posibles atributos

`uses-configuration`: especifica la combinación de mecanismos de entrada soportados.

- `reqFiveWayNav`: requiere trackball o pad direccional
- `reqHardKeyboard`: requiere teclado físico
- `reqKeyboardType`: tipo de teclado requerido, `nokeys, qwerty, twelvekey, or undefined.`
- `reqNavigation`: tipo de pad requerido, `nonav, dpad, trackball, wheel, or undefined`
- `reqTouchScreen`: tipo de interacción, `notouch, stylus, finger, or undefined`

AndroidManifest.xml

```
<uses-configuration android:reqTouchScreen=”finger”
                    android:reqNavigation=”trackball”
                    android:reqHardKeyboard=”true”
                    android:reqKeyboardType=”qwerty”/>
<uses-configuration android:reqTouchScreen=”finger”
                    android:reqNavigation=”trackball”
                    android:reqHardKeyboard=”true”
                    android:reqKeyboardType=”twelvekey”/>
```

AndroidManifest.xml

`uses-feature`: necesidad de alguna característica hardware.

```
bluetooth, camera, location, microphone, NFC,
sensors, telephony, touchscreen, USB, Wi-Fi
```

AndroidManifest.xml

```
<uses-feature android:name=”android.hardware.camera” />
<uses-feature android:name=”android.hardware.camera.autofocus”
          android:required=”false” />
<uses-feature android:name=”android.hardware.camera.flash”
          android:required=”false” />
```

AndroidManifest.xml

`supports-screens`: especifica los tamaños de pantalla para los que la aplicación ha sido diseñada y testeada

- `smallScreens`: tamaño de pantalla QVGA
- `normalScreens`: tamaño de pantalla WVGA, WQVGA
- `largeScreens`: tamaño de pantalla mayor que un teléfono
- `xlargeScreens`: tamaño de pantalla mayor que un tablet
- `requiresSmallestWidthDp`: ancho mínimo en px
- `compatibleWidthLimitDp`: ancho máximo escalable
- `largestWidthLimitDp`: ancho máximo compatible

AndroidManifest.xml

```
<supports-screens android:smallScreens=”false”
                  android:normalScreens=”true”
                  android:largeScreens=”true”
                  android:xlargeScreens=”true”
                  android:requiresSmallestWidthDp=”480”
                  android:compatibleWidthLimitDp=”600”
                  android:largestWidthLimitDp=”720”/>
```

Optimizar las aplicaciones todo tipo de resoluciones, en lugar de restringir las resoluciones

AndroidManifest.xml

`uses-permission`: permisos de usuario requeridos por la aplicación.

```
<uses-permission android:name=”android.permission.ACCESS_FINE_LOCATION”/>
```

`permission`: permite definir permisos propios de la aplicación.

```
<permission android:name=”com.paad.DETONATE_DEVICE”
            android:protectionLevel=”dangerous”
            android:label=”Self Destruct”
            android:description=”@string/detonate_description”>
</permission>
```

AndroidManifest.xml

`application`: un manifiesto solo puede contener un nodo de este tipo. Es utilizado para definir todos los metadatos de la aplicación. También funciona como contenedor para los componentes `Activity`, `Service`, `Content Provider` y `Bradcast Receiver`.

```
<application android:icon=”@drawable/icon”
             android:logo=”@drawable/logo”
             android:theme=”@android:style/Theme.Light”
             android:name=”.MyApplicationClass”
             android:debuggable=”true”>
             [ ... application nodes ... ]
</application>
```

AndroidManifest.xml

`activity`: esta etiqueta es requerida por cada actividad existente en la aplicación. Si se intenta lanzar una actividad no declarada, se produce una excepción.

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

AndroidManifest.xml

`service`: al igual que la etiqueta **`activity`**, es necesario añadir una etiqueta **`service`** por cada servicio existente en la aplicación.

```
<service android:name=”.MyService”>

</service>
```

AndroidManifest.xml

`provider`: etiqueta que especifica cada uno de los proveedores de contenidos de la aplicación.

```
<provider android:name=”.MyContentProvider”
  android:authorities=”com.paad.myapp.MyContentProvider”/>
```

AndroidManifest.xml

`receiver`: es posible registrar un **`Broadcast Receiver`**, el cual se iniciará de manera automática cuando se lance un evento que coincida con los filtros especificados.

```
<receiver android:name=”.MyIntentReceiver”>
  <intent-filter>
    <action android:name=”com.paad.mybroadcastaction” />
  </intent-filter>
</receiver>
```

AndroidManifest.xml

`uses-library`: utilizado para especificar las librerías de terceros utilizadas por la aplicación. Se especifican los paquetes en particular, y la aplicación no se instala si no existen dichos paquetes en el dispositivo.

```
<uses-library android:name=”com.google.android.maps”
    android:required=”false”/>
```

Editor de AndroidManifest.xml

Componentes

Manifest, resources, Activity

Recursos

Android permite externalizar recursos como imágenes, textos, colores, animaciones, temas...

