

Tema 3: Desarrollo de Aplicaciones ▶ para móviles

Laura Sacristán Matesanz

Objetivos del tema

Entender la estructura de una aplicación móvil

Profundizar en el Desarrollo de aplicaciones móviles

Conocer controles avanzados de las aplicaciones móviles

Debug de una aplicación Android I

- ▶ Nos ayuda a depurar un programa cuando se produce algún fallo o error y detectar dónde se está produciendo.
- ▶ Para ello utilizaremos los BREAK POINT.
- ▶ Break Point: es un punto del programa donde queremos que se detenga el programa para poder inspeccionar los valores de las variables.

Debug de una aplicación Android II

- ▶ Vamos a utilizar el ejemplo del ciclo de vida para ver como funcionan los BreakPoint.
- ▶ Lo primero que vamos a hacer es añadir un variable privada de tipo integer y la inicializaremos a 0 en el método onCreate.

```
HelloActivity onCreate()
3  import android.support.v7.app.AppCompatActivity;
4  import android.os.Bundle;
5  import android.util.Log;
6
7  public class HelloActivity extends AppCompatActivity {
8      private int i;
9
10     @Override
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.activity_hello);
14
15         Log.i("TAG cicloVida ", "Ciclovida: onCreate");
16
17         i = 0;
18     }
```

```
activity_main.xml x MainActivity.java x

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    i = 0;
}

@Override
protected void onStart (){
    super.onStart();
    Log.i( tag: "TAG:cicloVida", msg: "Ciclovida:onStart");
    i++;
}

@Override
protected void onResume (){
    super.onResume();
    Log.i( tag: "TAG:cicloVida", msg: "Ciclovida:onResume");
    i++;
}

@Override
protected void onPause (){
    super.onPause();
    Log.i( tag: "TAG:cicloVida", msg: "Ciclovida:onPause");
    i++;
}

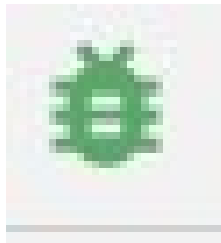
@Override
protected void onStop (){
    super.onStop();
    Log.i( tag: "TAG:cicloVida", msg: "Ciclovida:onStop");
    i--;
}
}
```

Debug de una aplicación Android III

- ▶ A continuación, en cada uno de los métodos que creamos en su momento, onStart, onResume, onPause, onStop y onDestroy vamos a ir incrementando la variable i para ver el valor que tiene.
- ▶ En el caso de los método onStop y onDestroy la decrementaremos para ver la diferencia.

Debug de una aplicación Android IV

- ▶ Una vez modificado el código, vamos a ejecutar la aplicación en modo Debug. Para ello pulsaremos el siguiente icono, que aparece en la barra de herramientas de la aplicación.



- ▶ Antes de pulsar esta opción tendremos que poner distintos puntos de BreakPoint para que si el programa pasa por ese punto se pare y podamos comprobar el valor de las variables de nuestro programa.

Debug de una aplicación Android V

- Vamos a poner un BreakPoint en cada uno de los métodos, por ejemplo donde tenemos el log. Para poner esos puntos tendremos que pulsar en el margen izquierdo donde queramos ponerlo.



The screenshot shows the MainActivity.java file in an Android Studio IDE. The file is open, and the code is displayed with line numbers from 10 to 41. The code implements the lifecycle methods of an Android Activity. Breakpoints are set in the left margin of the IDE at the following line numbers: 12, 14, 18, 20, 21, 24, 26, 27, 30, 32, 33, 36, 38, 39, and 40. The code is as follows:

```
10 |
11 |     @Override
12 |     protected void onCreate(Bundle savedInstanceState) {
13 |         super.onCreate(savedInstanceState);
14 |         setContentView(R.layout.activity_main);
15 |         i = 0;
16 |     }
17 |     @Override
18 |     protected void onStart (){
19 |         super.onStart();
20 |         Log.i( tag: "TAG:cicloVida", msg: "Ciclovida:onStart");
21 |         i++;
22 |     }
23 |     @Override
24 |     protected void onResume (){
25 |         super.onResume();
26 |         Log.i( tag: "TAG:cicloVida", msg: "Ciclovida:onResume");
27 |         i++;
28 |     }
29 |     @Override
30 |     protected void onPause (){
31 |         super.onPause();
32 |         Log.i( tag: "TAG:cicloVida", msg: "Ciclovida:onPause");
33 |         i++;
34 |     }
35 |     @Override
36 |     protected void onStop (){
37 |         super.onStop();
38 |         Log.i( tag: "TAG:cicloVida", msg: "Ciclovida:onStop");
39 |         i--;
40 |     }
41 | }
```

Debug de una aplicación Android VI

- ▶ Pulsamos en la opción de debug y nos pedirá el emulador en el que queremos probarlo. Cada uno arrancará el que tenga instalado.
- ▶ Nos aparecerá la siguiente información:
 - ▶ El programa para en el primer BreakPoint marcado, en mi caso en la línea 14.
 - ▶ Aparece el valor que tiene nuestra variable i. En este punto la variable i solo está definida y al ser un integer su valor es cero.
 - ▶ También aparece el objeto This en el que se incluyen todas las propiedades definidas a nivel de clase. Aquí también aparece nuestra variable i.

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help Aplicacion Hello Word - MainActivity.java [Aplicacion_Hello_Word.app]

AplicacionHelloWord > app > src > main > java > com > example > aplicacionhelloworld > MainActivity

Android > app > manifests > java > com.example.aplicacionhelloworld > MainActivity

```
10
11
12 @Override
13 protected void onCreate(Bundle savedInstanceState) { savedInstanceState: null
14     super.onCreate(savedInstanceState); savedInstanceState: null
15     setContentView(R.layout.activity_main);
16     i = 0;
17 }
18 @Override
19 protected void onStart (){
20     super.onStart();
21     Log.i( tag: "TAG:cicloVida", msg: "Ciclovida:onStart");
22     i++;
23 }
24 @Override
25 protected void onResume (){
26     super.onResume();
27     Log.i( tag: "TAG:cicloVida", msg: "Ciclovida:onResume");
28     i++;
29 }
30 @Override
31 protected void onPause (){
32     super.onPause();
33 }
```

Debug: app

Debugger Console

Frames

- "main" @ 15,7...n": RUNNING
- onCreate:14, MainActivity (com.example.aplicacionhelloworld)
- performCreate:7994, Activity (android.app)
- performCreate:7978, Activity (android.app)
- callActivityOnCreate:1309, Instrumentation (android.app)
- performLaunchActivity:3422, ActivityThread (android.app)
- handleLaunchActivity:3601, ActivityThread (android.app)

