

# Tecnologías y Desarrollo en Dispositivos Móviles

## **Apartado 4:**

Componentes de una app.  
Ciclo de vida de actividades.

### **Autores:**

Víctor M. Rivas Santos  
(Juan Ruiz de Miras)

# Componentes de una aplicación

- Actividades
- Servicios
- Proveedores de contenidos
- Widgets
- Intents

# Actividades

- Cada una de las “pantallas” de tu aplicación
- Suele abarcar toda la pantalla...
- ... pero también puede ser más pequeña y quedar “flotando” delante de las otras.

```
public class MainActivity extends Activity {  
  
    public static final int CONTACT_QUERY_LOADER = 0;  
    public static final String QUERY_KEY = "query";  
}
```

# Servicios

- Realizan operaciones de larga duración en segundo plano, sin interfaz de usuario
- Ejemplos: transacciones de red, sonido, I/O sobre ficheros...

```
public class LocalService extends Service {  
    private NotificationManager mNM;  
  
    // Unique Identification Number for the Notification.  
    // We use it on Notification start, and to cancel it.  
    private int NOTIFICATION = R.string.local_service_started;
```

# Proveedores de contenidos

- Administran el acceso a un conjunto de datos estructurado
  - Ejemplos: calendario, contactos, copiar&pegar...

```
private static List<Event> queryEvents(Context context, long beginTime, long
endTime) {
    ContentResolver contentResolver = context.getContentResolver();
    Uri.Builder builder =
CalendarContract.Instances.CONTENT_URI.buildUpon();
    ContentUris.appendId(builder, beginTime);
    ContentUris.appendId(builder, endTime);
```

# Widgets

- Miniaturas de aplicación que pueden empotrarse en otras aplicaciones
  - Ejemplos típicos: reproductor de música, reloj, tiempo meteorológico...

```
public class ExampleAppWidgetProvider extends AppWidgetProvider {  
  
    public void onUpdate(Context context  
        , AppWidgetManager appWidgetManager  
        , int[] appWidgetIds) {  
        final int N = appWidgetIds.length;
```

# Intents

- Enlazan los componentes anteriores
- Permiten a un componente solicitar la acción de otro componente
- Almacenan información que Android usa para identificar qué **componente** iniciar, indicándole qué **acción** debe realizar y sobre qué **datos**.

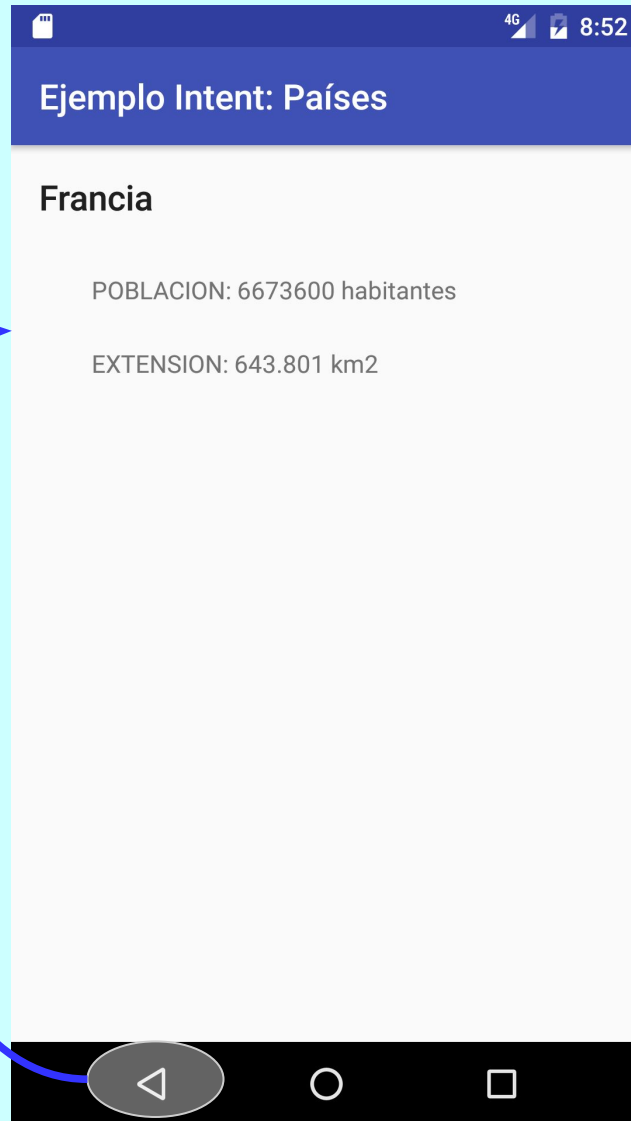
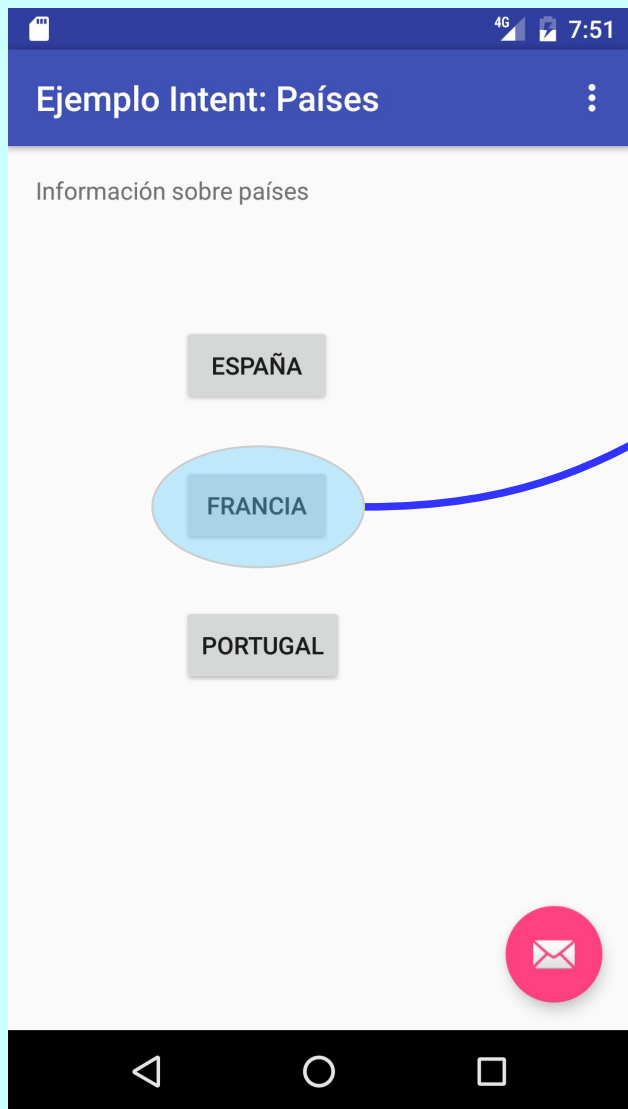
```
@Override
```

```
protected void onNewIntent(Intent intent) {  
    handleIntent(intent);  
}
```




