Permite separar el código de la vista, y mejorar aspectos como la internacionalización o el soporte para varios dispositivos.

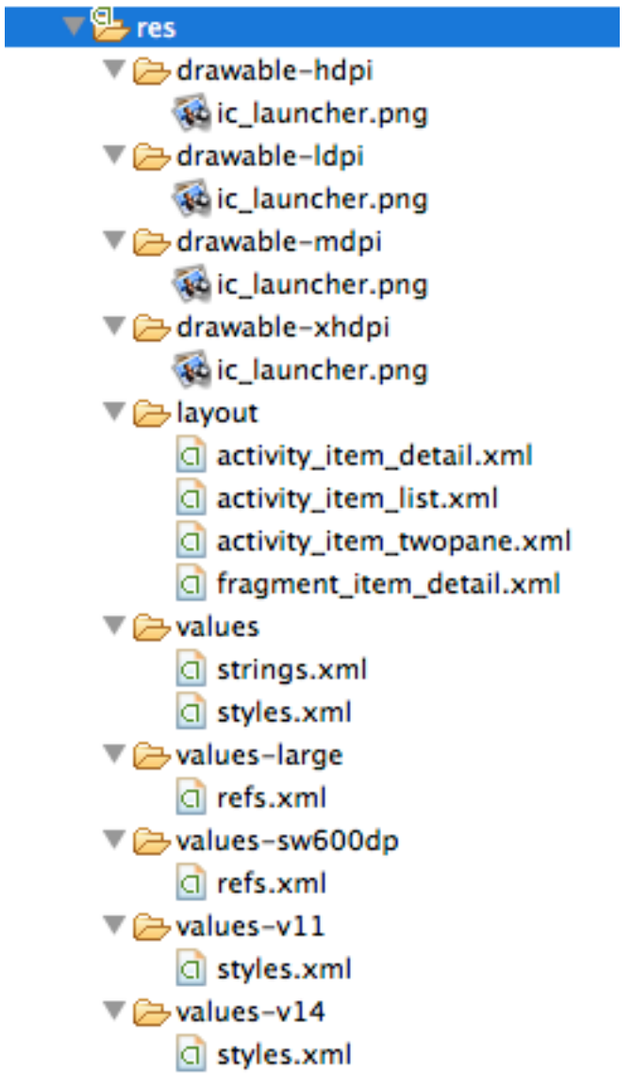
Muy interesante de cara a la interfaz

Definir diferentes layouts para diferentes resoluciones

Imágenes para diferentes densidades de pantalla

Textos/imágenes para diferentes idiomas

Recursos



Los recursos de la aplicación se almacenan en el dir. `/res`

Cada tipo de recurso se almacena en un subdirectorio, dependiendo de su tipo

Al crear la aplicación, estos recursos se comprimen, para mejorar la eficiencia

Valores simples: /res/values

Valores simples a almacenar pueden ser: cadenas de caracteres, colores, medidas, estilos y arrays de strings y enteros

Cada uno de estos tipos es definido por su propia etiqueta

Lo ideal es separar cada tipo de valor por ficheros

Valores simples: /res/values

```
<?xml version=”1.0” encoding=”utf-8”?>
<resources>
  <string name=”app_name”>To Do List</string>
  <plurals name=”androidPlural”>
    <item quantity=”one”>One android</item>
    <item quantity=”other”>%d androids</item>
  </plurals>
  <color name=”app_background”>#FF0000FF</color>
  <dimen name=”default_border”>5px</dimen>
  <string-array name=“string_array“>
    <item>Item 1</item>
    <item>Item 2</item>
    <item>Item 3</item>
  </string-array>
  <array name=“integer_array“>
    <item>3</item>
    <item>2</item>
    <item>1</item>
  </array>
</resources>
```

Estilos y temas

Permiten disponer de un “look&feel” consistente en toda la aplicación

El uso más común es almacenar colores y fuentes

Cada estilo es definido por un nombre y unas propiedades

```
<?xml version=”1.0” encoding=”utf-8”?>
<resources>
  <style name=”base_text”>
    <item name=”android:textSize”>14sp</item>
    <item name=”android:textColor”>#111</item>
  </style>
</resources>
```