Variables

- this = (MainActivity@16078)
- savedInstanceState = null
- i = 0

Launch succeeded (12 minutes ago)

Event Log Layout Inspector

14:1 LF UTF-8 4 spaces

Debug de una aplicación Android VII

- ▶ Dentro de la opción Debug hay varias opciones:



Siguiente línea de código



Seguir un paso hacia adentro en el caso de que queramos entrar en el método



Volver al BreakPoint en el que estamos

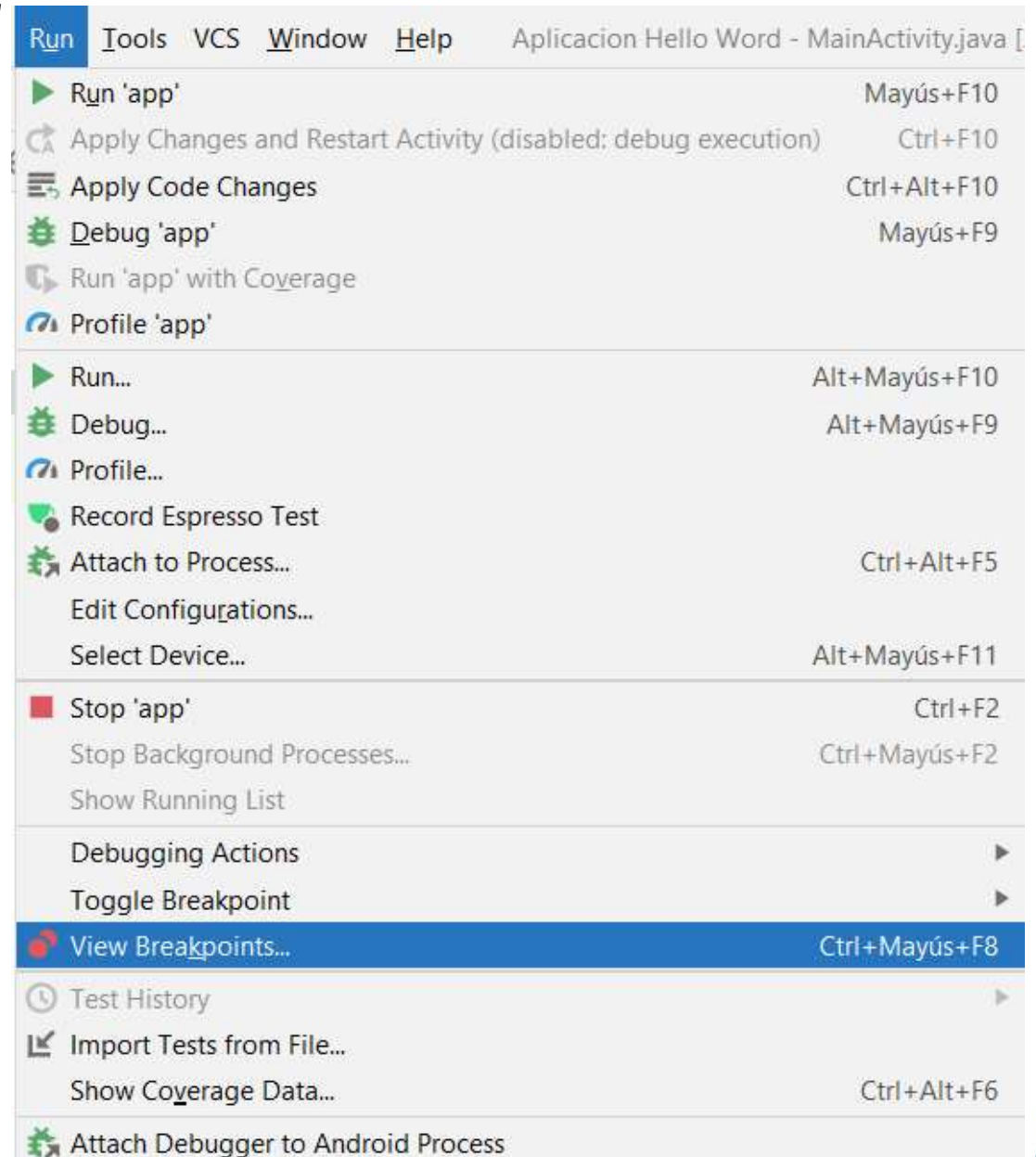


Siguiente punto de interrupción

- ▶ Como ya vimos en el ciclo de vida de una aplicación, podemos ir viendo los resultados en la opción logcat y elegir ver los TAG.

