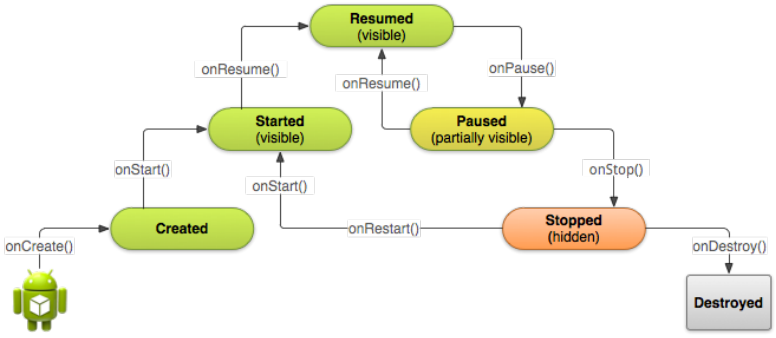
PRÁCTICA 3:

CICLO DE VIDA Y MENSAJES DE LOG



Empar Carrasquer Moya

2 DAM

Objetivos

- Aprender a gestionar el **ciclo de vida** de las aplicaciones Android y sus *activities*.
- Utilizar los mensajes de *log* registrar en **LogCat** los diferentes cambios de estado de las aplicaciones en Android.
- Crear nuevas *Activities* y añadirlas al proyecto.
- Conocer cómo podemos realizar **acciones** al pulsar un botón.
- Abrir una **nueva** *Activity* en respuesta a la pulsación de un botón.

Material a utilizar

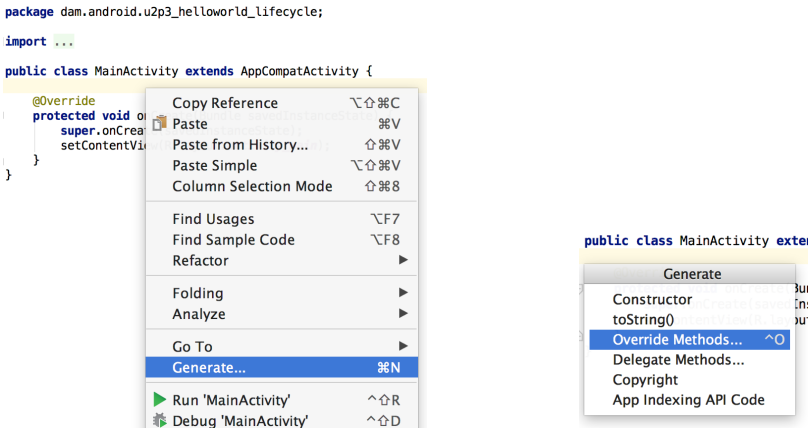
En esta práctica vamos a utilizar el siguiente software:

- Entorno de programación en **Android Studio IDE**, instalado y configurado en prácticas anteriores.
- **Crearemos** un **nuevo proyecto en Android Studio** (similar a aplicación *HelloWorld* realizada en la práctica anterior) que **muestra** en el *TextView* un mensaje de bienvenida (registrado en strings.xml).

1. Ciclo de vida de la aplicación y mensajes de *log*

Vamos a comprobar las diferentes fases del **ciclo de vida** de la aplicación, y cómo se **transita** de un **estado** a otro tal y como se ha visto en la teoría, procederemos a sobrescribir en la *activity* los **métodos** relacionados con el **ciclo de vida** y así observar los cambios **entre** los diferentes **estados**.

Para ello, y con el código fuente de la *MainActivity* del proyecto *HelloWorld* abierto en el entorno de desarrollo, seleccionaremos la opción ***Generate → Override Methods***.



En la siguiente imagen (*Figura 1*) podremos seleccionar aquellos métodos que deseamos implementar o sobrescribir, con lo que se autocompletará el código fuente con la implementación básica de estos métodos. En nuestro cas, los **métodos a sobrescribir son: *onPause()*, *onRestart()*, *onResume()*, *onStart()*, y *onStop()***.