Layout

Los Layout son la manera de separar la lógica de negocio de la vista

Es posible crear diferentes vistas para diferentes dispositivos, manteniendo la misma lógica

```
<?xml version=”1.0” encoding=”utf-8”?>
<LinearLayout xmlns:android=”http://schemas.android.com/apk/res/android”
  android:orientation=”vertical”
  android:layout_width=”fill_parent”
  android:layout_height=”fill_parent”>
  <TextView
    android:layout_width=”fill_parent”
    android:layout_height=”wrap_content”
    android:text=”@string/hello”
  />
</LinearLayout>
```

Animaciones

Android soporta tres tipos de animaciones:

- `Property animations`: permitir animaciones en las propiedades de los objetos; color, tamaño…
- `View animations`: transiciones de rotación, movimiento
- `Frame animations`: utilizado para mostrar secuencias de imágenes

Definir animaciones de esta manera, permite utilizar las mismas secuencias en toda la aplicación, o decidir cual utilizar según el dispositivo

Menús

En la medida de lo posible, es recomendable crear los menús como un recurso, no directamente en el código

```
<?xml version=”1.0” encoding=”utf-8”?>
<menu xmlns:android=”http://schemas.android.com/apk/res/android”>
  <item android:id=”@+id/menu_refresh”
    android:title=”@string/refresh_mi” />
  <item android:id=”@+id/menu_settings”
    android:title=”@string/settings_mi” />
</menu>
```

Cómo utilizar los recursos

Los recursos pueden ser accedidos desde el código o desde otros recursos

Código

```
// Inflate a layout resource.
setContentView(R.layout.main);
// Display a transient dialog box that displays the
// error message string resource.
Toast.makeText(this, R.string.app_error, Toast.LENGTH_LONG).show();
```

Recursos

```
attribute="@[packagename:]resourcetype/resourceidentifier"
```

Cómo utilizar los recursos

Android ofrece sus propios recursos que podemos utilizar

```
<EditText android:id=”@+id/myEditText”
    android:layout_width=”match_parent”
    android:layout_height=”wrap_content”
    android:text=”@android:string/httpErrorBadUrl”
    android:textColor=”@android:color/darker_gray”
/>
```

```
CharSequence httpError = getString(android.R.string.httpErrorBadUrl);
```

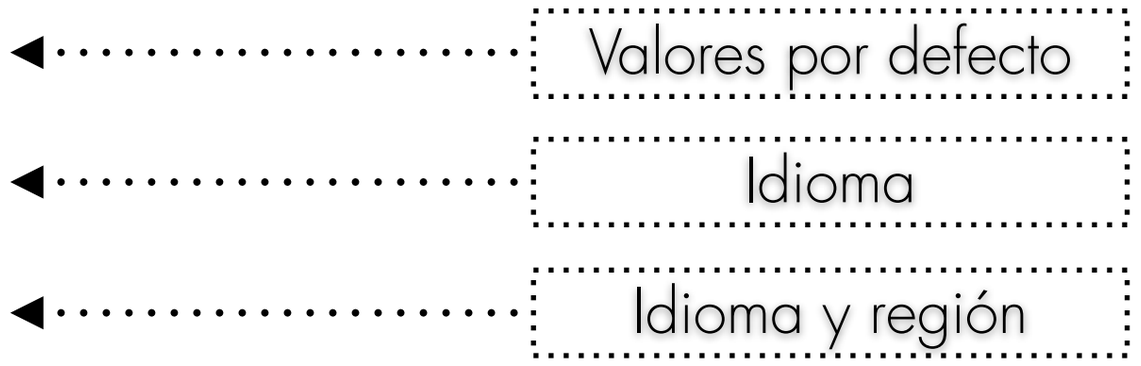
Crear recursos dependientes

Android nos permite definir diferentes recursos según el idioma, localización y hardware

El recurso que mejor se adapta se selecciona dinámicamente en tiempo de ejecución

Las condiciones para seleccionar uno u otro recurso se especifican con un guión medio (-)

```
Project/
  res/
    values/
      strings.xml
    values-es/
      strings.xml
    values-es-rAR/
      strings.xml
```



Selectores de recursos

Android dispone de los siguientes selectores de recursos

- **Mobile Country Code / Mobile Network Code (MCC/MNC)**: código de país asociado a la tarjeta SIM.

  http://en.wikipedia.org/wiki/Mobile_network_code

- **Idioma y región**: representados en formato ISO, precedido con una **r** para la región.