EJEMPLO

# Intent\_Paises

MTV: Ejemplo de uso de *intent*





- ▼  **app**
  - ▼  manifests
    -  **AndroidManifest.xml**
  - ▼  java
    - ▼  com.example.vrivas.intent\_paises
      -   espana
      -   infoPais
      -   MainActivity
    - ▶  com.example.vrivas.intent\_paises (androidTest)
    - ▶  com.example.vrivas.intent\_paises (test)
  - ▼  res
    -  drawable
    - ▼  layout
      -  activity\_espana.xml
      -  activity\_info\_pais.xml
      -  activity\_main.xml
      -  content\_main.xml
    - ▶  menu
    - ▶  mipmap
    - ▼  values
      -  colors.xml
      - ▶  dims.xml (2)
      -  strings.xml
      - ▶  styles.xml (2)

# AndroidManifest.xml

- Permisos que necesita la app
- Versión de API mínima
- HW que usará: cámara, GPS, pantalla multitáctil
- Librerías adicionales requeridas: Google Maps...
- Componentes y capacidades de la aplicación

# AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.vrivas.intent_paises">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportRtl="true"
        android:theme="@style/AppTheme">

        <activity
            android:name="com.example.vrivas.intent_paises.MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

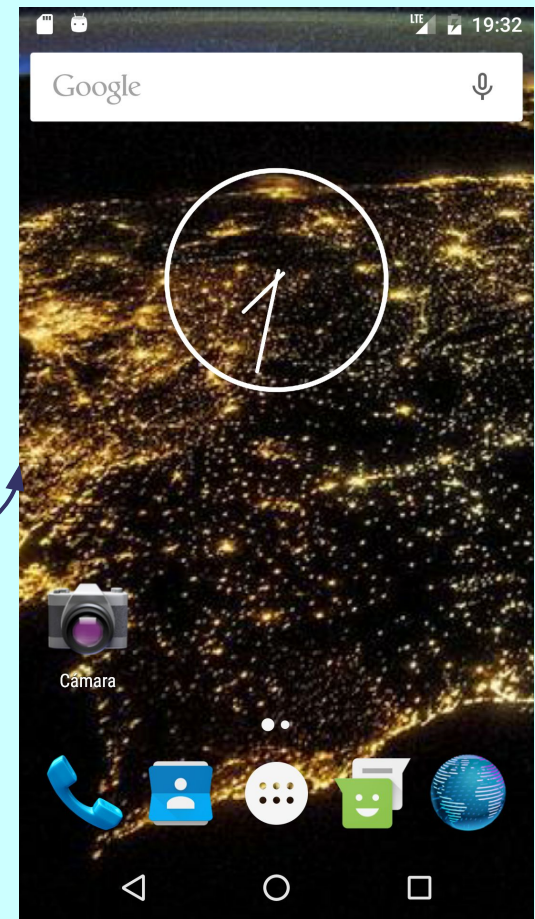
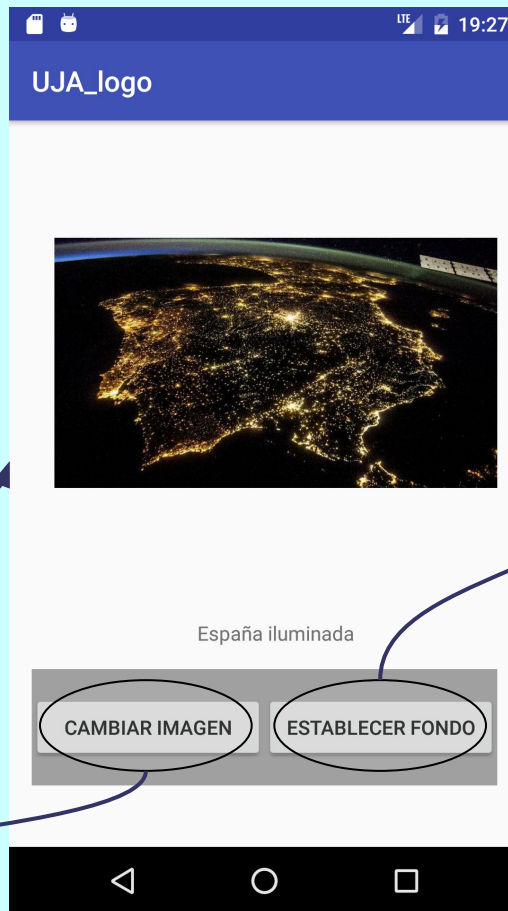
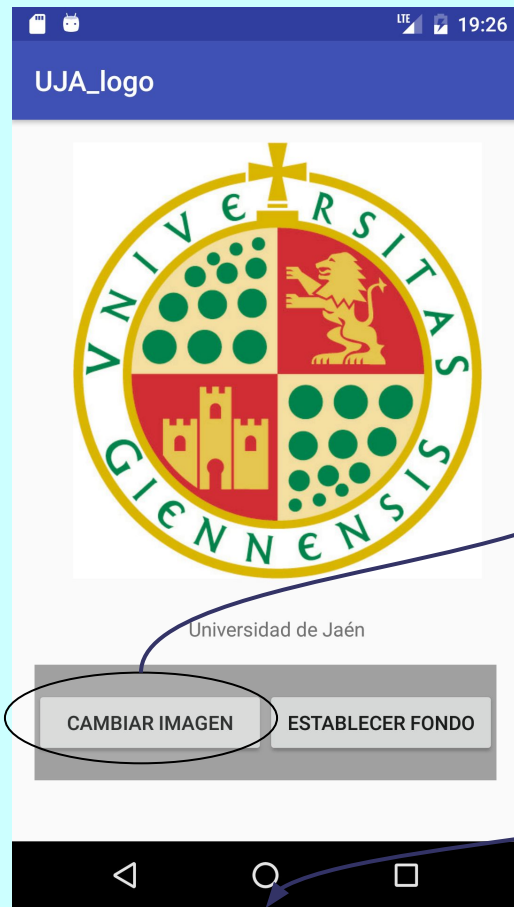
        <activity android:name="com.example.vrivas.intent_paises.espana" />