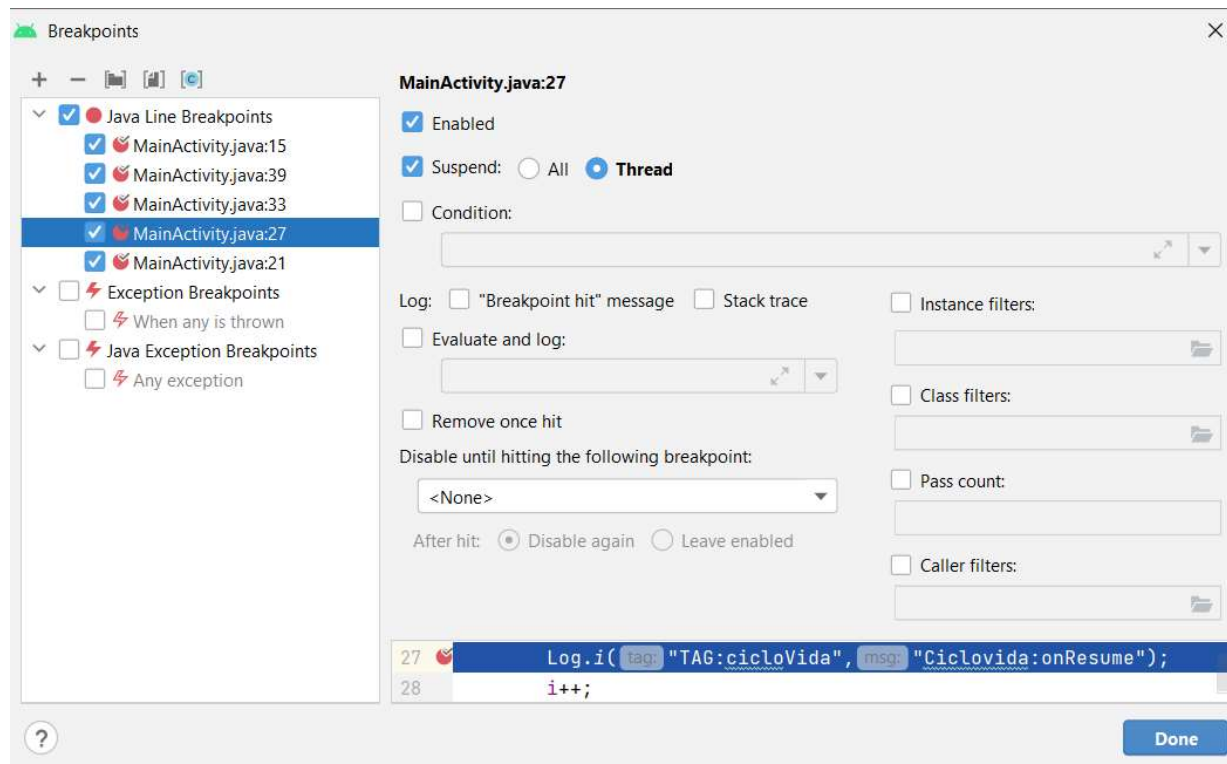
Debug de una aplicación Android VIII

- ▶ En la opción de Debug en la zona de variables iremos viendo el resultado de las variables.
- ▶ Si tenemos muchos puntos de interrupción, podemos gestionarlos a través de la opción run del menú y elegir la opción View Breakpoint



Debug de una aplicación Android IX

- ▶ Cuando pulsamos esa opción nos aparece la siguiente pantalla, desde la que podemos gestionar esos puntos de interrupción.