- **Ancho mínimo de pantalla**: dimensiones mínimas (alto o ancho) de la pantalla para mostrar los recursos asociados. Representado de la siguiente manera: `sw<dimension>dp`

- **Ancho disponible**: dimensiones mínimas (ancho) para mostrar los recursos. Cambia cuando la orientación del dispositivo cambia. `w<dimension>dp`

Selectores de recursos

- **Alto disponible**: dimensiones mínimas (alto) para mostrar los recursos. Cambia cuando la orientación del dispositivo cambia. `h<dimension>dp`
- **Orientación de pantalla**: puede ser `port` (portrait), `land` (lanscape) o `square`.
- **Modo dock**: puede ser `car` o `desk`.
- **Modo noche**: puede ser `night` o `nonight`.
- **Densidad de pixels**: en puntos por pulgada (dpi); `ldpi`, `mdpi`, `hdpi` o `xhdpi`.

Es importante mantener el orden de los selectores, si se utilizan varios a la vez

Cambios de configuración

Android reacciona ante los cambios de idioma, localización y hardware reiniciando la actividad activa

Es posible detectar estos cambios, y actuar en consecuencia, añadiendo en el manifiesto los cambios que se quieren controlar

```
<activity
  android:name=”.MyActivity”
  android:label=”@string/app_name”
  android:configChanges=”screenSize|orientation|keyboardHidden”>
  <intent-filter >
    <action android:name=”android.intent.action.MAIN” />
    <category android:name=”android.intent.category.LAUNCHER” />
  </intent-filter>
</activity>
```

Cambios de configuración

Es posible especificar los siguientes cambios de configuración:

`mcc, mnc, locale, keyboardHidden, keyboard, fontScale, uiMode, orientation, screenLayout, screenSize, smallestScreenSize`

Al especificar cualquier atributo, evitamos el reinicio de la actividad, para esos cambios en concreto

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
  super.onConfigurationChanged(newConfig);
  // [ ... Update any UI based on resource values ... ]
  if (newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE)
  {
     // [ ... React to different orientation ... ]
  }
  if (newConfig.keyboardHidden == Configuration.KEYBOARDHIDDEN_NO) {
     // [ ... React to changed keyboard visibility ... ]
  }
}
```

Ejercicio

Componentes

Manifest, resources, Activity

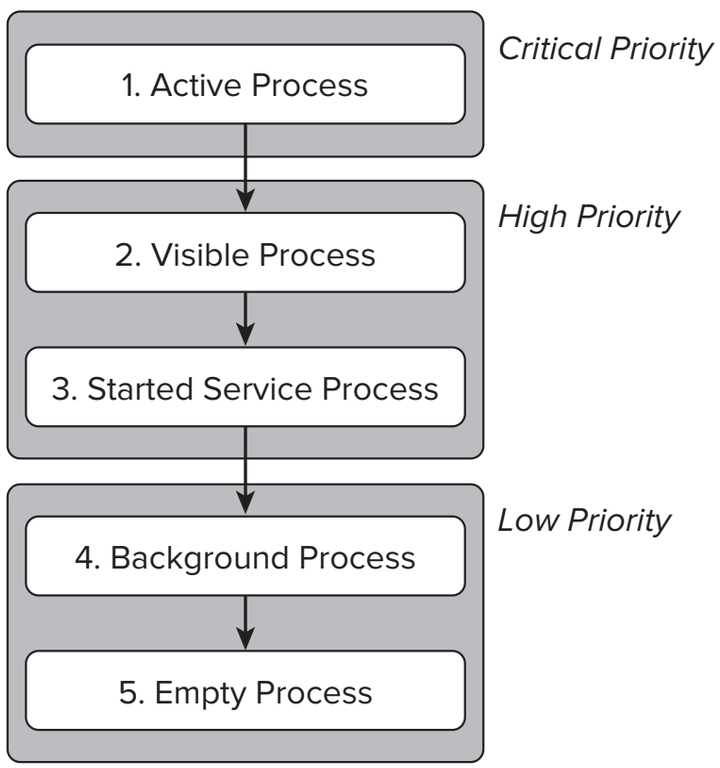
Ciclo de vida de la aplicación

Android maneja los recursos de una manera muy **agresiva**

Las aplicaciones son cerradas según su prioridad

La prioridad viene definida por el componente de mayor prioridad

También está basada en la interdependencia entre procesos



Critical Priority

Active processes: componentes de la aplicación con los que el usuario puede estar interactuando:

- **Actividades** en estado activo
- **Broadcast Receivers** ejecutando el método `onReceive`
- **Servicios** ejecutando los métodos `onStart`, `onCreate` o `onDestroy`
- **Servicios** en ejecución que han sido marcados para correr en primer plano

High Priority

Visible processes: son procesos visibles, pero que no se encuentran en primer plano (tras una ventana modal, una actividad transparente...) y no interactúan con el usuario.

Started Service processes: servicios iniciados por actividades, y que se encuentran en ejecución. Cuando el sistema termina un servicio, intenta reiniciarlo cuando hay recursos disponibles nuevamente.

Low Priority

Background processes: procesos que corresponden con actividades, y que no están ejecutando con ningún servicio. Android los terminará utilizando el patrón **last-seen-first-killed**

Empty processes: para mejorar el rendimiento, Android almacena en memoria las aplicaciones que han finalizado su ciclo de vida. Estos son los primeros recursos en liberarse.

Android Application Class

El objeto `Application` es instanciado cada vez que la aplicación se ejecuta

A diferencia de las actividades, la aplicación no se reinicia como consecuencia del cambio de configuración

Podemos utilizar esta clase para lo siguiente:

Responder a mensajes a nivel de aplicación, como restricciones de memoria

Transferir objetos entre componentes de la aplicación

Gestionar y mantener recursos utilizados por diversos componentes de la aplicación

Extendiendo Application Class

Esqueleto de la clase `Application`