        <activity android:name="com.example.vrivas.intent_paises.infoPais" />

    </application>
</manifest>
```

EJEMPLO

# Ejemplo guiado



# 1. Crear proyecto

- Crea un nuevo proyecto: UJA\_logo
  - API 15
  - “Empty activity” para la principal
  - “Hello world”: Bombilla > Extraer recurso
    - Manualmente: @string/hello\_world
    - String.xml: añade dicho string
  - Crear recurso en español: “open editor”

## 2. Activity layout, imagen y descripción

- Cambiar el layout a LinearLayout
- Añadir una ImageView
  - Asignarle el logo de la UJA
    - NombreProyecto/app/src/main/res/drawable
  - Cambiarle el id a “imagen”
  - Establecer tamaño
- Editar el TextView
  - Cambiarle el id a “imagen\_titulo”
  - Cambiar gravity a center

## 3. Botón Cambiar Imagen

- Introducir el botón
- Cambiar su id a `bt_cambiar_imagen`
- Cambiar su text a `st_bt_cambiar_imagen`
  - Añadirlo a `string.xml` y traducirlo
- Añadir el evento `onClick`
- Programarlo en `MainActivity.java`

# EJEMPLO

```
public class MainActivity extends AppCompatActivity {  
    // Indica la imagen que se está visualizando  
    int numImagen=0;  
  
    // Se pulsa el botón cambiar imagen  
    public void cb_cambiar_imagen(View view) {  
        TextView t= (TextView) findViewById(R.id.imagen_titulo);  
        ImageView i= (ImageView) findViewById(R.id.imagen);  
        numImagen=(numImagen+1)%2;  
        switch (numImagen) {  
            case 0: {  
                t.setText(getResources().getString(R.string.uja));  
                i.setImageBitmap(  
                    BitmapFactory.decodeResource(  
                        getApplicationContext().getResources()  
                        , R.drawable.ujaen_default_icon  
                    )  
                );  
                break;  
            }  
            case 1: {  
                t.setText(getResources().getString(R.string.spain));  
                i.setImageBitmap(  
                    BitmapFactory.decodeResource(  
                        getApplicationContext().getResources()  
                        , R.drawable.spain  
                    )  
                );  
                break;  
            }  
        }  
    }  
}
```



## 4. Añadir botón Establece fondo

- ¿Se puede añadir al lado del otro?: No
  - Incluir un `LinearLayout` e insertarle los botones
  - Cambiar `gravity`, `background` y `layout_marginTop` si se desea
  - Modificar `layout_weight` a los botones (de 1 a 0 y viceversa) para comprobar qué ocurre
  - Codificar la función `cb_establecer_fondo`

# EJEMPLO

```
public void cb_cambiar_fondo( View view ) {  
    //obtener el gestor del fondo de pantalla:  
    WallpaperManager wallpaperManager =  
    WallpaperManager.getInstance(getApplicationContext());  
  
    Bitmap bitmap=null;  
  
    //obtener el bitmap de la imagen:  
    switch (numImagen) {  
        case 0: {  
            bitmap = BitmapFactory.decodeResource(  
                getApplicationContext().getResources()  
                , R.drawable.ujaden_default_icon);  
  
            break;  
        }  
        case 1: {  
            bitmap = BitmapFactory.decodeResource(  
                getApplicationContext().getResources()  
                , R.drawable.spain);  
  
            break;  
        }  
    }  
  
    // establecer el fondo de pantalla, capturando excepción  
    try {  
        wallpaperManager.setBitmap(bitmap);  
    } catch (Exception e) { e.printStackTrace();}  
}
```

## 5. Establecer permisos

- ¿Ha funcionado la aplicación?: NO
- 11-21 21:11:21.661 4232-4232/com.example.vrivas.uja\_logo W/System.err: java.lang.SecurityException: Access denied to process: 4232, must have permission android.permission.SET\_WALLPAPER
- Han de establecerse los permisos correspondientes en AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.vrivas.uja_logo">
```

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

```
<application...
```

## 6. Rotación de pantalla (a)

- Prueba a rotar la pantalla en el emulador:  
¿se ven bien la imagen y los botones?
- Podemos solucionarlo de dos formas:
  - 1. Impidiendo que el usuario rote la pantalla
  - 2. Aportando layouts para ambas orientaciones