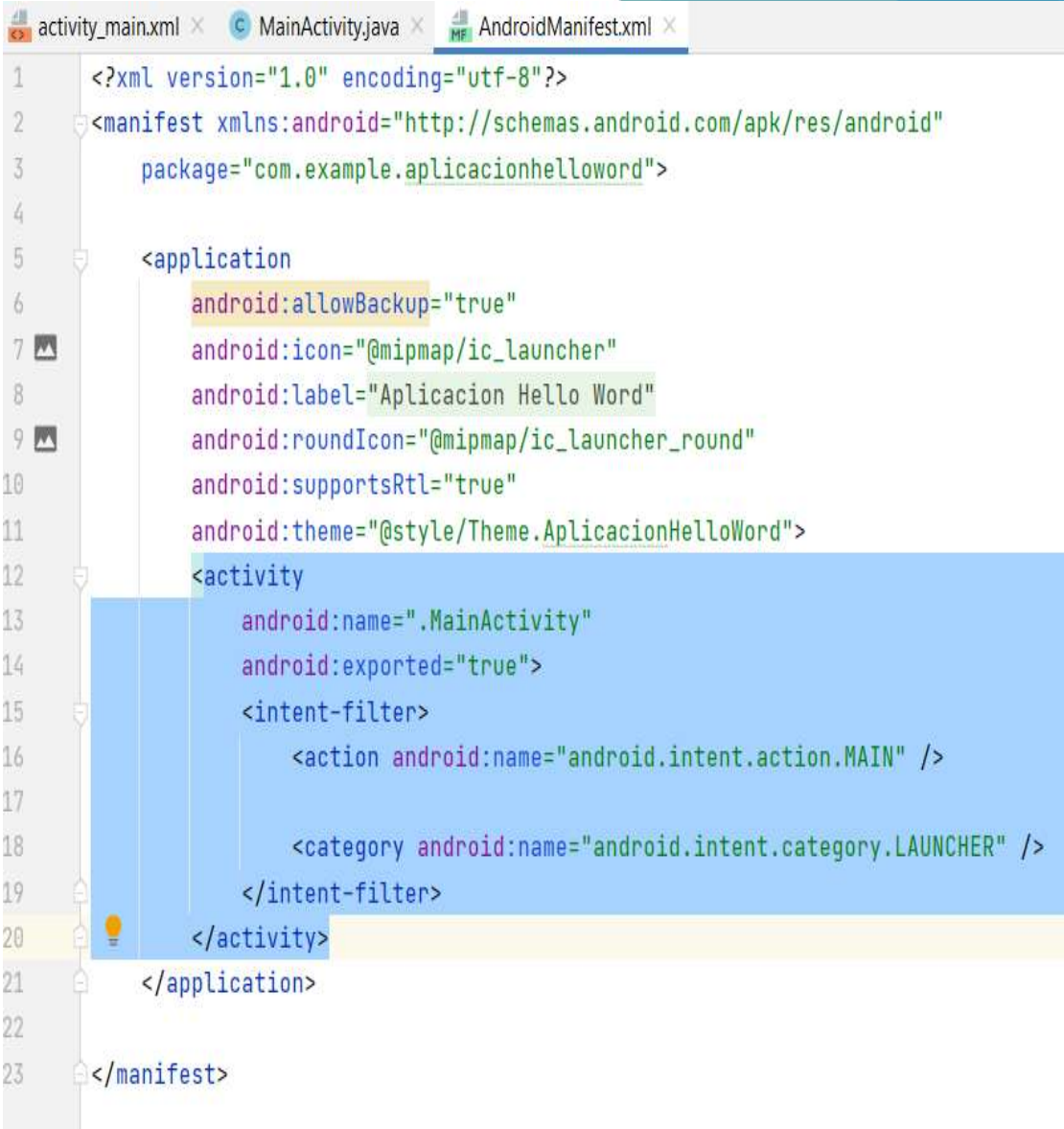


Creación de un Activity Secundario



Activity secundario I

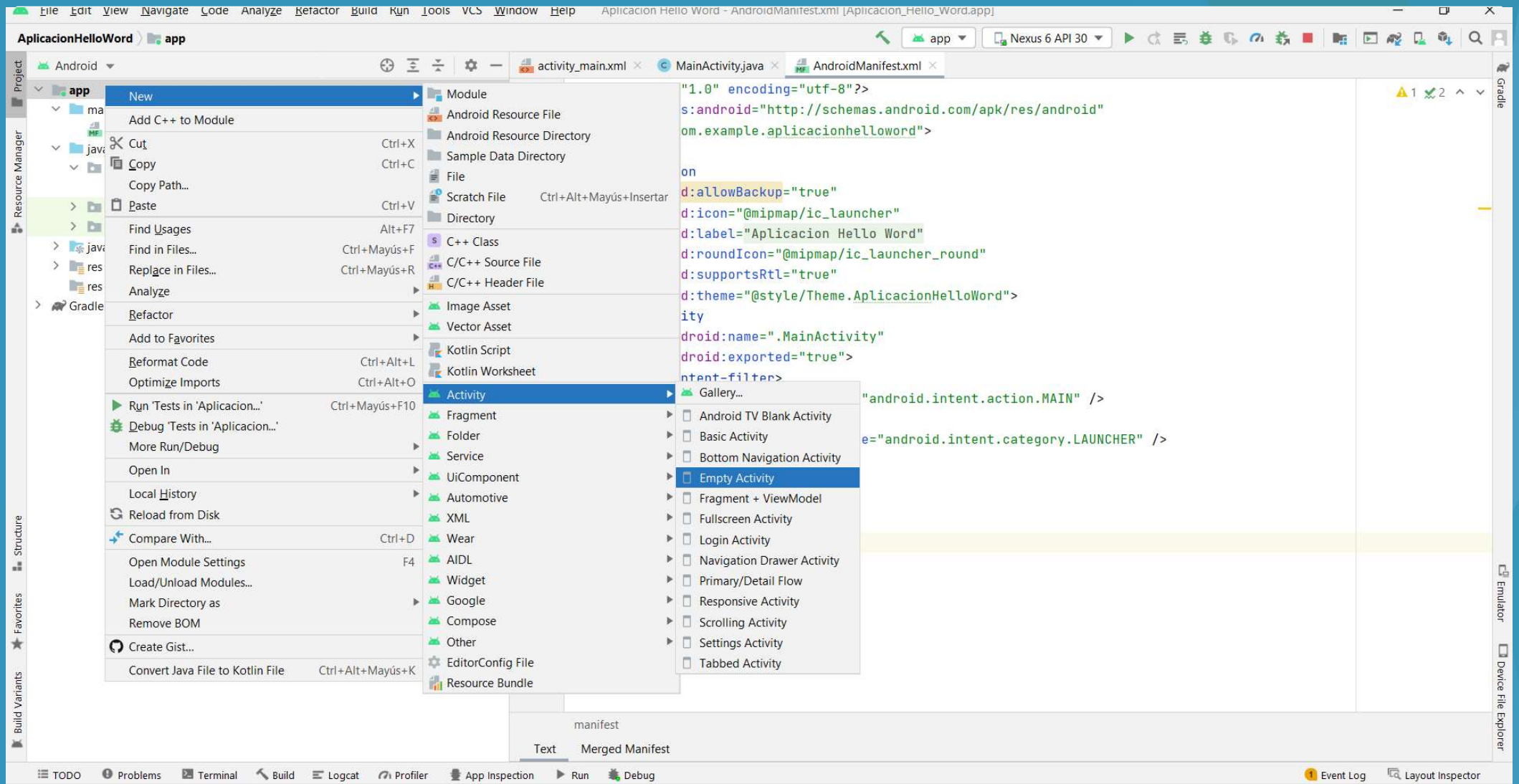
- ▶ Para crear una nueva pantalla en nuestra aplicación para móviles necesitaremos tener un Activity Secundario.
- ▶ Necesitaremos saber también como pasar de una pantalla a otra para poder utilizarlo en nuestra aplicación.
- ▶ Cuando nosotros abrimos una aplicación nueva desde cero, únicamente tenemos un Activity Principal que viene indicado en nuestro AndroidManifest con la etiqueta <activity> y dentro de ella la etiqueta <intent-filter>