```
import android.app.Application;
import android.content.res.Configuration;

public class MyApplication extends Application {
  private static MyApplication singleton;

  // Returns the application instance
  public static MyApplication getInstance() {
    return singleton;
  }

  @Override
  public final void onCreate() {
    super.onCreate();
    singleton = this;
  }
}
```

Extendiendo Application Class

No hay que olvidarse de registrar la nueva clase `Application` en el manifiesto de Android

```
<application android:icon=”@drawable/icon”
             android:name=”.MyApplication”>
  [... Manifest nodes ...]
</application>
```

Extendiendo Application Class

Implementar comportamientos específicos para cada evento

`onCreate`: se llama al iniciar la aplicación. Válido para inicializar variables y recursos a utilizar por toda la aplicación.

`onLowMemory`: una oportunidad para liberar memoria en caso estar bajos de recursos. Es llamado cuando todos los anteriores procesos han sido eliminado, y todavía se necesita más memoria.

`onTrimMemory`: es llamado cuando Android determina que el proceso debe liberar memoria.

`onConfigurationChanged`: recalcular los valores dependientes de configuración.

Componentes

Manifest, resources, Activity

Activity

Cada actividad representa una pantalla en la interfaz de usuario

Incluye los elementos de la interfaz, que permiten la interacción con el usuario

Para movernos entre pantallas, es necesario crear una nueva actividad, o volver a una anterior

La mayoría de las actividades ocupan toda la pantalla, pero pueden no hacerlo o ser semitransparentes

Crear una Activity

Extender la clase `Activity` de Android para crear una nueva actividad

```
package com.paad.activities;
import android.app.Activity;
import android.os.Bundle;

