U.T.2 PROGRAMACIÓN PARA DISPOSITIVOS MÓVILES. ACTIVIDADES.

COMPONENTES PRINCIPALES DE UNA APLICACIÓN ANDROID.

ACTIVITIES. Una actividad (Activity) viene a ser una ventana que tiene implícito un proceso, puede estar activo o en segundo plano.

VIEW. Las vistas (*view*) son los componentes básicos con los que se construye la interfaz gráfica de la aplicación..

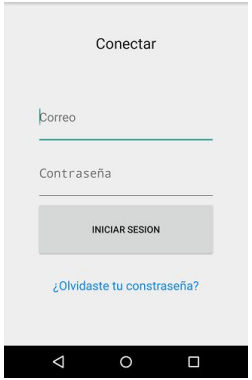
CONTENT PROVIDERS. Proveedores de datos. Ponen un grupo de datos a disposición de distintas aplicaciones, extienden de la clase ContentProvider.Con esta clase se permite acceder al sistema de ficheros, bases de datos SQLite o cualquier otra fuente de datos unificada.

SERVICES. Es un programa que no tiene interfaz de usuario y se ejecuta en segundo plano sin bloquear la aplicación ya que se ejecutan en un hilo distinto.

INTENTS. Un intent se produce cuando una actividad llama a otra, es análogo a un servicio de mensajería. Las Actividades, Servicios y BroadcastReceiver se activan a través de mensajes asíncronos llamados intent

Actividades (*Activities*)

★ Una actividad (Activity) viene a ser una **pantalla con la que los usuarios pueden interactuar para realizar una acción.**



★ Son las encargadas de mostrar la interfaz de usuario e interactuar con ella. Responden a los eventos generados por el usuario (pulsar botones, buscar en la agenda,hacer una foto, escribir un mensaje etc).

★ Por cada pantalla distinta hay una actividad, normalmente las aplicaciones tienen múltiples actividades y una actividad fijada como punto de entrada, que es la **actividad principal** **MainActivity**.

Actividades (*Activities*)

- ★ Heredan de la clase **Activity** o de una de sus subclases.
- ★ Cada actividad puede a su vez **iniciar otra actividad** para poder realizar diferentes acciones, deteniendo la actividad anterior.
- ★ Las actividades se almacenan en una "pila de actividades" donde "*el último en entrar es el primero en salir*", por lo que, cuando el usuario termina de interactuar con la actividad actual y presiona el botón *Atrás*, se quita de la pila (y se destruye) y se reanuda la actividad anterior.
- ★ Cuando se detiene una actividad porque se inicia otra, se notifica el cambio de estado a través de los métodos callback del **ciclo de vida de la actividad**.

Ciclo de vida de una actividad

Una actividad en Android puede estar en uno de estos cuatro estados:

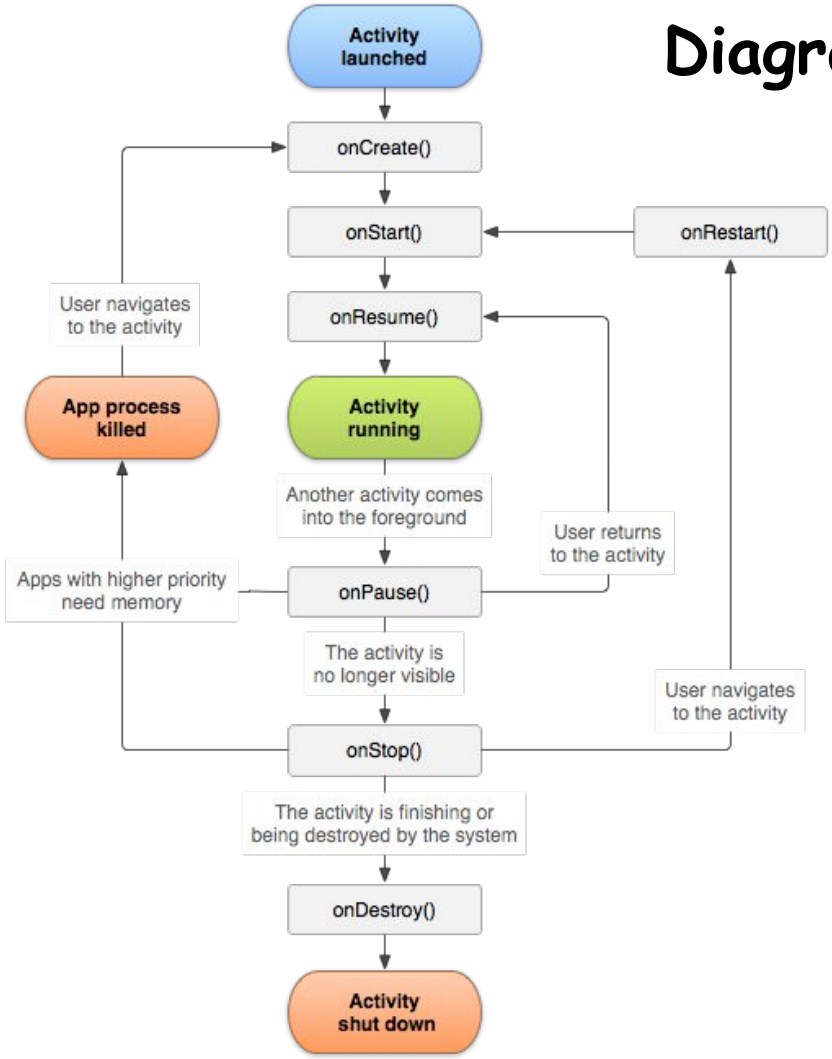
- ★ **Activa** (Running): La actividad está encima de la pila, es decir, es visible y tiene el foco.
- ★ **Visible** (Paused): La actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a activa otra actividad con alguna parte transparente o que no ocupa toda la pantalla. Cuando una actividad está tapada por completo, pasa a estar parada.
- ★ **Parada** (Stopped): Cuando la actividad no es visible, se recomienda guardar el estado de la interfaz de usuario, preferencias, etc.
- ★ **Destruida** (Destroyed): Cuando la actividad termina al invocarse el método finish(), o es matada por el sistema Android, sale de la pila de actividades.

Métodos para capturar los estados de una Actividad

Cada vez que una actividad cambia de estado se van a producir eventos que podrán ser capturados por ciertos métodos de la actividad.

- ★ **onCreate(Bundle)**: Se llama en la creación de la actividad. Se utiliza para realizar inicializaciones, como la creación de la interfaz de usuario o la inicialización de estructuras de datos. Puede recibir información de estado de instancia (en una instancia de la clase Bundle), por si se reanuda desde una actividad que ha sido destruida y vuelta a crear.
- ★ **onStart()**: Nos indica que la actividad está a punto de ser mostrada al usuario.
- ★ **onResume()**: Se llama cuando la actividad va a comenzar a interactuar con el usuario. Es un buen lugar para lanzar las animaciones y la música.
- ★ **onPause()**: Indica que la actividad está a punto de ser lanzada a segundo plano, normalmente porque otra aplicación es lanzada. Es el lugar adecuado para detener animaciones, música o almacenar los datos que estaban en edición.
- ★ **onStop()**: La actividad ya no va a ser visible para el usuario. Si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.
- ★ **onRestart()**: Indica que la actividad va a volver a ser representada después de haber pasado por onStop().
- ★ **onDestroy()**: Se llama antes de que la actividad sea totalmente destruida. Por ejemplo, cuando el usuario pulsa el botón <volver> o cuando se llama al método finish(). Si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

Diagrama de estados de una Actividad



El ciclo de vida de la actividad se extiende desde la creación de la actividad hasta su destrucción, cuando el sistema recupera los recursos de esa actividad. A medida que un usuario navega dentro de una actividad y fuera de ella, cada actividad pasa de un estado a otro en su ciclo de vida.

1. Cuando compilas y ejecutas la app se llama a onCreate(), onStart() y onResume() la actividad se inicia por primera vez.
2. Si presiona el botón **Back** en el dispositivo, se llama a onPause() y onStop() en ese orden. El SO Android podría cerrar tu actividad onDestroy() si el código llama al método finish() de la actividad o si el usuario fuerza el cierre de la app.
3. Presiona el botón **Home** en el dispositivo. Cuando regresas a la pantalla principal, tu app se ejecuta en segundo plano en lugar de cerrarse por completo. Observa que se llama a los métodos onPause() y onStop().

Actividad principal de la aplicación.

Está representada por la clase **MainActivity.java** y la interfaz de usuario **activity_main.xml.**

En Android no existe método main, la clase invoca a una serie de métodos relativos al ciclo de vida de la actividad: onCreate(), onPause(), onDestroy(),...