## 6. Rotación de pantalla (b)

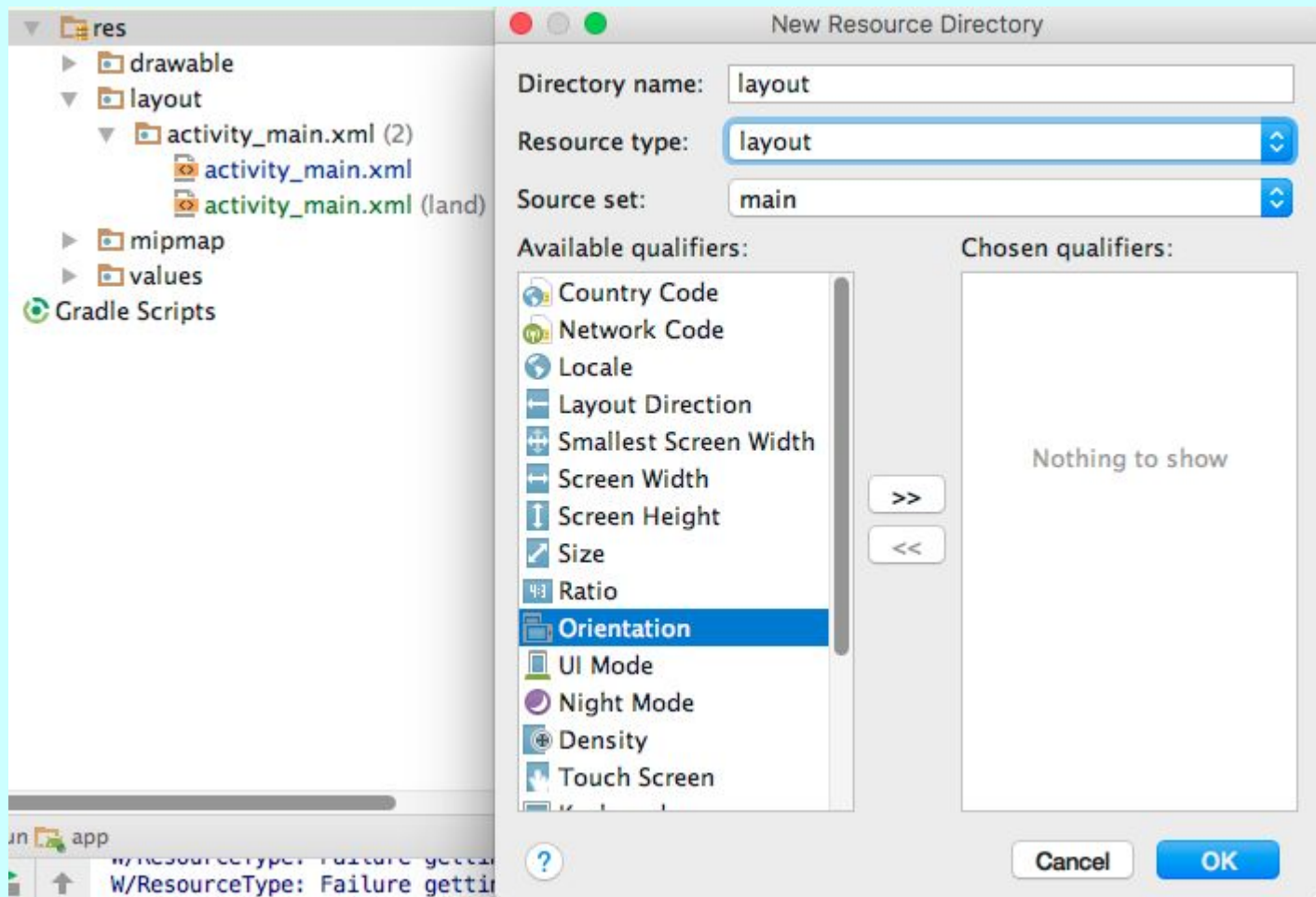
- Para impedir que el usuario rote la pantalla, modificamos el `AndroidManifest.xml`

```
<activity android:name=".MainActivity"  
    android:screenOrientation="portrait">
```

- Comprueba qué ocurre ahora al girar la pantalla en el emulador

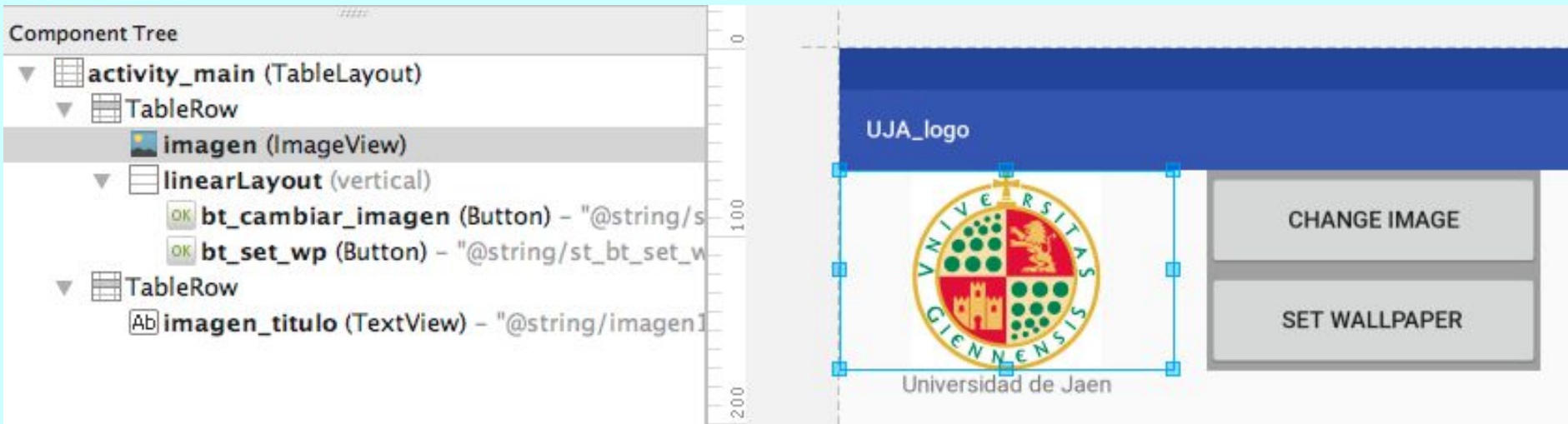
## 6. Rotación de pantalla (c)

- Para indicar un Layout distinto cuando la pantalla está rotada, hemos de proporcionar un res > layout-land