(los ***ordenamos alfabéticamente*** y los ***seleccionamos*** uno a uno con la tecla **Ctrl**)

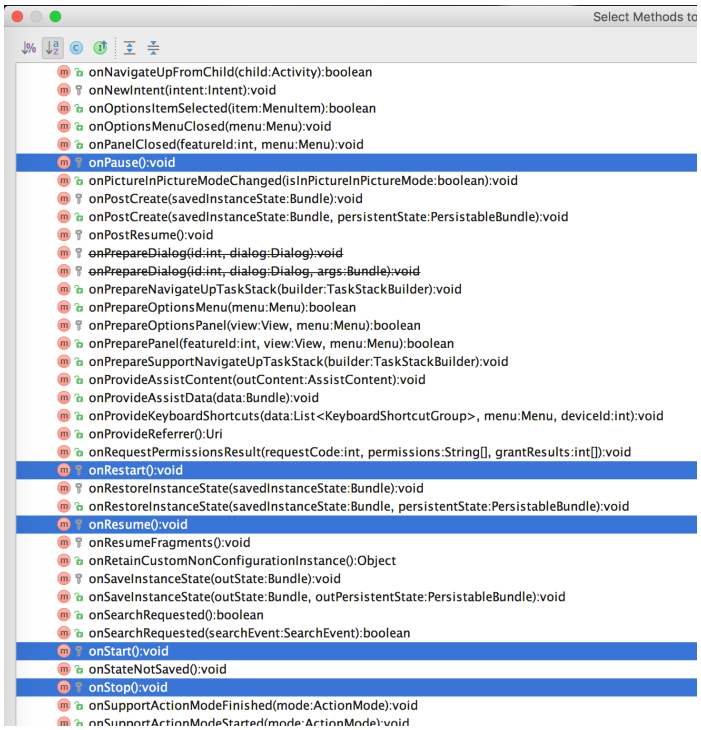


Figura 1. Ventana de selección de los métodos a sobreescribir

El resultado sería:

```
public class MainActivity extends AppCompatActivity {

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

    @Override
    protected void onPause() {
        super.onPause();
    }

    @Override
    protected void onResume() {
        super.onResume();
    }

    @Override
    protected void onStart() {
        super.onStart();
    }

    @Override
    protected void onStop() {
        super.onStop();
    }

    @Override
    protected void onRestart() {
        super.onRestart();
    }

}
```

Modificaremos cada uno de estos métodos para incluir un mensaje de ***Log*** y así, por medio de la ventana ***Logcat*** (donde se muestran los mensajes del sistema) poder seguir el flujo de las transiciones en el estado de la activity que se vayan produciendo durante su ejecución.

La clase ***Log***, de la biblioteca ***android.util***, dispone de métodos estáticos para la generación de mensajes de error, advertencia, información, etc.
Más detalles de la clase Log en:
https://developer.android.com/reference/android/util/Log.html

Para **registrar** desde el código un mensaje de log, se requiere de una cadena de texto(***tag***) que permite ***identificar el origen del mensaje***, y el **texto del mensaje** en sí que se desea registrar.
Además, desde la ventana **Logcat** podremos **filtrar** los que queremos visualizar.

Más detalles en el siguiente enlace:
https://developer.android.com/studio/debug/am-logcat?hl=es-419

`Log.e(String, String)` (error)
`Log.w(String, String)` (advertencia)
`Log.i(String, String)` (información)
`Log.d(String, String)` (depuración)
`Log.v(String, String)` (registro detallado)

Es una **convención** el declarar en una clase las diferentes constantes para los TAG que vayamos a necesitar en el desarrollo de la aplicación, tal y como muestra el fragmento de Código 1:

```
private static final String DEBUG_TAG = "LogsAndroid-1";
```

Código 1. Creación de constante para su uso como identificadores de mensajes de Log.

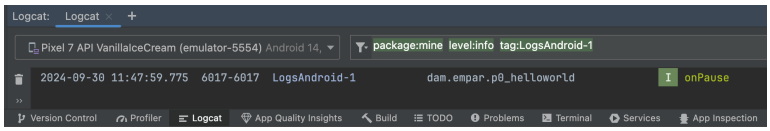
Un **ejemplo** de mensaje de Log para el método ***onPause()***, del tipo ***info*** (Log.**i**) sería:

```
@Override
protected void onPause() {
    super.onPause();
    Log.i(DEBUG_TAG, "onPause");
}
```

Atención*: Recuerda añadir el import para la clase Log*: `import android.util.Log;`

Si ejecutamos la aplicación, y sin salir de la misma, mostramos el escritorio, se pondrá la activity actual en **estado Pausado** y se **registrará** el mensaje de log.