El método **onCreate()** es llamado cuando se crea la actividad, hace referencia al método onCreate de la clase padre. Es donde debemos iniciar los componentes fundamentales de la actividad, el método setContentView() llama a un recurso de layout que define la **interfaz gráfica de usuario** de la actividad.

```
package com.example.appsegundabutton;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        /*ViewCompat.setOnApplyWindowInsetsListener(findViewB
            Insets systemBars = insets.getInsets(WindowInsets
            v.setPadding(systemBars.left, systemBars.top, sys
            return insets;
        });*/
    }
}
```

Declarar la actividad en el archivo Android Manifest

La actividad debe estar declarada en el archivo de manifiesto para que pueda estar accesible para el sistema. Viene indicada por un elemento <activity> como campo secundario del elemento <application>.

El elemento <action> especifica que este es el punto de entrada "principal" a la aplicación. El elemento <category> especifica que esta actividad debe aparecer en el lanzador de aplicaciones del sistema (para que los usuarios puedan iniciar esta actividad).

```
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="HolaMundoAndroid"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivityHolaMundoAndroid"
            android:label="HolaMundoAndroid" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Existen otros atributos que puedes incluir para definir propiedades.

El atributo android:name es el único obligatorio; especifica el **nombre de clase de actividad**.

Una vez que publicas tu aplicación, no debes cambiar este nombre porque podrías anular funcionalidades, como accesos directos de la aplicación

Fichero XML del Diseño de la actividad principal (activity_main.xml)

Representa la interfaz de usuario de la actividad principal

Cada archivo de diseño debe contener exactamente un elemento raíz, que debe ser un objeto View o ViewGroup (p.Ej. ConstraintLayout) al que puedes agregar widgets u objetos de diseño adicionales (TextView, Button,...) para crear gradualmente una jerarquía de vistas que defina el diseño.

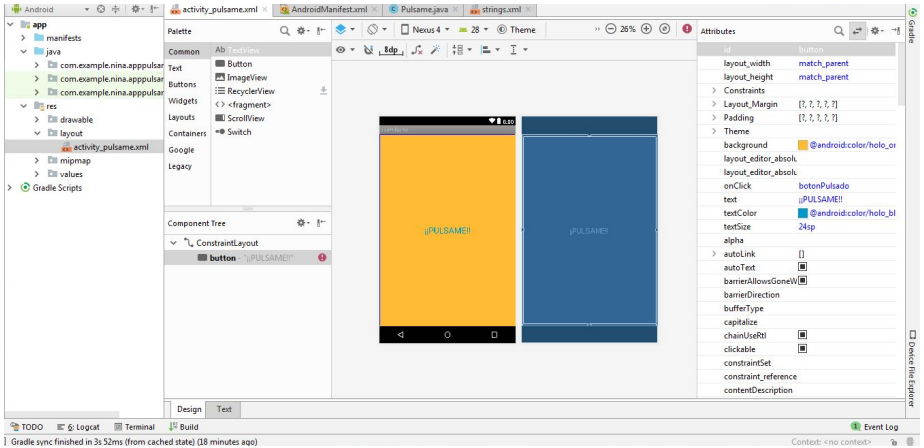
La ventaja de declarar la IU en XML es que permite separar mejor la presentación de la aplicación del código que controla su comportamiento.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Pulsame">

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@android:color/holo_orange_light"
        android:onClick="botonPulsado"
        android:text="¡¡PULSAME!!"
        android:textColor="@android:color/holo_blue_dark"
        android:textSize="24sp"
        tools:layout_editor_absoluteX="160dp"
        tools:layout_editor_absoluteY="227dp" />
</android.support.constraint.ConstraintLayout>
```

Vista de Diseño de la actividad.

- ★ **Android Studio Layout Editor** permite crear diseños fácilmente arrastrando y soltando vistas desde la ventana **Palette**, a la izquierda.
- ★ En la ventana **Component Tree** de la parte inferior izquierda se muestra la jerarquía de vistas del diseño.
- ★ En la ventana **Attributes** de la parte derecha se muestran las propiedades del objeto que pueden ser modificadas.



Todos los objetos View y ViewGroup admiten su propia variedad de atributos XML. Algunos atributos son específicos de un objeto View y otros son comunes para todos los objetos View ya que se heredan desde la clase View raíz (como el atributo id).