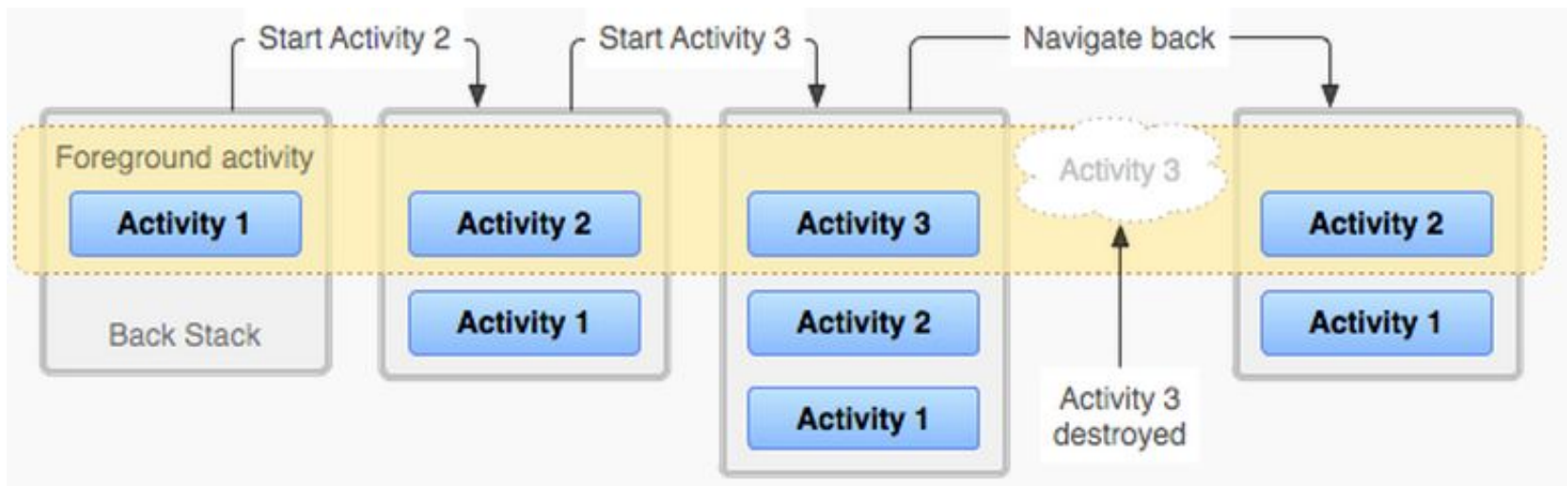
## 6. Rotación de pantalla (d)

- Finalmente, definiremos un nuevo layout para la orientación apaisada



# Ciclo de vida de una actividad

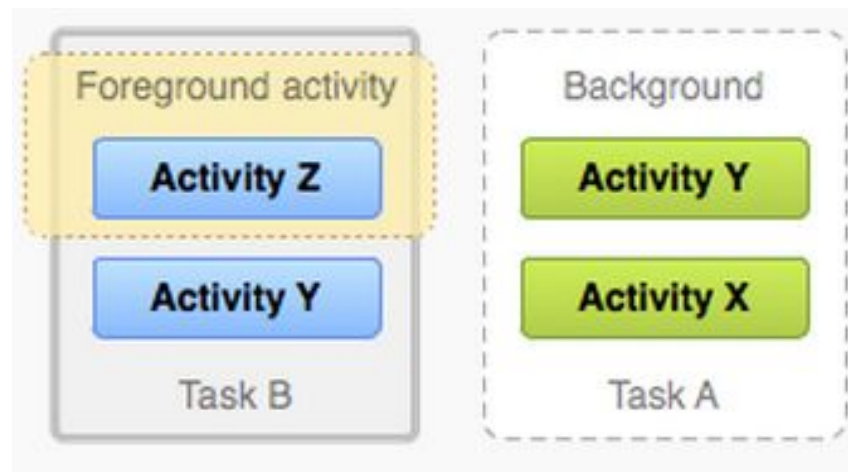
- Cada aplicación se ejecuta en un proceso.
- Si Android requiere liberar recursos matará aplicaciones, salvo la que esté en ejecución
- *Back Stack*:
  - pila de actividades iniciadas por una aplicación: la última en la cima
  - si una actividad abre otra, entonces la nueva se sitúa en la cima
  - al pulsar el botón “atrás” del dispositivo, la actividad en ejecución se destruye, se elimina del tope y se abre la que queda en la cima





# Pulsando el botón *Home*

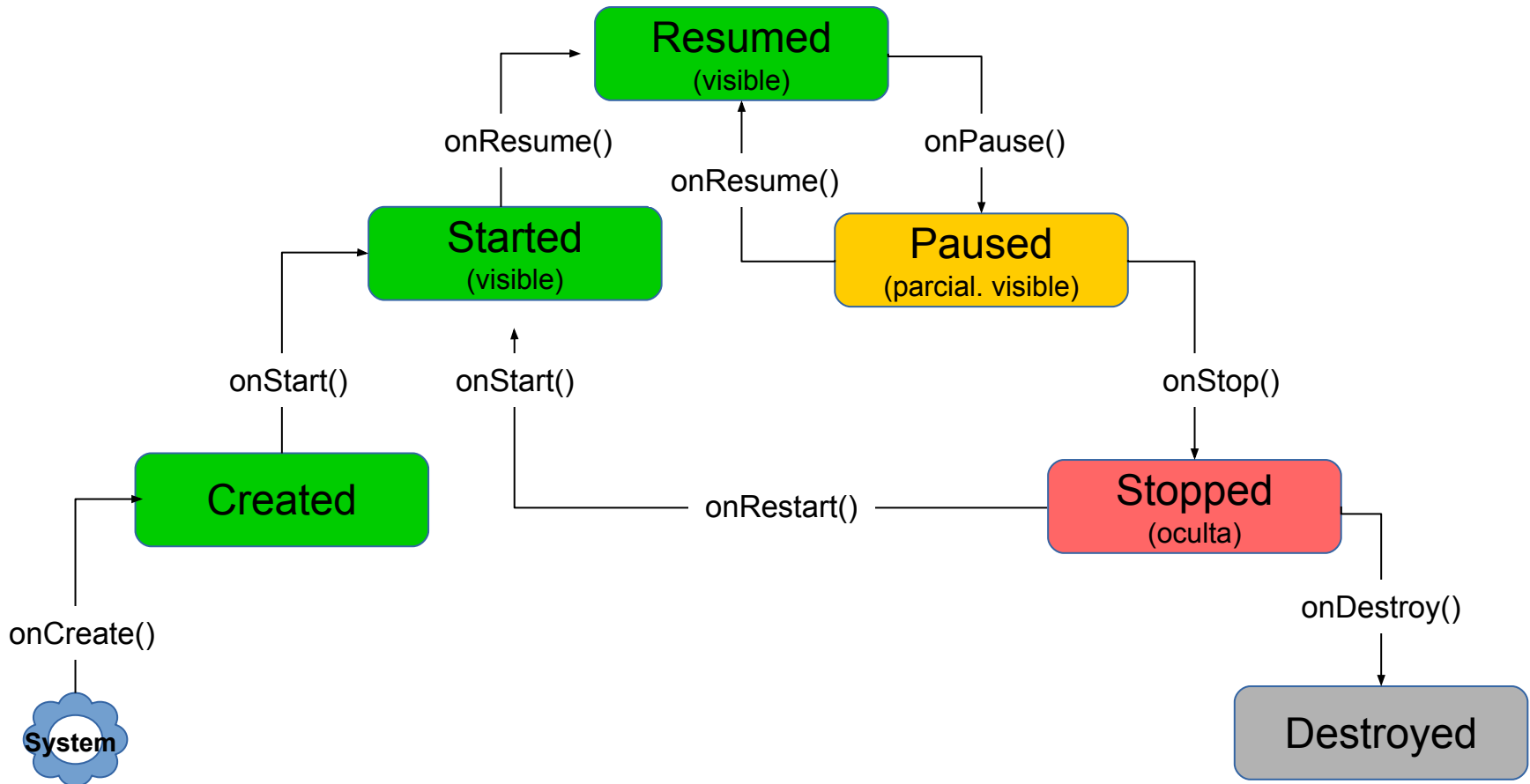
- Las actividades pasan a segundo plano, no se destruyen
- Cuando se vuelve a pulsar sobre el icono de la aplicación, se reanuda la actividad pasando a primer plano