public class MyActivity extends Activity {
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }
}
```

Crear una Activity

Es necesario registrar la actividad en el manifiesto

Si es la actividad principal, hay que indicar los `Intents` sobre los que debe reaccionar

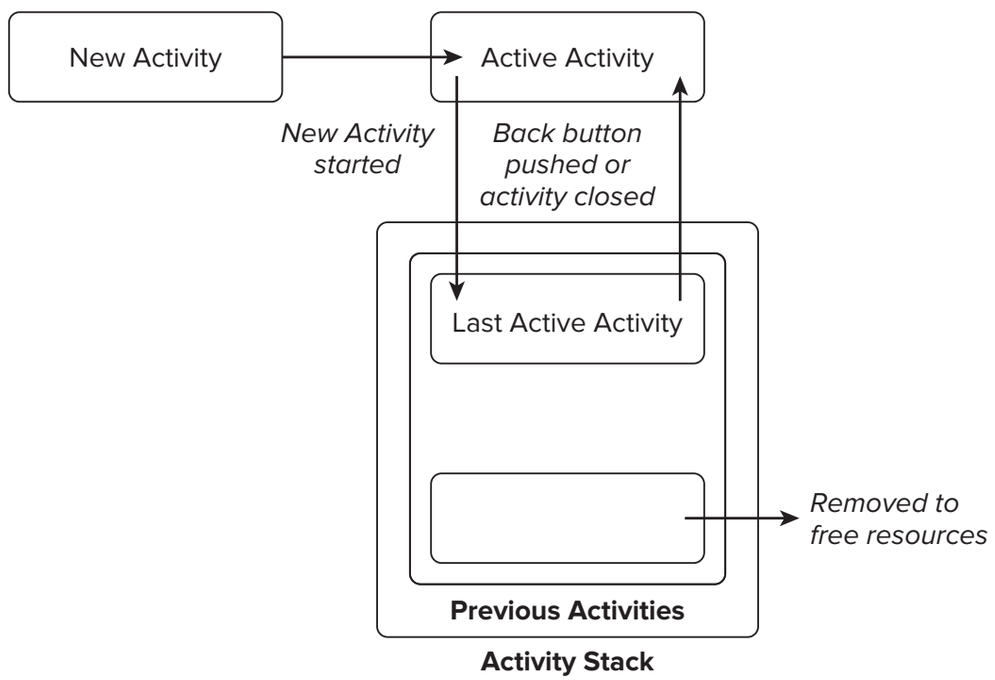
```
<activity android:label=”@string/app_name”
  android:name=”.MyActivity”>
  <intent-filter>
      <action android:name=”android.intent.action.MAIN” />
      <category android:name=”android.intent.category.LAUNCHER” />
  </intent-filter>
</activity>
```

Ciclo de vida: pila de actividades

El estado de cada actividad es determinado por su posición en la pila de actividades

Cuando una actividad es iniciada, se activa y pasa a lo alto de la pila