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.example.aplicacionhelloworld">
4
5     <application
6         android:allowBackup="true"
7         android:icon="@mipmap/ic_launcher"
8         android:label="Aplicacion Hello Word"
9         android:roundIcon="@mipmap/ic_launcher_round"
10        android:supportsRtl="true"
11        android:theme="@style/Theme.AplicacionHelloWord">
12        <activity
13            android:name=".MainActivity"
14            android:exported="true">
15            <intent-filter>
16                <action android:name="android.intent.action.MAIN" />
17
18                <category android:name="android.intent.category.LAUNCHER" />
19            </intent-filter>
20        </activity>
21    </application>
22
23 </manifest>
```

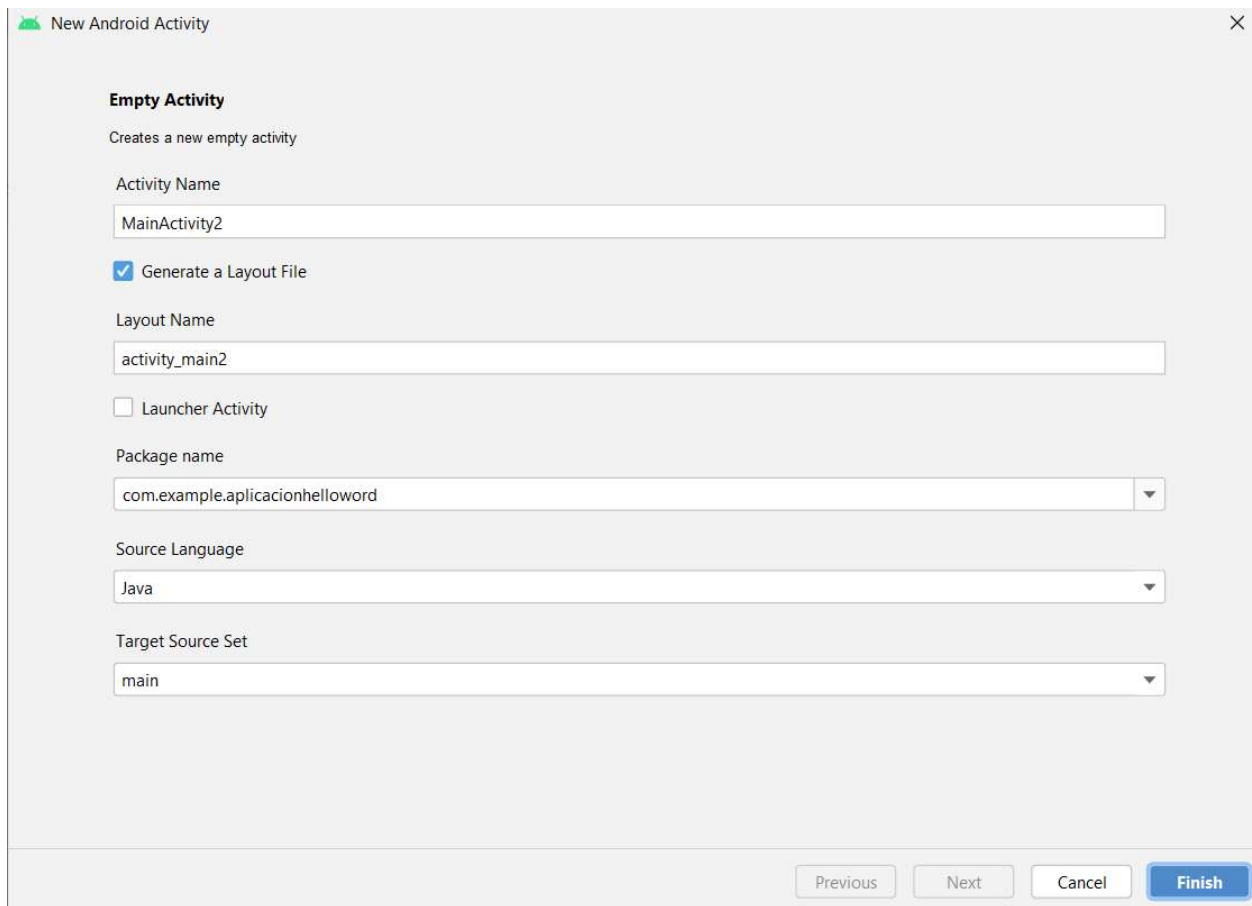

Activity secundario II

- ▶ Para que los Activity existan deben estar declarados en el archivo AndroidManifest.
- ▶ Todos los ficheros .java que definen un activity deben estar declarados en el archivo AndroidManifest. Si no están declarados es como si no existieran.
- ▶ Para crear un nuevo Activity podemos hacerlo desde la carpeta app, botón derecho y elegir la opción new y dentro de new Activity.
- ▶ Elegiremos la opción Empty Activity, que es el modelo que hemos seleccionado para todos nuestros ejemplos hasta ahora.
- ▶ También podemos elegir la opción de Gallery y nos mostrará la pantalla de Activity que nos muestra al inicio de una aplicación nueva



Activity secundario III

- En cualquiera de los dos casos, la siguiente pantalla que nos aparece es



New Android Activity

Empty Activity
Creates a new empty activity

Activity Name
MainActivity2

☒ Generate a Layout File

Layout Name
activity_main2

☐ Launcher Activity

Package name
com.example.aplicacionhelloworld

Source Language
Java

Target Source Set
main

Previous Next Cancel Finish

Activity secundario IV

- ▶ En esta pantalla tendremos que poner un nombre al Activity. Lo ideal es poner un nombre que describa la función que va a tener o realizar y terminado siempre con la palabra Activity. De esta manera será más sencillo identificarlas.
- ▶ En este caso por ejemplo vamos a llamarlo SecundarioActivity.
- ▶ Se creará también el archivo .XML que será la interfaz de usuario de esta Activity.
- ▶ También tenemos la opción de asignar a este Activity que se el principal o inicial. No es recomendable marcar esta opción, ya que podría crear conflictos de programación porque ya tenemos otro Activity inicial
- ▶ En la siguiente pantalla veremos que se nos han creado los dos nuevos archivos.

AplicacionHelloWord > app > src > main > java > com > example > aplicacionhelloworld

Project

Resource Manager

- Android
- app
 - manifests
 - AndroidManifest.xml
 - java
 - com.example.aplicacionhelloworld
 - MainActivity
 - SecundarioActivity
 - com.example.aplicacionhelloworld (androidTest)
 - com.example.aplicacionhelloworld (test)
 - java (generated)
 - res
 - drawable
 - layout
 - activity_main.xml
 - activity_secundario.xml
 - mipmap
 - values
 - res (generated)
 - Gradle Scripts

Activity secundario V

- ▶ Si nos fijamos en los archivos MainActivity y el SecundarioActivity los dos llaman al método setContentView.
- ▶ Este método lo que hace es cargar en la pantalla del dispositivo la interfaz de usuario de cada uno de los Activity.
- ▶ Como ya os habréis dado cuenta, para referenciar al fichero XML, utilizamos un fichero estático, que es el fichero R que contiene todas las referencias a las variables y a los ficheros que tenemos en nuestro directorio de recursos (la carpeta res)
- ▶ Es un archivo que genera automáticamente AndroidStudio cada vez que creamos una nueva aplicación.

Activity secundario VI Actividad

BUSCAR EL FICHERO R DENTRO DE NUESTRA APLICACIÓN.

UNA VEZ LO HAYAS ENCONTRADO, BUSCA DENTRO DE ÉL EL BLOQUE ESTÁTICO LLAMADO LAYOUT.

¿HAY ALGO QUE TE RESULTE CONOCIDO DENTRO DE ESTE BLOQUE?

¿QUÉ OCURRE SI BORRAMOS UNA DE LAS LÍNEAS DE ESTE BLOQUE?

¿ME PRODRÍAS EXPLICAR QUE SIGNIFICA ENTONCES R.LAYOUT.ACTIVITY_SECUNDARIO?

ESPERO VUESTRAS RESPUESTAS

Activity secundario VII Actividad

Crear una aplicación Android con dos activity.

La primera mostrará un botón que, al pulsarlo, lance la segunda.

El segundo activity mostrará la imagen que vosotros decidáis.

Observad que al matar el activity 2 se mostrará el activity principal.

Introducción a eventos.

Evento onClick



Introducción a eventos I

- ▶ Para poder navegar por varias pantallas dentro de nuestra aplicación necesitaremos configurar eventos.
- ▶ Hay varias formas de poder configurarlo pero vamos a utilizar la más sencilla y que ya hemos realizado en algunos de nuestros ejemplos.
- ▶ Para ello nos iremos a nuestro layout principal y de nuestro ejemplo HelloWorld.
- ▶ Dentro de ese archivo XML solo tenemos declarado un TextView en el que aparece el texto de Hola Mundo.
- ▶ El atributo que vamos a configurar es el atributo onClick de nuestro TextView.
- ▶ El valor que le vamos a dar es `initSecActivity` que hará referencia a nuestro activity secundario


```
activity_main.xml x MainActivity.java x AndroidManifest.xml x activity_secundario.xml x SecundarioActivity.java x

1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context=".MainActivity">
8
9     <TextView
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:onClick="initSecActivity"
13         android:text="Hello World!"
14         app:layout_constraintBottom_toBottomOf="parent"
15         app:layout_constraintLeft_toLeftOf="parent"
16         app:layout_constraintRight_toRightOf="parent"
17         app:layout_constraintTop_toTopOf="parent" />
18
19 </androidx.constraintlayout.widget.ConstraintLayout>
```

Pixel 31 AplicacionHelloWord Default (en-us)

Attributes

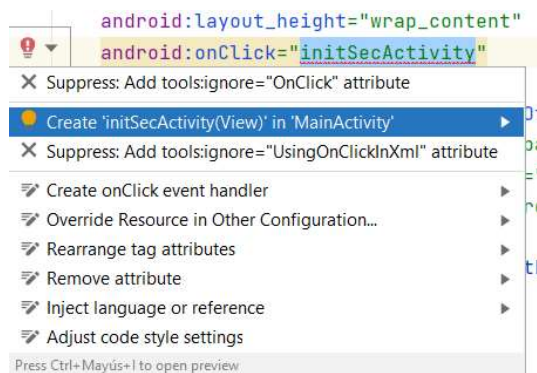
Attribute	Value
marqueeRepeatLi...	
maxEms	
maxLength	
maxLines	
minEms	
minLines	
nestedScrollingEn...	
nextFocusDown	
nextFocusForward	
nextFocusLeft	
nextFocusRight	
nextFocusUp	
numeric	
onClick	initSecActivity
orientation	
outlineProvider	
overScrollMode	
padding	[?, ?, ?, ?]
password	
phoneNumber	
privateImeOptions	
requiresFadingEd...	
rotation	
rotationX	

Event Log Layout Inspector

12:42 (34 chars) LF | jueves, 4 de noviembre de 2021

Introducción a eventos II

- ▶ Como hemos visto en las imágenes lo que definimos en los atributos del entorno gráfico, aparecerá en el programa XML.
- ▶ El valor que aparece en onClick debe ser el nombre de un método que exista en el archivo MainActivity. Como en este caso no aparece, nos lo marca como un error.
- ▶ Si pulsamos en la bombilla amarilla que aparece, automáticamente nos crea un método en el archivo MainActivity con la siguiente estructura.