Lo comprobaremos en la ventana de **Logcat** (lo filtramos para localizarlo mejor):



- **Completa** todos los métodos sobrescritos (onCreate, onPause, onResume,…) con mensajes de Log que proporcionen información de los cambios de estado de la aplicación.

- **Sobreescribe** también el mètodo ***onDestroy()*** y añádele una condición que llame al método de la activity ***isFinishing(),*** que retorna un *booleano,* y que registre un mensaje de log que indique si es el **usuario** o si es el **sistema** el que va a finalizar la *activity*. (si no lo tienes claro, busca la funcionalidad del método ***isFinishing()***)

- Ahora, tienes que **ejecutar** la aplicación para que pase por **todas las transiciones de estado posibles** (comprueba desde ***LogCat*** que, efectivamente, así haya sido, **fíltralos** por el TAG para visualizar mejor las transiciones entre estados).

  Para conseguirlo, **prueba** las siguientes acciones:

  - Ejecuta tu aplicación por primera vez y pulsa el botón ***atrás***. **Observa y comprende los mensajes que se muestran.**
  - Ahora **abre** 2 o 3 **aplicaciones** más (las que prefieras).
  - De nuevo, pon tu aplicación en **primer plano**:
    - Ahora, pulsa el botón ***Home*** y **selecciona** a **otra *aplicación*** accediendo a la lista de aplicaciones abiertas, de forma que la nueva aplicación quedará en primer plano.
    - Retorna tu **aplicación** a **primer plano** y cambia la **orientación** del dispositivo (desbloquea la orientación si es necesario). **¿Qué ocurre? Observa y comprende los mensajes que se muestran en el Logcat.**
    - Devuelve el dispositivo a la orientación **vertical** y **elimina** la **aplicación** desde la lista de aplicaciones abiertas. **Observa y comprende los mensajes mostrados en el Logcat.**
  - Abre de nuevo la aplicación y **detenla** de forma **forzosa** (por ejemplo, desde la configuración del sistema). **Observa y comprende los mensajes mostrados en el Logcat.**
  - Por último, ábrela de nuevo. **Observa y comprende los mensajes mostrados.**

2. Navegación entre múltiples ventanas. Creamos una nueva *Activity*.

Normalmente, las aplicaciones que desarrollemos consistirán en más de una ventana y el usuario, además de poder interactuar para introducir/visualizar información, podrá transitar entre ellas.

En este apartado veremos como incluir **nuevas ventanas** en la aplicación *HelloWorld!* desarrollada. Esto nos permitirá, además, observar cómo se gestiona el ciclo de vida de las diferentes actividades que compondrán nuestra aplicación al transitar de una a otra.

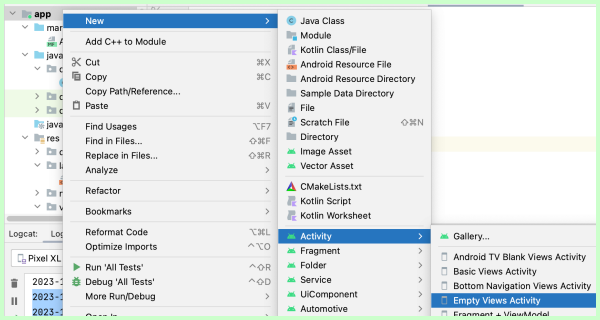
Actualmente la aplicación consiste en una única pantalla que muestra un mensaje de bienvenida. La segunda pantalla que añadiremos a la aplicación simplemente mostrará un mensaje diferente al usuario. Por lo tanto, debemos generar un nuevo *layout* que presentará la misma estructura que el definido para *MainActivity* así como su *Activity* asociada.