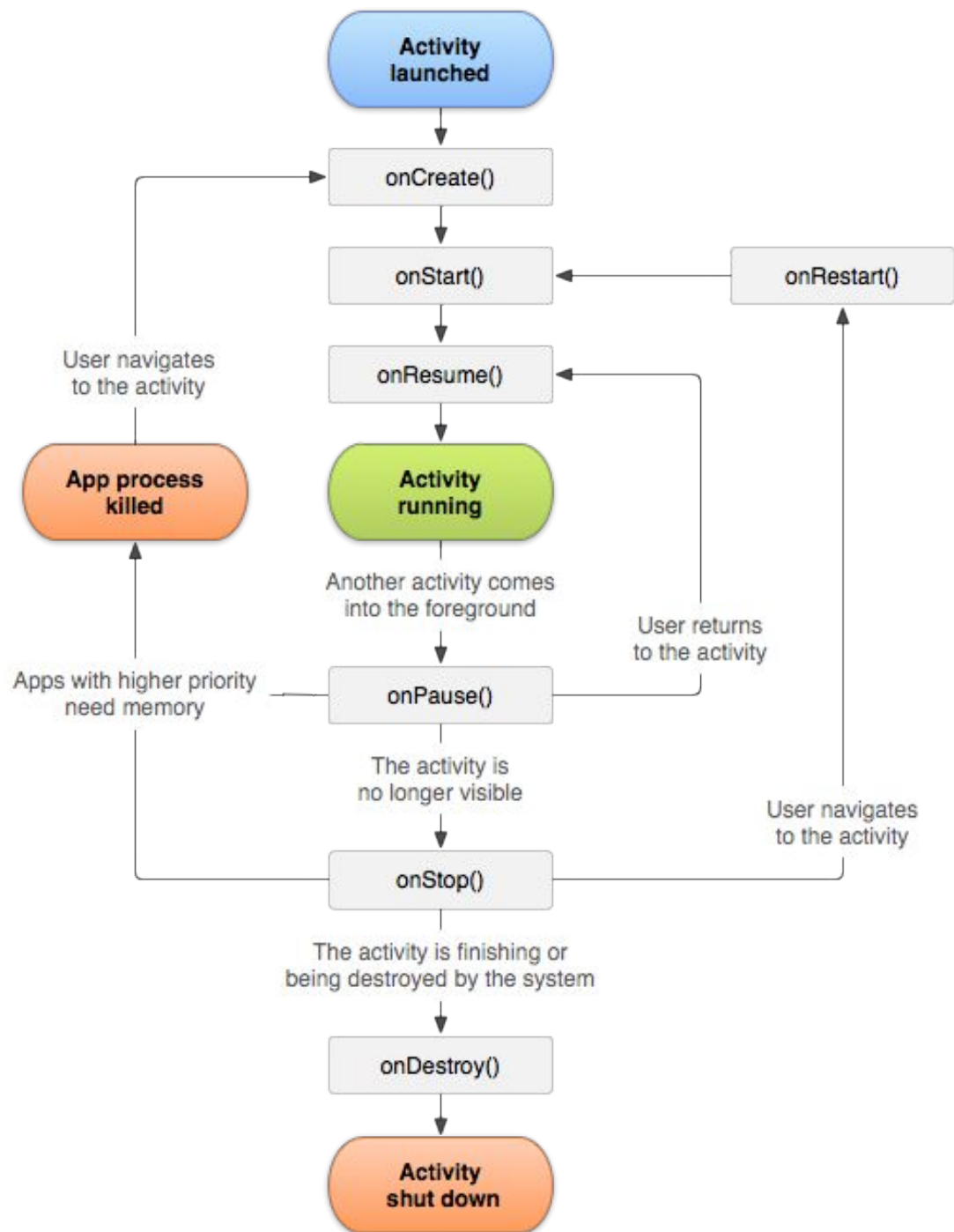
# Estados de una actividad

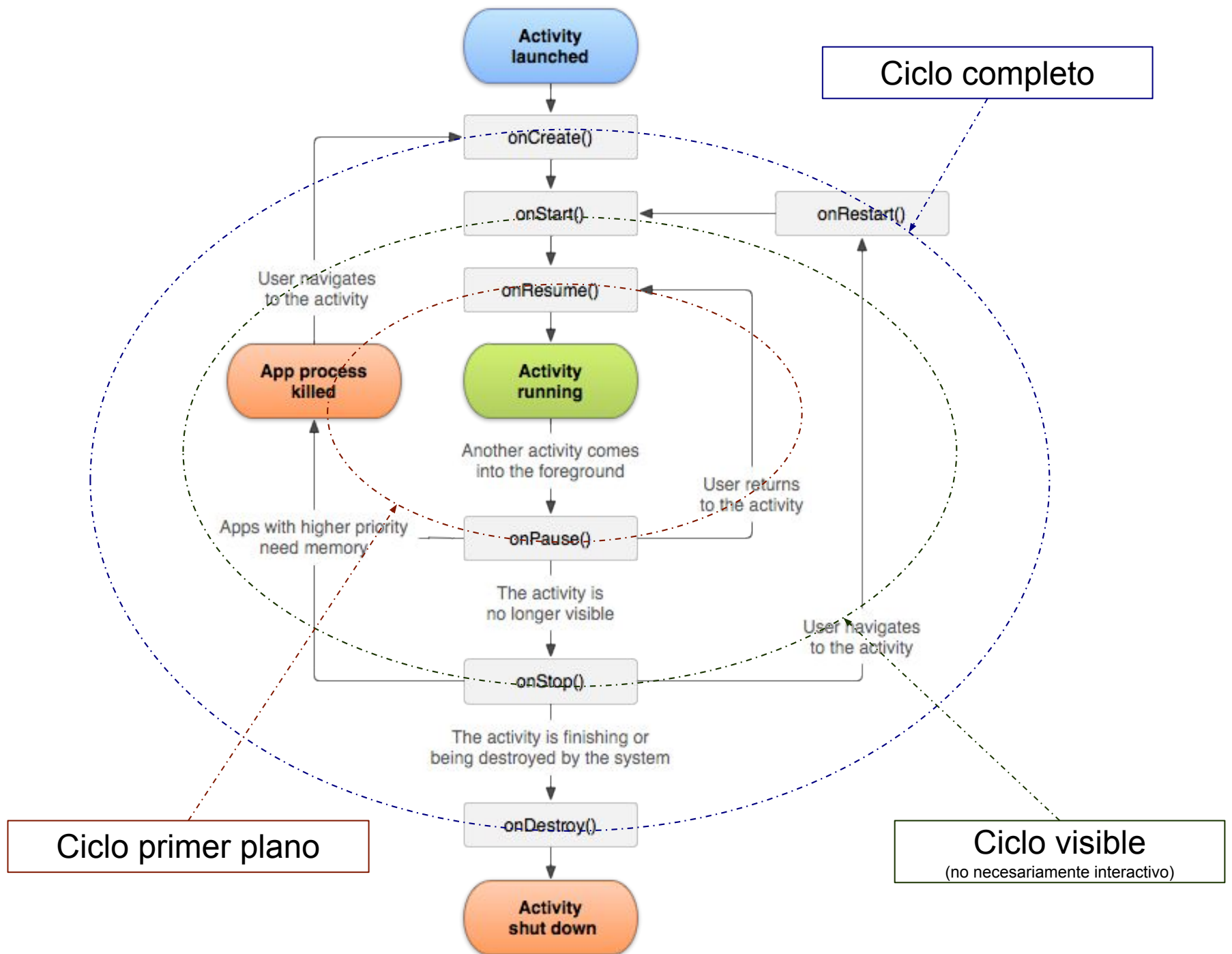
- En ejecución (*resumed*)
  - en primer plano recibiendo la interacción del usuario.
- Pausada (*paused*)
  - visible pero el usuario no puede interaccionar con ella (no recibe eventos de usuario).
- Parada (*stopped*)
  - no es visible pero sigue en ejecución. Los datos deben guardarse y detener todos los procesamientos.
- Muerta (*destroyed*)
  - el sistema la ha eliminado, no está en ejecución y desaparece del back stack. Los datos no almacenados se pierden.

# Transiciones entre estados



¡¡Callbacks pueden sobrecargarse!!

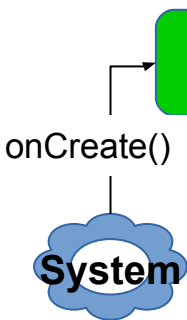




# Callbacks de una Actividad

```
public class Activity extends ApplicationContext {  
    protected void onCreate(Bundle savedInstanceState);  
  
    protected void onStart();  
  
    protected void onRestart();  
  
    protected void onResume();  
  
    protected void onPause();  
  
    protected void onStop();  
  
    protected void onDestroy();  
  
}
```

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