```
public void initSecActivity(View view) {  
    }  
}
```

Introducción a eventos III

- ▶ Para comprobar que al ejecutar la aplicación entra en el método vamos a utilizar una nueva forma.
- ▶ En lugar de utilizar el log que utilizamos en el ciclo de vida, vamos a usar la función TOAST. Esta función lo que hace es mostrar un mensaje emergente durante unos segundos en la pantalla del usuario.
- ▶ Vamos a crear uno nuevo (opción que aparece en autocompletar).



Introducción a eventos IV

- ▶ This nos va a definir el contexto en el que estamos lanzando este mensaje. Como estamos dentro del MainActivity es suficiente con poner la palabra reservada this porque hará referencia al propio Activity.
- ▶ En el segundo parámetro vamos a escribir el texto: “Has hecho click en el texto” que es el mensaje emergente que nos mostraría.
- ▶ El último parámetro define el número de segundos que va a aparecer el mensaje en la pantalla. Tenemos dos opciones:
 - ▶ Toast.LENGTH_SHORT: que lo mostrará dos segundos.
 - ▶ Toast.LENGTH_LONG: que lo mostrará cuatro segundos.

- ▶ Nos quedaría algo así:

```
public void initSecActivity(View view) {  
    Toast.makeText( context: this, text: "Has hecho click en el texto", Toast.LENGTH_LONG).show();  
}  
}
```

Introducción a eventos V

- El resultado es el siguiente:



Introducción a eventos VI

- ▶ Hay algunas formas más de hacer este proceso:
 - ▶ Definiendo un identificador para el componente, utilizando el atributo id del elemento.
 - ▶ Después creamos una variable del mismo tipo, en este caso TextView y lo asociamos al elemento visual.
 - ▶ Una vez realizado lo anterior, debemos crear un escuchador de eventos.
 - ▶ Todos estos pasos quedarían así:

```
<TextView
    android:id="@+id/txtEvento"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

:./androidx.constraintlayout.widget.ConstraintLayout>
```

```
public class MainActivity extends AppCompatActivity {
    private int i;
    TextView texto;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.i( tag: "TAG:cicloVida", msg: "Ciclovida:onCreate");
        i = 0;
        texto = (TextView) findViewById(R.id.txtEvento);
        texto.setOnClickListener();
    }
}
```

Introducción a eventos VI

En este punto podemos elegir dos formas diferentes de realizar el proceso:

- ▶ La primera es gestionar, en el propio escuchador, una clase anónima que definiría el evento click.
- ▶ Dentro del escuchador se generaría un método onClick similar al que hemos realizado antes.

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    Log.i( tag: "TAG:cicloVida", msg: "Ciclovida:onCreate");  
    i = 0;  
    texto = (TextView) findViewById(R.id.txtEvento);  
    texto.setOnClickListener(new View.OnClickListener() {  
        @Override  
        public void onClick(View view) {  
            Toast.makeText( context: MainActivity.this, text: "Has hecho click en el escuchador", Toast.LENGTH_LONG).show();  
        }  
    });  
}
```


Introducción a eventos VII

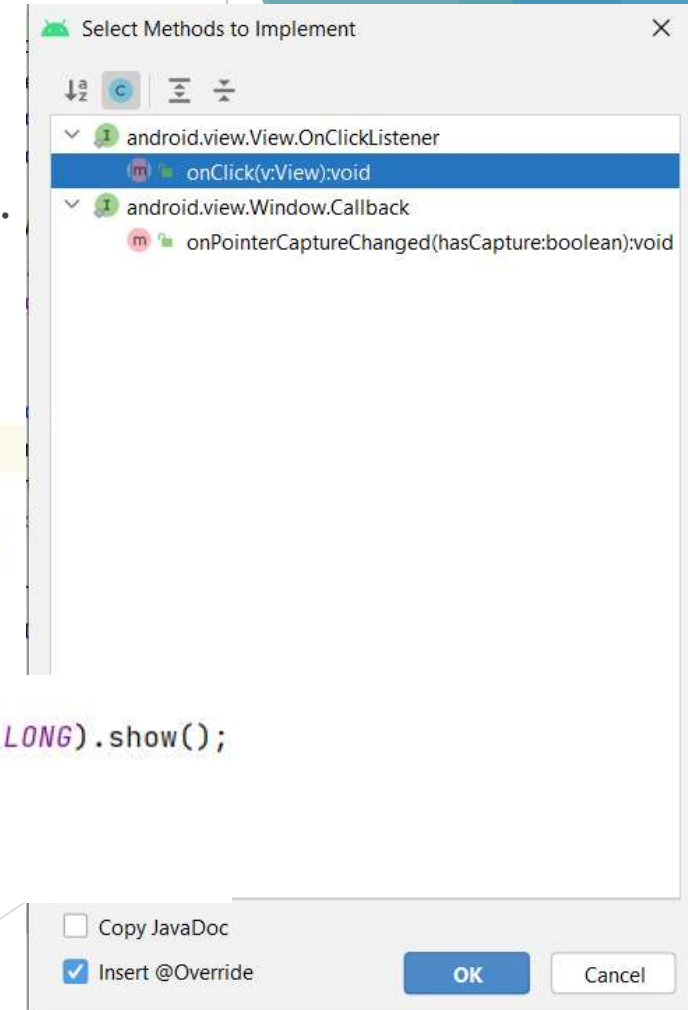
- ▶ La segunda es declarar en la clase la opción de gestionar eventos click.
- ▶ Quedaría de la siguiente manera:

```
texto = (TextView) findViewById(R.id.txtEvento);  
texto.setOnClickListener(this);
```

❗ Cast parameter to 'android.view.View.OnClickListener'
❗ Make 'MainActivity' implement 'android.view.View.OnClickListener'
Press Ctrl+Mayús+I to open preview

override

```
public void onClick(View view) {  
    Toast.makeText(context: this, text: "Has hecho click en el texto", Toast.LENGTH_LONG).show();  
}
```



Las Intents.