Si se vuelve atrás, o una nueva actividad de activa, la actividad actual deja paso a la nueva



Ciclo de vida: estados

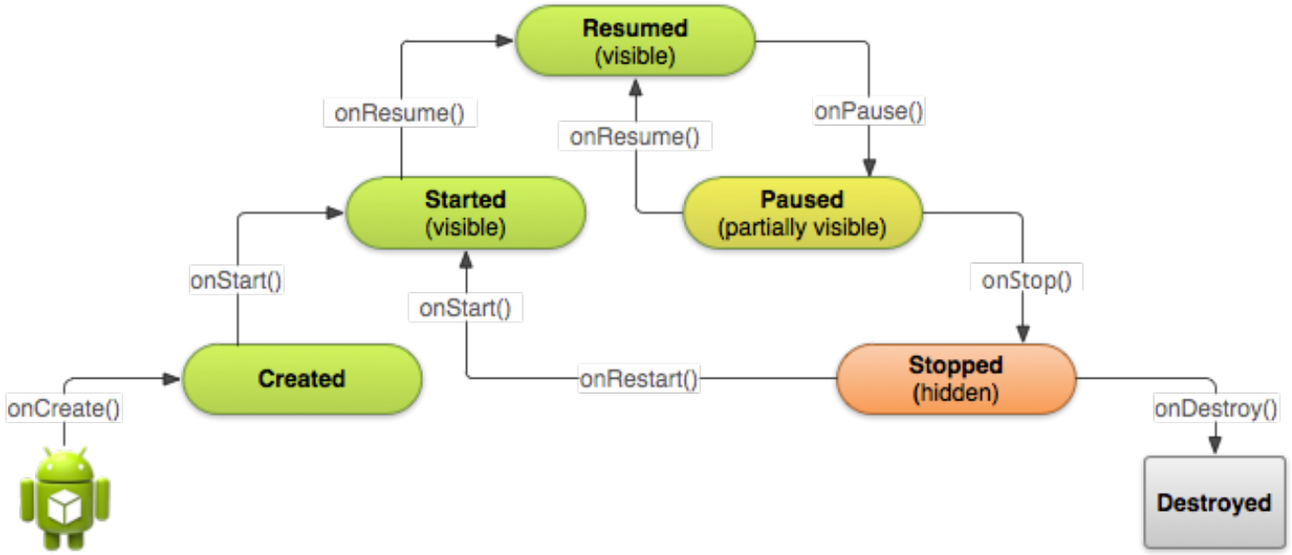
Active: se encuentra en lo alto de la pila, es visible, tiene el foco e interactúa con el usuario. Cuando otra actividad se convierte en activa, esta pasa a estar en pausa.

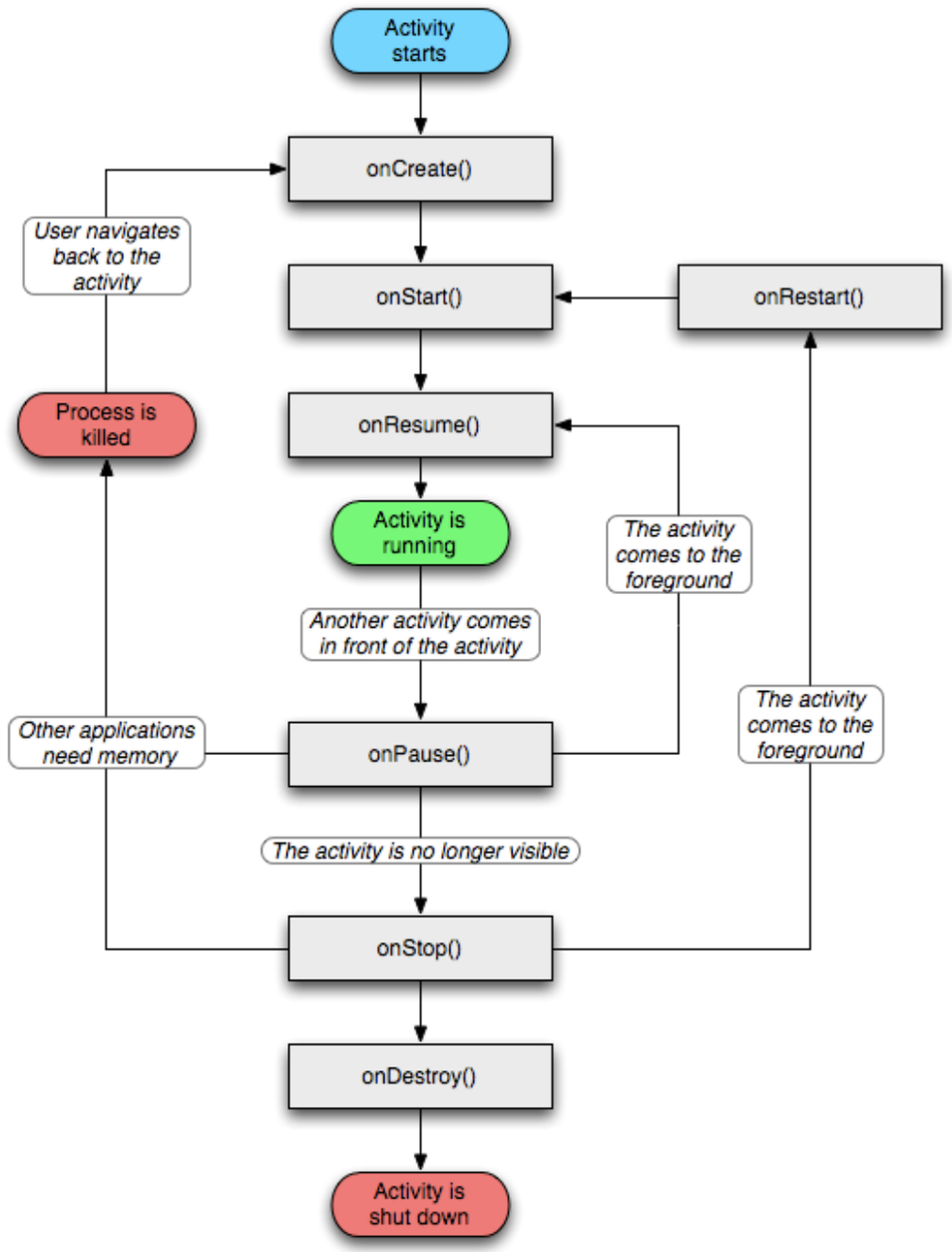
Paused: todavía visible, pero sin interacción con el usuario.

Stoped: cuando una actividad no es visible, se "para". Se mantiene en memoria con toda la información, aunque es importante guardarla, por si Android decide terminarla.

Inactive: la actividad ha sido terminada y sacada de la pila de actividades. Necesita ser reiniciada para poder ser usada.

Ciclo de vida: cambios de estado





Ciclo de vida: cambios de estado