*onCreate*


*(Bundle savedInstanceState)*

- Se invoca al crear la actividad
- Debe cargar la interfaz e inicializar variables de clase
- Se ejecuta una sola vez para todo el ciclo de vida
- A continuación, se invocan automáticamente `onStart()` y `onResume()`

# Objeto *Bundle* *savedInstanceState*

- Conjunto de pares clave-valor utilizado para guardar el estado de una instancia de un actividad
  - Por defecto, almacena estado de las *views* del *layout* de la actividad
  - MUY IMPORTANTE: asignar *android:id* a las *views*
- Sirve para **recrear** una instancia destruida
  - Ejemplo: al cambiar la orientación de la pantalla se destruye y se recrea la actividad
- Podemos añadir y recuperar pares:

```
savedInstanceState.putInt( "NUM_IMG", numImagen )
```



Mejor definir  
una  
*static final string*

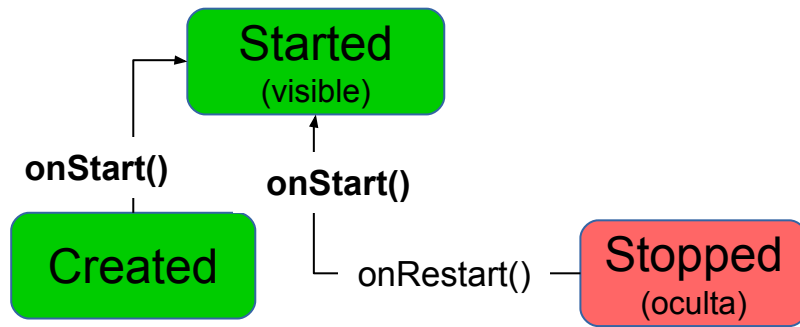


```
@Override
protected void onSaveInstanceState( Bundle savedInstanceState) {
    savedInstanceState.putInt("NUM_IMG", numImagen);
    super.onSaveInstanceState(savedInstanceState);
}
```

// Opción 1:

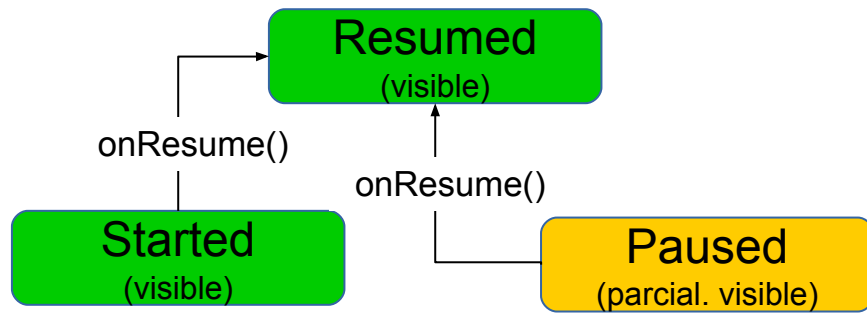
```
@Override
protected void onRestoreInstanceState( Bundle savedInstanceState) {
    numImagen=savedInstanceState.getInt("NUM_IMG");
    super.onRestoreInstanceState(savedInstanceState);
    mostrar_imagen();
}
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    /*Opción 2: Después del setContentView
        if( savedInstanceState!=null) {
            numImagen=savedInstanceState.getInt("NUM_IMG");
            mostrar_imagen();
        }*/
}
```



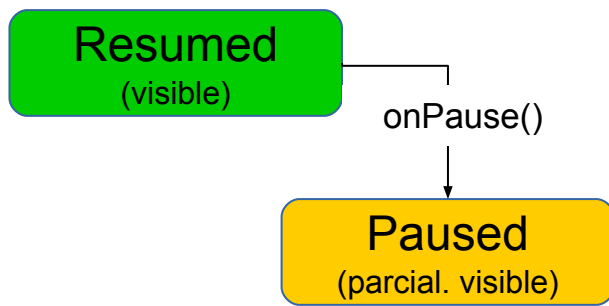
## *onStart*

- Se llama tras *onCreate*, o tras *onRestart* si la actividad había sido detenida y vuelve a primer plano.
  - Debe continuar con la llamada a *onResume*.
- Las clases derivadas deben llamar al método de la superclase



# *onResume*

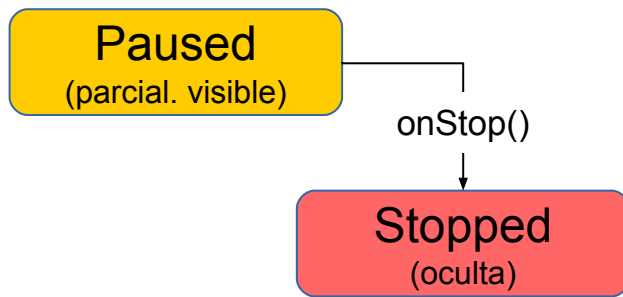
- Inicialización de variables y procesos necesarios para la ejecución de la actividad
  - Dichos procesos deberán detenerse en *onPause()* y los recursos liberados
  - Volver a inicializar los componentes que fueron liberados en *onPause()*



# *onPause*

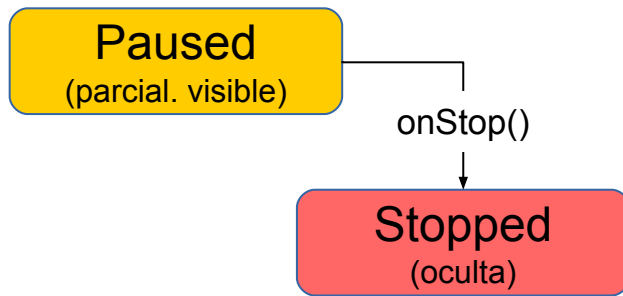
- Se ejecuta cuando la actividad deja de estar en primer plano porque otra actividad lo ocupa
  - La actividad sigue en memoria
- Debe:
  - parar acciones que no deben seguir si el foco no lo tiene la actividad (vídeos, animaciones, ...)
  - liberar recursos que consumen batería (cámara, GPS, ...)
- Si el usuario vuelve a la actividad se ejecutará *onResume()*

```
public void onPause() {  
    super.onPause(); // Call superclass method first  
    // Release the camera  
    if( mCamera!=null) {  
        mCamera.release();  
        mCamera=null;  
    }  
}
```



# *onStop*

- Se ejecuta cuando la actividad deja de verse:
  - el usuario cambia a otra aplicación (desde aplicaciones recientes)
  - se crea una nueva actividad desde la aplicación
    - la primera se reiniciará si se pulsa el botón atrás
  - se recibe una llamada de teléfono
- La actividad se mantiene en memoria
- La actividad se recupera automáticamente
  - No hay que re-arrancar lo iniciado en onCreate(), onStart() y onResume()
  - Sistema guarda los valores actuales de *views* de la actividad



## *onStop* (y II)

- Si el sistema necesita liberar memoria, ejecutará *onDestroy()*
  - en casos extremos el sistema puede matar la activity sin llamar a *onDestroy()*, así que *onStop()* debe liberar recursos

Started  
(visible)

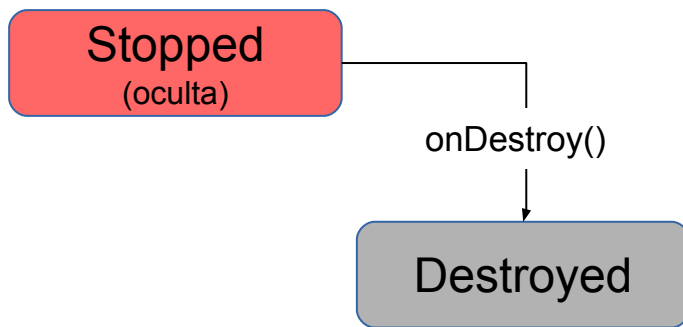
onStart()

onRestart()

Stopped  
(oculta)

*onRestart*

- Se ejecuta si una actividad parada vuelve al primer plano
- Se debe hacer procesamiento necesario porque la actividad se había parado, pero no destruido.
  - Volver a asignar recursos liberados en onStop()



# *onDestroy*

- Último *callback* invocado antes de que la actividad sea eliminada de la memoria del sistema
- Previamente ya deben estar liberados todos los recursos
  - antes se ejecuta *onPause()* y *onStop()*
- Solo hay que implementarlo si en *onCreate()* se lanzaron hebras en segundo plano que podrían quedarse consumiendo recursos



# Consideraciones finales

- No es obligatorio implementar todos los *callback* pero sí considerar situaciones como:
  - El usuario cambia de aplicación: permitir la vuelta en el mismo estado en que estaba
  - El usuario cambia la orientación del dispositivo: la aplicación debe seguir funcionando sin perder datos
  - No consumir recursos cuando no sea necesario y recuperarlos cuando hagan falta