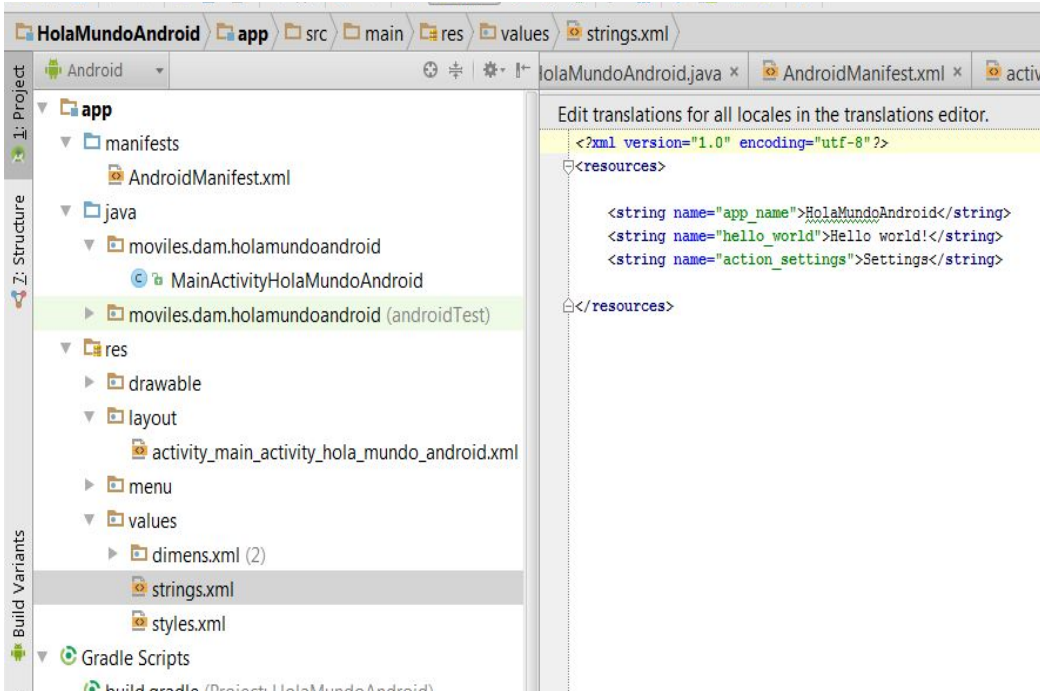
Pincha en el siguiente enlace para ver cómo crear una IU sencilla.

https://developer.android.com/training/basics/firstapp/building-ui?hl=es-419

El archivo **strings.xml**

Las cadenas de texto son un tipo especial de recursos, que el sistema puede gestionar automáticamente por nosotros. Se encuentran definidas en el fichero de localización de cadenas en **res/values/strings.xml**

Nos permiten reutilizar las cadenas en distintas partes de nuestra aplicación, así como, crear juegos de las mismas cadenas en diferentes idiomas a este proceso se le denomina **internacionalización.**



LogCat: el registro en Android

Como ayuda a la depuración, en Android tenemos el sistema de log llamado LogCat. Las aplicaciones envían mensajes al sistema de log, y, durante la ejecución, Eclipse obtiene todos los mensajes y nos los muestra en la ventana LogCat.

Los mensajes de registro tienen la siguiente información:

- Fecha/Hora del mensaje.
- Criticidad. Nivel de gravedad del mensaje (se detalla más adelante).
- PID. Código interno del proceso que ha introducido el mensaje.
- Tag. Etiqueta identificativa del mensaje (se detalla más adelante).
- Mensaje. El texto completo del mensaje.

```
09-08 16:52:37.081 2765-2801/com.example.nina.apppulsame I/OpenGLRenderer: Initialized EGL, version 1.4

09-08 16:53:07.383 2765-2765/com.example.nina.apppulsame D/Pulsame: botonPulsado 1
```

Envío de mensajes al Log

Tag: etiqueta que identifica al mensaje. Normalmente se utiliza el nombre del programa o de la actividad.

Nivel: es el nivel de prioridad (gravedad) del mensaje:

- Verbose: mensajes de poca importancia, con detalles extremadamente finos de la ejecución. Nunca deberían generarse mensajes con esta prioridad en una aplicación en producción.
- Debug: mensajes de depuración.
- Information: mensajes informativos sobre la ejecución.
- Warning: avisos de sucesos importantes que podrían terminar desembocando en problemas.
- Error: notificaciones de errores que afectan a la aplicación.