```
// Called at the start of the full lifetime.
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  // Initialize Activity and inflate the UI.
}

// Called after onCreate has finished, use to restore UI state
@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
  super.onRestoreInstanceState(savedInstanceState);
  // Restore UI state from the savedInstanceState.
  // This bundle has also been passed to onCreate.
}
```

Ciclo de vida: cambios de estado

```
// Called before subsequent visible lifetimes
// for an Activity process.
@Override
public void onRestart(){
  super.onRestart();
  // Load changes knowing that the Activity has already
  // been visible within this process.
}

// Called at the start of the visible lifetime.
@Override
public void onStart(){
  super.onStart();
  // Apply any required UI change now that
  // the Activity is visible.
}
```

Ciclo de vida: cambios de estado

```
// Called at the start of the active lifetime.
@Override
public void onResume(){
  super.onResume();
  // Resume any paused UI updates, threads, or processes required
  // by the Activity but suspended when it was inactive.
}
```

```
// Called to save UI state changes at the
// end of the active lifecycle.
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
  // Save UI state changes to the savedInstanceState.
  // This bundle will be passed to onCreate and
  // onRestoreInstanceState if the process is
  // killed and restarted by the run time.
  super.onSaveInstanceState(savedInstanceState);
}
```

Ciclo de vida: cambios de estado

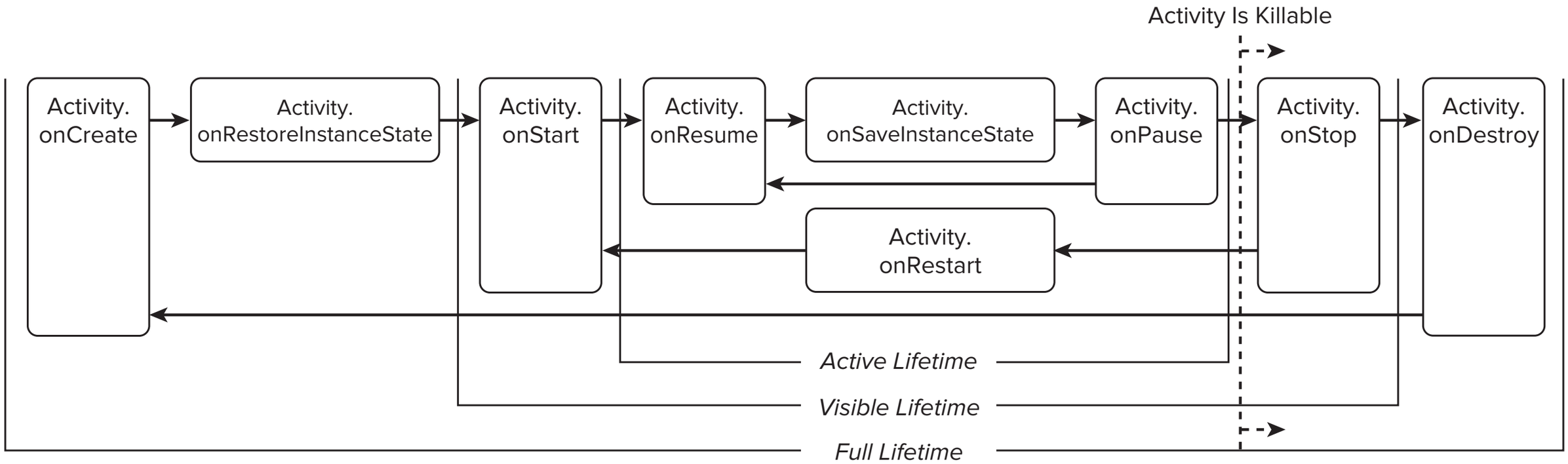
```
// Called at the end of the active lifetime.
@Override
public void onPause(){
  // Suspend UI updates, threads, or CPU intensive processes
  // that don’t need to be updated when the Activity isn’t
  // the active foreground Activity.
  super.onPause();
}

// Called at the end of the visible lifetime.
@Override
public void onStop(){
  // Suspend remaining UI updates, threads, or processing
  // that aren’t required when the Activity isn’t visible.
  // Persist all edits or state changes
  // as after this call the process is likely to be killed.
  super.onStop();
}
```

Ciclo de vida: cambios de estado

```
// Sometimes called at the end of the full lifetime.
@Override
public void onDestroy(){
  // Clean up any resources including ending threads,
  // closing database connections etc.
  super.onDestroy();
}
```

Ciclo de vida



Ciclo de vida: tips

No es inusual que una actividad se termine sin llamar al método `onDestroy`

Utilizar el método `onCreate` para inicializar la actividad

Crear la interfaz de usuario

Variables estáticas de la aplicación

Asignar datos a los controles

Si la actividad ha sido finalizada de manera inesperada, en bundle **`savedInstanceState`** contiene la última información

Implementar el método `onDestroy` para cerrar todas las conexiones

Ciclo de vida: tips

El método `onStop` es ideal para parar animaciones, escuchadores de sensores, GPS, servicios... u otros procesos utilizados para actualizar la UI

Utilizar `onStart (onRestart)` para volver a lanzar estos procesos

Tratar de mantener el código lo más "ligero" posible en los métodos `onPause` y `onResume`

Ciclo de vida: tips

Justo antes de `onPause`, se llama al método `onSaveInstanceState`, permitiendo almacenar en el bundle la información actual de la UI

Siempre se llama a este método antes de terminar una actividad

Checkbox, foco del usuario, texto introducido...

Utilizar esta información en el método **`onResume`**, para devolver la actividad al mismo estado

Ejercicio