Podemos hacerlo de dos formas:

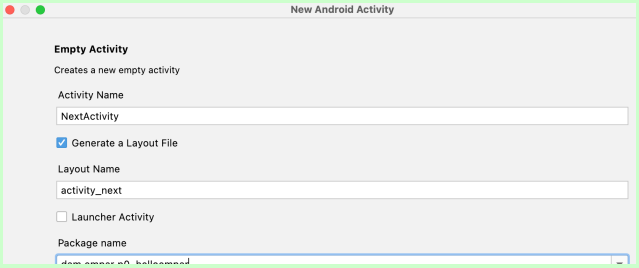
1. De forma **automática**: sobre el módulo ***app*** con el botón derecho del ratón: ***New→Activity*** (o también desde menú *File*). La *activity* se registrará **automáticamente** en el manifiesto de la aplicación y se crearán sus ficheros de *layout* y código fuente iniciales asociados.
2. De forma **manual**: copiar (*Copy*) el fichero *activity_main.xml* y pegarlo (*Paste*) en la misma carpeta. Luego crearemos su *activity*, le asociaremos el nuevo layout creado y la añadiremos **manualmente** al manifestó de la aplicación. (**NO ES RECOMENDABLE**)

Creamos una nueva activity llamada: *NextActivity*.

Por comodidad, crearemos la *activity* **automáticamente**:



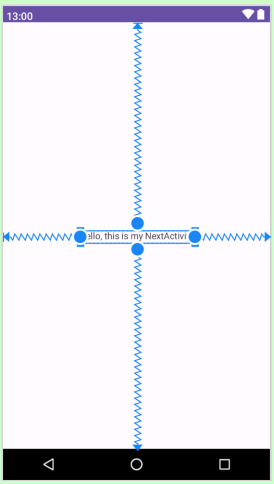
- Rellenamos los **datos** básicos para la **nueva** *activity* (dejamos el resto por defecto):



- **Observa** que queda añadida **automáticamente** a ***AndroidManifest.xml***:

```
<activity
    android:name=".NextActivity"
    android:exported="false" />
```

- **Modifica** este segundo ***layout*** para que muestre **centrado** un **mensaje** al usuario. Añade el mensaje como un recurso de cadena en *strings.xml* y utilízalo:



- **Sobrescribe** los mismos **métodos** del **ciclo de vida** que en la *activity_main* con los mensajes adecuados para esta activity. **Declara** también una **constante** para el ***TAG*** de los mensajes del ciclo de vida de esta *activity*:

```
private static final String DEBUG_TAG = "LogsAndroid-2";
```

Ahora, solo quedaría incluir algún elemento en la interfaz (UI) de la primera pantalla/*activity* que permita al usuario **abrir** la nueva pantalla/*activity* creada.

Para ello, **añadiremos un nuevo botón** al *layout* de la primera pantalla/*activity*:

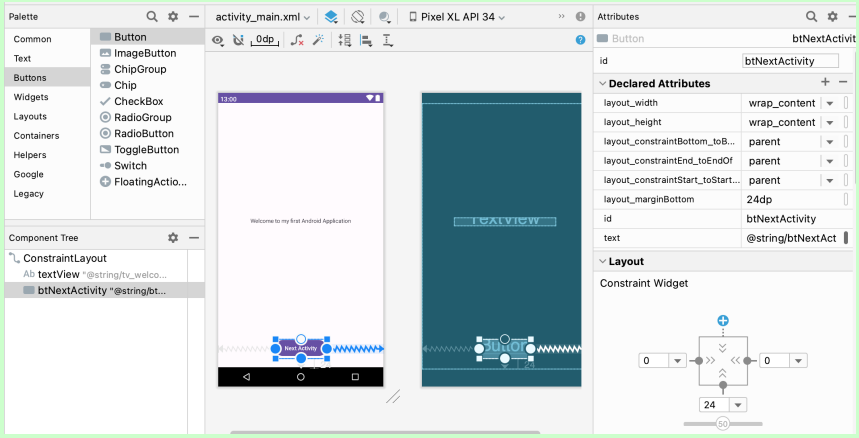
> ***ATENCIÓN:*** *Recuerda que, una vez en la segunda pantalla/activity, para volver a visualizar la primera pantalla/activity solo será necesario pulsar el* ***botón BACK (o el gesto asociado)****.*

Añadimos un botón al layout de la activity principal : *activity_main.xml*

- Abre el fichero asociado al layout de *activity_main*: ***activity_main.xml***.
- Desde la vista diseño seleccionamos en la ***Palette*** → ***Buttons*** el View ***Button*** y lo arrastramos sobre el ***ConstraintLayout*** de la ventana ***Component Tree*** (recomendado) o bien lo arrastramos directamente sobre la vista de la **interfaz gráfica**.

  El botón estará centrado en la parte inferior de la pantalla con un margen de 24p.