Intent

- ▶ La clase Intent nos permite iniciar nuevas actividades en nuestra aplicación.
- ▶ Tenemos dos tipos
 - ▶ Explícitas: que son activity que existen en nuestra aplicación
 - ▶ Implícitas: que son activity que existen en otra aplicación.
- ▶ Comenzaremos las intent explícitas.

Intent explícitas I

- ▶ Las intents explícitas especifican qué aplicación las administrará, ya sea incluyendo el nombre del paquete de la app de destino o el nombre de clase del componente completamente calificado.
- ▶ Normalmente, el usuario usa una intent explícita para iniciar un componente en su propia aplicación porque conoce el nombre de clase de la actividad o el servicio que desea iniciar.
- ▶ Por ejemplo, puedes utilizarla para iniciar una actividad nueva en respuesta a una acción del usuario o iniciar un servicio para descargar un archivo en segundo plano.

Intent explícitas II

- ▶ Vamos a crear un proyecto nuevo que vamos a llamar IntentExplicita.
- ▶ Vamos a crear un segundo Activity que llamaremos DatosActivity.
- ▶ En el activity-main vamos a generar un botón que vamos a llamar Iniciar Datos Activity. En este botón vamos a asignar un método al onClick que se va a llamar InicarActivity.
- ▶ Ahora vamos a crear el método IniciarActivity en el fichero MainActivity.
- ▶ Cuando nosotros hagamos click en el botón se lanzará el método IniciarActivity.
- ▶ Nuestro método contendrá el siguiente código

Intent explícitas III

```
16  
17 public void IniciarActivity(View view) {  
18     Intent intentDatos = new Intent( packageContext: this, DatosActivity.class);  
19     startActivity(intentDatos);  
20 }  
21 }
```

- ▶ La primera línea del método es para crear una nueva intent. Si no estuviera la clase Intent, habría que importarla.
 - ▶ Dentro del constructor indicamos en primer lugar el contexto, que hace referencia al Activity sobre el que estamos trabajando. Como en este caso estamos trabajando sobre MainActivity, podemos poner la palabra reservada this.
 - ▶ En el segundo parámetro, pondremos la clase java que queremos lanzar, en nuestro caso DatosActivity.class.
- ▶ La segunda línea es la que nos indica la ejecución de Intent. Incluimos como parámetro la intent que hemos definido en la línea anterior.

Intent explícitas IV

- ▶ Para comprobar que funciona correctamente, podemos modificar el Activity secundario y poner algún texto significativo (por ejemplo: estas en DatosActivity) para distinguir si está accediendo a él cuando pulsamos el botón del Activity principal.
- ▶ Una vez modificado, podemos lanzar la aplicación y comprobamos que funciona correctamente.

Intent explícitas V

- ▶ A través de las Intent también podemos pasar parámetros.
- ▶ Se añadirían entre la línea de declaración de la intent y el lanzamiento.
- ▶ Se pondrían tantas líneas de código como parámetros queramos declarar.
- ▶ Si por ejemplo queremos pasar por parámetro un nombre o un número pondríamos los siguiente:
 - ▶ `intentDatos.putExtra ("numero", 1);` o `intentDatos.putExtra ("nombre", "Laura");`
 - ▶ La estructura siempre es igual: primero el nombre de la variable que queremos pasar como parámetro al siguiente activity y después su valor.

Intent explícitas VI

- ▶ Para recuperar esos parámetros en el siguiente activity tendríamos que hacer lo siguiente:

- ▶ Declaramos un Bundle que es un conjunto de variables que podemos recuperar a través del nombre con el que hemos definido los parámetros. En el caso de nuestro ejemplo número y nombre.

```
Bundle extras = getIntent().getExtras();
```

- ▶ Para obtener valores tendremos que utilizar la función getIntent.
- ▶ Después recuperaremos nuestros datos. Para ello pondremos las siguientes líneas de código:

```
int n = extras.getInt("numero");
```

```
String s = extras.getString ("nombre");
```

Para mostrarlo por pantalla usaremos un TOAST

El código de programación sería

```
activity_main.xml x MainActivity.java x activity_datos.xml x DatosActivity.java x
package com.example.intentexplicita;

import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

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

    public void IniciarActividad(View view) {
        Intent intentDatos = new Intent( packageContext: this, DatosActivity.class);
        intentDatos.putExtra ( name: "numero", value: 1);
        intentDatos.putExtra ( name: "nombre", value: "Laura");

        startActivity(intentDatos);
    }
}

activity_main.xml x MainActivity.java x activity_datos.xml x DatosActivity.java x
package com.example.intentexplicita;

import ...

public class DatosActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_datos);
        Bundle extras = getIntent().getExtras();
        int n = extras.getInt( key: "numero");
        String s = extras.getString( key: "nombre");
        Toast.makeText( context: this, text: "Numero "+n+" y nombre "+s, Toast.LENGTH_LONG).show();
    }
}
```



El resultado visual

Intent implícitas I

- ▶ Las Intent implícitas no lanzan un activity concreto si no que se lanza una acción genérica.
- ▶ Permite llamar al S.O. o a cualquier otra aplicación para que resuelva nuestra necesidad. Por ejemplo reproducir un archivo de sonido o un vídeo, llamadas o direcciones en un mapa y muchas otras opciones más.
- ▶ Nos van a ahorrar mucha implementación, ya que no siempre es necesario realizar todas esas funcionalidades desde cero.
- ▶ Para verlo más claro vamos a realizar un ejemplo. En este caso vamos a realizar una alarma en nuestro smartphone.

Intent implícitas II

- ▶ Creamos un método que se llama `createAlarm` que tendrá tres parámetros:
 - ▶ El primero es el parámetro `message` que será el nombre que aparezca cuando se active la alarma.
 - ▶ Los siguientes son la hora y los minutos en los que queremos que se active la alarma.
- ▶ En la siguiente línea de código aparece el `startActivity` que es el método para llamar a las `INTENT`
- ▶ Un ejemplo de código sería el siguiente:

Intent implícitas III

- Para ello vamos a escribir el siguiente código:

```
public void createAlarm(String message, int hour, int minutes) {  
    Intent intent = new Intent(AlarmClock.ACTION_SET_ALARM)  
        .putExtra(AlarmClock.EXTRA_MESSAGE, message)  
        .putExtra(AlarmClock.EXTRA_HOUR, hour)  
        .putExtra(AlarmClock.EXTRA_MINUTES, minutes);  
    startActivity(intent);  
}
```

Para llamar al intent lo hacemos igual que lo hacíamos con los explícitos con el método startActivity.