Para enviar mensajes al log, utilizamos la clase android.util.Log, y sus métodos v(), d(), i(), w() y e(), para cada uno de las cinco prioridades de errores. Todos son estáticos, y reciben dos parámetros. El primero es el tag, que identifica a la aplicación; el segundo es el mensaje que queremos registrar.

```
Ej. android.util.Log.d(TAG, "botonPulsado " + numveces);
```

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

appPulsame › app › src › main › java › com › example › nina › apppulsame › Pulsame

app

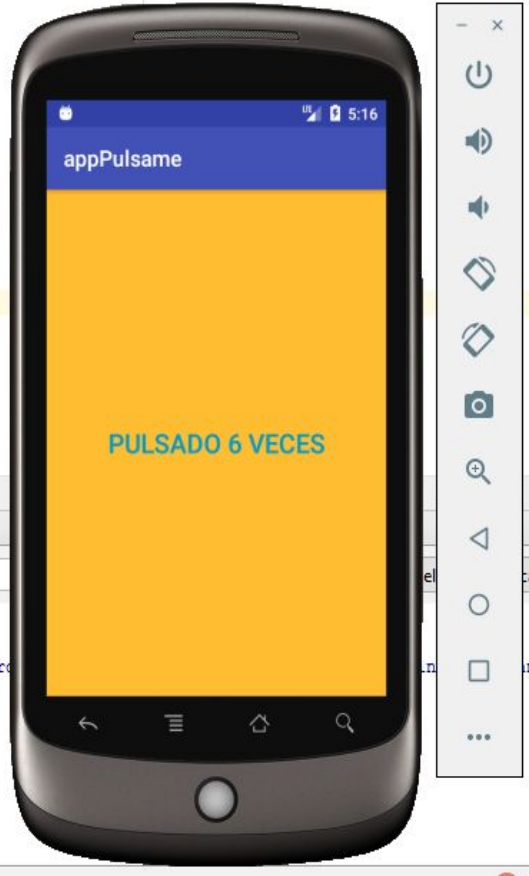
Android

activity_pulsame.xml | AndroidManifest.xml | Pulsame.java | strings.xml

```
app
  manifests
  java
    com.example.nina.apppulsar
      Pulsame
    com.example.nina.apppulsar
    com.example.nina.apppulsar
  res
Gradle Scripts
```

```
7
8   public class Pulsame extends AppCompatActivity {
9
10      private static final String TAG = "Pulsame";
11      private int numveces=0;
12
13      @Override
14      protected void onCreate(Bundle savedInstanceState) {
15          super.onCreate(savedInstanceState);
16          setContentView(R.layout.activity_pulsame);
17      }
18
19      public void botonPulsado(View v){
20       numveces++;
21       Button boton=(Button)v;
22       boton.setText("pulsado "+numveces+" veces");
23       android.util.Log.d(TAG,  msg: "botonPulsado " + numveces);
24
25      }
```

Pulsame › botonPulsado()



Logcat

Emulator Nexus_One_API_23 | com.example.nina.apppulsame | Verbose

```
    gralloc.default.so not found in /vendor. Trying /system/lib/hw/gralloc.default.so...
09-08 16:52:37.081 2765-2801/com.example.nina.apppulsame I/OpenGLRenderer: Initialized EGL, version 1.4
09-08 16:52:37.141 2765-2765/com.example.nina.apppulsame W/art: Before Android 4.1, method int android.support.v7.widget.Dr
09-08 16:53:07.383 2765-2765/com.example.nina.apppulsame D/Pulsame: botonPulsado 1
09-08 16:53:09.338 2765-2765/com.example.nina.apppulsame D/Pulsame: botonPulsado 2
09-08 16:53:10.378 2765-2765/com.example.nina.apppulsame D/Pulsame: botonPulsado 3
09-08 16:53:11.410 2765-2765/com.example.nina.apppulsame D/Pulsame: botonPulsado 4
09-08 16:53:12.427 2765-2765/com.example.nina.apppulsame D/Pulsame: botonPulsado 5
09-08 16:53:29.182 2765-2765/com.example.nina.apppulsame D/Pulsame: botonPulsado 6
```

4: Run | TODO | 6: Logcat | Android Profiler | Terminal | Build | Event Lo

Gradle build finished in 14s 226ms (23 minutes ago)

22:14 CRLF UTF-8 Context: <no context>