  Esto añadirá un nuevo componente (view) que representa un botón sobre el que poder pulsar y reaccionar a su pulsación (ver en la imagen)
- Desde ***Attributes*** cambia el ***nombre*** (**id**) del botón por defecto para utilizarlo luego desde el programa de manera cómoda: ***btNextActivity***
- Modifica los ***textos*** que aparecen como se **observa** en la siguiente **imagen** (los definimos primero en **strings.xml**)



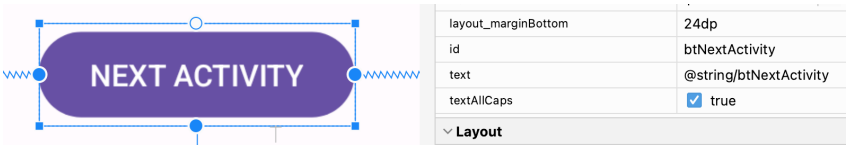
Descripción XML del botón.

```
<Button
    android:id="@+id/btNextActivity"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="24dp"
    android:text="@string/btNextActivity"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent" />
```

Fichero strings.xml

```
<resources>
    <string name="app_name">U2P3_Helloworld-LifeCycle</string>
    <string name="tv_welcome">Welcome to my first Android Application</string>
    <string name="btNextActivity">Next Activity</string>
    <string name="tv_welcome_next_activity">Hello, this is my NextActivity</string>
</resources>
```

Si lo requerimos, podremos **forzar** que el **texto** del **botón** se muestre siempre en **mayúsculas** independientemente a como se haya escrito en strings.xml. Para ello, activamos la propiedad ***textAllCaps*** del ***Button***:



Si abrimos la vista XML del layout, aparece ya la propiedad añadida al botón con valor true:

```
android:textAllCaps="true"
```

Ahora quedaría programar la **acción a realizar cuando se pulse el botón**. El botón, por defecto, no responde a ninguna acción del usuario.

Puede realizarse de **dos formas** (***de momento no modifiques nada en el código***):

- Una **primera opción** consiste en indicar de manera **explícita** en el código que el botón desea recibir los **eventos** de tipo ***Click*** y cuál es el **manejador** asociado (método) a dichos eventos. El manejador asociado deberá lanzar a ejecución (mostrar) la *Activity* correspondiente a la segunda pantalla. Él fragmento de Código 3 muestra un ejemplo de cómo podría realizarse esta asociación:

  - Para mantener limpio onCreate(), se añade un método setupUI() que realice el acceso a los componentes de UI que nos interesen.

  - Primeramente, y por medio del método ***findViewById(),*** se obtiene una referencia al *Button* a través de su identificador (tal y como se haya definido en su *layout*).

  - Luego se asocia a (***setOnClickListener()***) un nuevo manejador (***View.OnClickListener()***) para gestionar las pulsaciones realizadas sobre el mismo (sobrescribir el método *onClick()*).

    Para lanzar a ejecución una nueva *Activity* utilizaremos el método ***startActivity(),*** en el que se indicará el *Intent* a lanzar. Este *Intent* se crea especialmente para un componente concreto: *NextActivity.class* (***intent*** **explícito**), en lugar de dejar al sistema que busque una clase apropiada para ello.

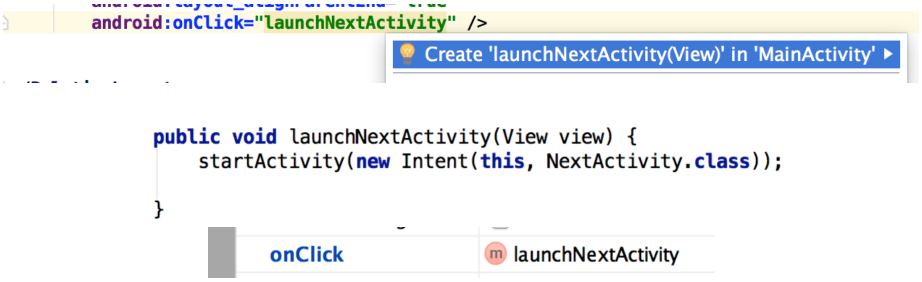
MainActivity.java

```
11  public class MainActivity extends AppCompatActivity {
12      public static final String DEBUG_TAG = "LogsAndroid-1";
13
14      @Override
15      protected void onCreate(Bundle savedInstanceState) {
16          super.onCreate(savedInstanceState);
17          setContentView(R.layout.activity_main);
18
19          Log.i(DEBUG_TAG, msg: "onCreate");
20
21          setupUI();
22      }
23
24      private void setupUI() {
25          Button btNextActivity = findViewById(R.id.btNextActivity);
26
27          btNextActivity.setOnClickListener(new View.OnClickListener() {
28              @Override
29              public void onClick(View v) {
30                  startActivity(new Intent( packageContext: MainActivity.this, NextActivity.class));
31              }
32          });
33      }
```

Código 2. Manejador del evento onClick generado por el botón.

- La **segunda opción**, consiste en definir en el código de la Activity un **método público** que acepte como único argumento un objeto de tipo *View* (ver el fragmento de Código 4), y asignar este método al atributo ***android:onClick*** en la descripción XML del botón en el *layout* correspondiente.

  También lo podemos **crear automáticamente** desde el código **xml** correspondiente al layout de la activity:

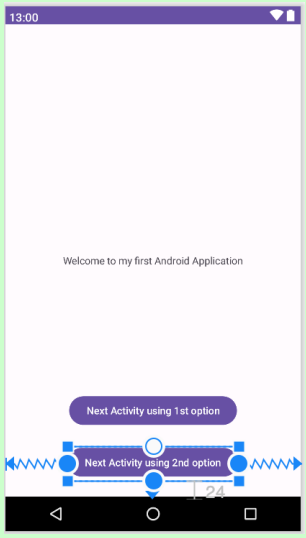


Código 3. Método que se asociará como manejador de eventos Clic.

Damos funcionalidad al *Button*

- **Añade** el **código** correspondiente al clic del botón, para **abrir** la "Activity Next" siguiendo la ***primera opción*** explicada.
- **Añade** un **nuevo botón** al layout (`btNextActivity2`) así como el **código** para **abrir** la "Activity Next" siguiendo la ***segunda opción***.

  Los botones se diseñaran como se muestra en la imagen:



- **Ejecuta** la aplicación y trata de que realice todas las **transiciones** posibles entre las dos *activities*. Comprueba que ambos botones funcionan correctamente y transitan a NextActivity.
- **Comprueba y analiza** en el ***LogCat*** que se muestran todos los mensajes correspondientes cuando se realiza el cambio de una a otra *Activity*.

Actividad 1

Busca información acerca de los métodos *onSaveInstanceState*() y *onRestoreInstanceState*().

```
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
}

@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
}
```

Nota: Bundle es una clase similar a un diccionario.

- ¿En qué **circunstancias** se llama a estos **métodos**? Explícalo con tus palabras. (con un comentario en el código).
- Comprueba su funcionamiento **sobrescribiéndolos** en **todas** las *activity*:
    - Añade un **mensaje** de **log** que indique a qué método concreto se llama.
    - Elige, de los dos métodos anteriores, el más adecuado para guardar en el Bundle que recibe una **cadena** con tu **nombre** y muestra, desde el método adecuado de los dos anteriores, un mensaje de log con la **cadena recuperada** (que contiene tu nombre).
    - Realiza las **pruebas** necesarias en el emulador/dispositivo para comprobar su correcto funcionamiento. Incluye un **comentario** en el código con el detalle de las **pruebas** que has realizado.

Actividad 2

- Añade una **nueva *activity*** a la aplicación llamada: MyProfile. Contendrá, los TextView, etc. necesarios para mostrar **tus datos**.

  Ten en cuenta lo siguiente:
    - Utiliza `strings.xml` para guardar los textos que utilices.
    - Realiza el diseño del `layout` de la forma más "adecuada".
    - Sobrescribe también todos los métodos del ciclo de vida, incluidos los métodos de la Actividad 1, y añade mensajes de log adecuados.
- **Añade** a NextActivity un **botón** para **abrir** la nueva *activity* **MyProfile**. (utiliza la opción que prefieras para abrirla)

Actividad 3

- En todas las *activity*, redefine el texto de la etiqueta ***DEBUG_TAG*** para asignarle el **nombre** de la ***activity* actual**. (deberás buscar un **método** del API que obtiene el nombre de la *activity*)
- **Ejecuta** la aplicación y **comprueba** que se muestran los mensajes de Log correspondientes al estado teniendo como TAG el nombre de la *activity* actual.