Intent implícitas IV

- ▶ Para poder utilizar este código en uno de nuestras aplicaciones, vamos a crear un botón en el `activityMain` de nuestro `Main` principal.
- ▶ A este botón le vamos a llamar `Crear Alarma` a las 12:15.
- ▶ A este botón le vamos a asociar un evento `onClick` con un método `crearAlarma` que crearemos en nuestro fichero `MainActivity`.
- ▶ En ese método vamos a poner el código que hemos explicado en las diapositivas anteriores.
- ▶ La programación quedará de la siguiente manera:

Intent implícitas V

```
public void crearAlarma(View view) {  
    Intent intent = new Intent(AlarmClock.ACTION_SET_ALARM)  
        .putExtra(AlarmClock.EXTRA_MESSAGE, value: "Ir al trabajo")  
        .putExtra(AlarmClock.EXTRA_HOUR, value: 11)  
        .putExtra(AlarmClock.EXTRA_MINUTES, value: 15);  
    startActivity(intent);  
}
```

```
}
```

- Recordad que si no reconoce los métodos, tendremos que importar la librería que necesitamos.

Intent implícitas VI

- ▶ Seguramente al pulsar el botón no aparezca nada. Eso es debido a que nos falta aún una línea de código que añadir en el AndroidManifest.
- ▶ Este archivo nos quedaría así:

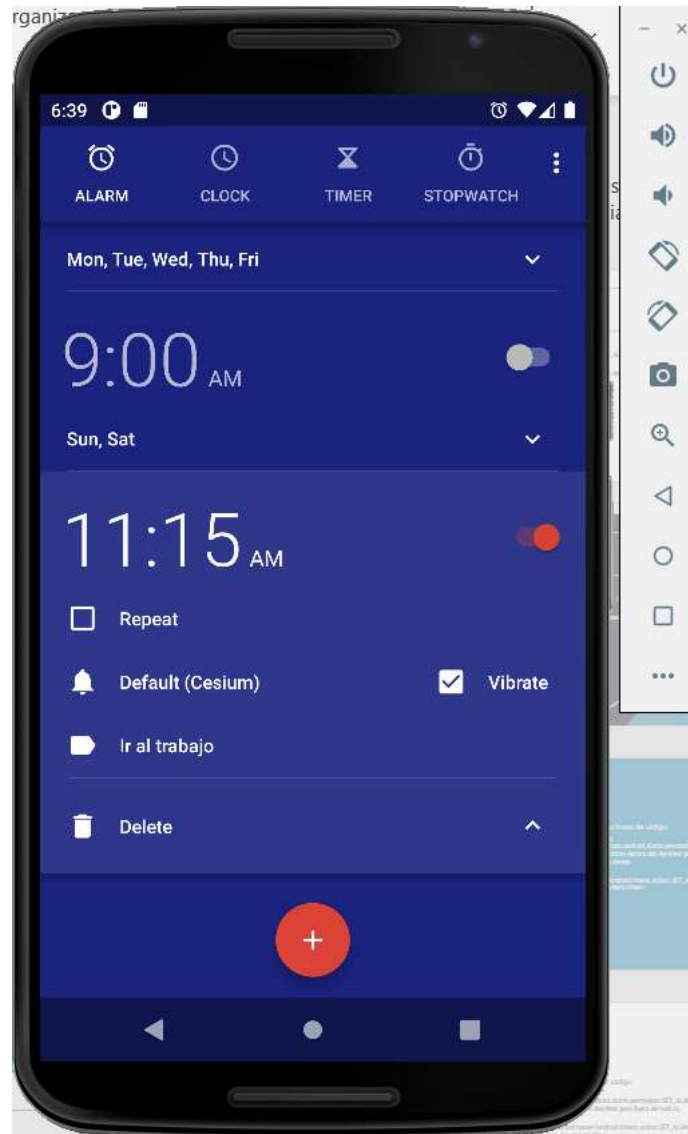
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.miprimeraalarma">
    <uses-permission android:name="com.android.alarm.permission.SET_ALARM" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="MiPrimeraAlarma"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MiPrimeraAlarma">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```


Intent implícitas VII

- ▶ Hemos añadido dos líneas de código:
 - ▶ `<uses-permission android:name="com.android.alarm.permission.SET_ALARM" />` al inicio dentro del Manifest pero fuera de todo lo demás
 - ▶ `<action android:name="android.intent.action.SET_ALARM" />` dentro del `<intent-filter>`
- ▶ Una vez puestas estas dos líneas de código el resultado debería ser que cuando se pulsa el botón que hemos creado, aparece la central de alarmas del dispositivo, con nuestra alarma creada.

Intent implícitas VIII



Intent implícitas IX. Ejemplo para realizar en clase

VAMOS A IMPLEMENTAR UNA APLICACIÓN PARA PODER REALIZAR UNA LLAMADA DE TELÉFONO.

VAMOS A TENER DOS OPCIONES ACTION_DIAL Y ACTION_CALL.

¿CUAL ES LA DIFERENCIA ENTRE LAS DOS OPCIONES?

HAZ UN EJEMPLO UTILIZANDO ACTION_DIAL

¿QUE ES UN FORMATO URI?

CUANDO TENGAIS HECHO EL EJEMPLO AVISADME PARA PODER VERLO

Intent implícitas X

- ▶ La otra opción de uso de las Intent implícitas es ser la aplicación que de respuesta a la solicitud de la Intent.
- ▶ Es decir y siguiendo con nuestro primer ejemplo, que nuestra aplicación sea una aplicación de gestión de alarmas.
- ▶ Para ello creamos un nuevo Activity que le vamos a llamar `gestionAlarmasActivity` de tipo empty.
- ▶ Para comprobar como funciona, en este activity que hemos creado, solo vamos a poner Soy el gestor de alarmas.
- ▶ Esta será nuestra aplicación, aunque como vemos no está creada como tal.

Intent implícitas XI

- ▶ Para que nuestra aplicación aparezca el S.O. como una aplicación que da respuesta a la creación de alarmas, vamos a tener que modificar nuestro AndroidManifest.
- ▶ En este archivo añadiremos un `<intent-filter>` que le indicará al S.O. que nosotros podemos resolver esa acción.
- ▶ El fichero quedaría así:

```
<activity
    android:name=".GestionAlarmasActivity"
    android:exported="true" >
    <intent-filter>
        <action android:name="android.intent.action.SET_ALARM" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Intent implícitas XI

- ▶ Una vez realizados todos estos pasos y si damos a ejecutar la práctica, nos aparecerán dos opciones para gestionar las alarmas: una la del propio S.O. y otra la que acabamos de crear nosotros.
- ▶ El resultado sería:

