
	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD COMPETENCIA					

Tema 4

Más sobre IU

Índice

1.	Introducción	1
2.	Orientación.....	1
2.1.	Usar un único layout.....	1
2.2.	Usar varios Layouts	3
2.3.	Evitar cambio de layout	4
2.4.	Control de orientación.....	4
2.5.	Mantener datos en cambios de orientación.....	5
3.	Fragments.....	6
3.1.	Creación de los fragments	7
3.2.	Asociar los fragments con las activities que los usan.....	9
4.	Dialogs.....	12
4.1.	AlertDialog	12
4.2.	DatePickerDialog	14
4.3.	TimePickerDialog.....	15
5.	CalendarView.....	16
6.	DatePicker	16
7.	TimePicker	17
8.	Navegación por pestañas	19
8.1.	Layout_scrollFlags:	19
8.2.	Definición de las pestañas	20
9.	Guías de interés	27
10.	¿Y ahora por dónde sigo?	27

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

1. Introducción

Este tema es continuación directa del anterior y en el veremos más aspectos sobre el uso las *Activities* y sus componentes.

2. Orientación¹

Como ya hemos comentado en el tema 2, en el momento que la orientación del dispositivo cambia la *Activity* se destruye y se vuelve a crear lanzando de nuevo *onCreate*.

Esto es debido a que la distribución del espacio y de los componentes es diferente dependiendo de la orientación: de si están en modo vertical (*Portrait*) u horizontal (*Landscape*).

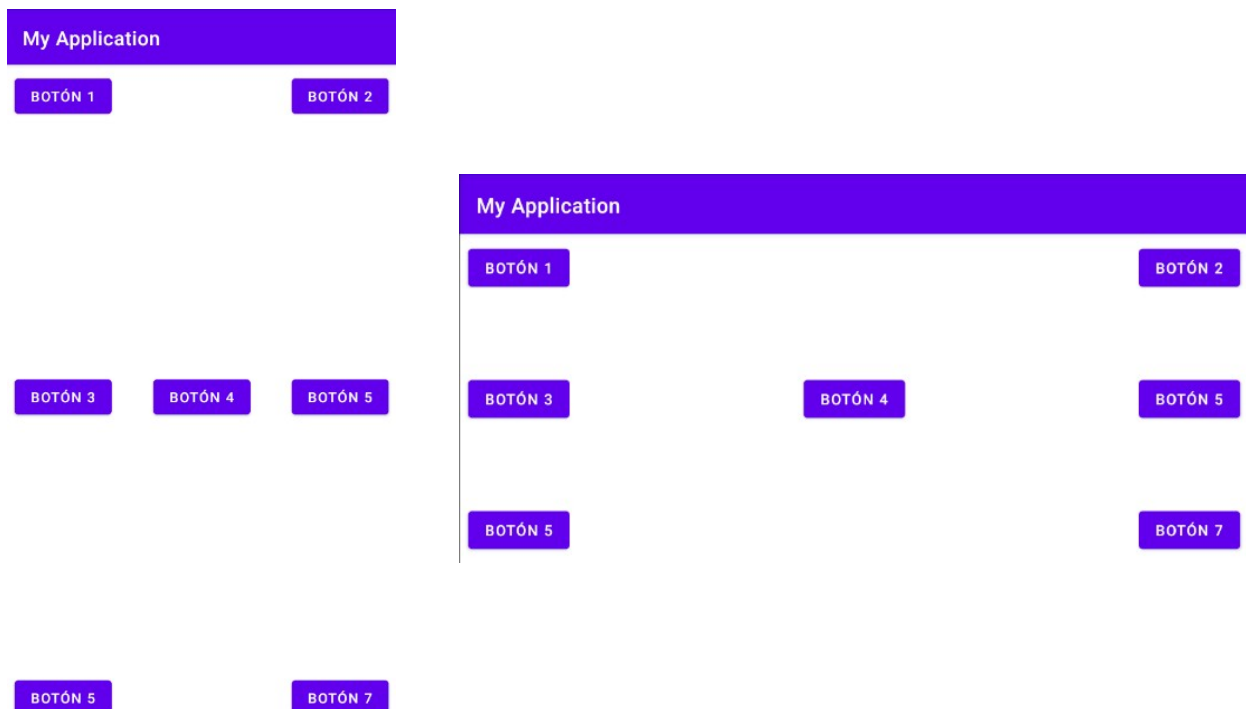
Para gestionar el cambio de orientación hay, entre otras, las siguientes tres opciones:

- Anclar los componentes a distintos elementos del *layout* (principalmente laterales) de forma que al cambiar se adapten al nuevo esquema.
- Usar un *layout* distinto para cada orientación. En este caso se define en la carpeta *layout-land* un *XML* para el posicionamiento horizontal.
- También existe la posibilidad de forzar una posición determinada y que no haya cambios de *layout*.


Veamos ejemplos de las distintas posibilidades.

2.1. Usar un único layout

En el ejemplo siguiente cada uno de los botones está posicionado en relación al contenedor padre. Por lo que se puede observar que al rotar el dispositivo la disposición de los elementos mantiene su colocación.




¹ Más información en: https://developer.android.com/guide/practices/screens_support.html?hl=es

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Un ejemplo de un *layout* de este tipo podría ser:

<pre> <?xml version="1.0" encoding="utf-8"?> <androidx.constraintlayout.widget.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=".MainActivity" tools:layout_editor_absoluteY="81dp"> <Button android:id="@+id/button1" android:layout_width="wrap_content" android:layout_height="wrap_content" android:layout_marginStart="8dp" android:layout_marginTop="8dp" android:text="Botón 1" app:layout_constraintStart_toStartOf="parent" app:layout_constraintTop_toTopOf="parent" /> <Button android:id="@+id/button2" android:layout_width="wrap_content" android:layout_height="wrap_content" android:layout_marginTop="8dp" android:layout_marginEnd="8dp" android:text="Botón 2" app:layout_constraintEnd_toEndOf="parent" app:layout_constraintTop_toTopOf="parent" /> <Button android:id="@+id/button3" android:layout_width="wrap_content" android:layout_height="wrap_content" android:layout_marginStart="8dp" android:layout_marginTop="8dp" android:layout_marginBottom="8dp" android:text="Botón 3" app:layout_constraintBottom_toBottomOf="parent" app:layout_constraintStart_toStartOf="parent" app:layout_constraintTop_toTopOf="parent" /> </pre>	<pre> <Button android:id="@+id/button4" android:layout_width="wrap_content" android:layout_height="wrap_content" android:layout_marginStart="8dp" android:layout_marginTop="8dp" android:layout_marginEnd="8dp" android:layout_marginBottom="8dp" android:text="Botón 4" app:layout_constraintBottom_toBottomOf="parent" app:layout_constraintEnd_toEndOf="parent" app:layout_constraintStart_toStartOf="parent" app:layout_constraintTop_toTopOf="parent" /> <Button android:id="@+id/button5" android:layout_width="wrap_content" android:layout_height="48dp" android:layout_marginTop="8dp" android:layout_marginEnd="8dp" android:layout_marginBottom="8dp" android:text="Botón 5" app:layout_constraintBottom_toBottomOf="parent" app:layout_constraintEnd_toEndOf="parent" app:layout_constraintTop_toTopOf="parent" /> <Button android:id="@+id/button6" android:layout_width="wrap_content" android:layout_height="wrap_content" android:layout_marginStart="8dp" android:layout_marginBottom="8dp" android:text="Botón 6" app:layout_constraintBottom_toBottomOf="parent" app:layout_constraintStart_toStartOf="parent" /> <Button android:id="@+id/button7" android:layout_width="wrap_content" android:layout_height="wrap_content" android:layout_marginEnd="8dp" android:layout_marginBottom="8dp" android:text="Botón 7" app:layout_constraintBottom_toBottomOf="parent" app:layout_constraintEnd_toEndOf="parent" /> </androidx.constraintlayout.widget.ConstraintLayout> </pre>
---	---

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

2.2. Usar varios Layouts

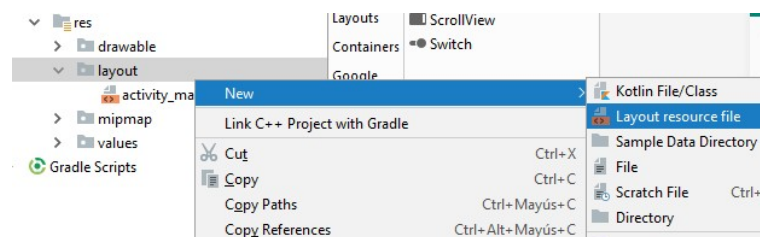
Cuando se necesitan reposicionar los componentes, cambiar su tamaño de una forma más compleja o incluso cambiar de forma completa los componentes del *layout* esta es la alternativa adecuada.

Podemos crear, por ejemplo, *layouts* diferentes según la orientación del dispositivo, según el idioma, el tamaño, ...

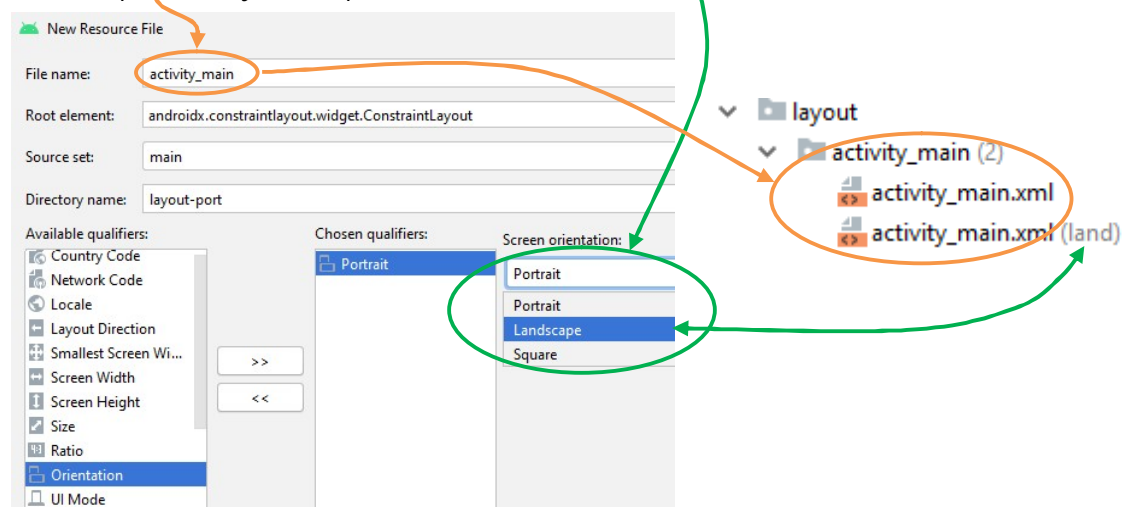
Vamos como crear un *layout* para cada orientación específica. Se utilizara *layout-land* para *landscape* (pantalla con orientación horizontal), *layout-port* para *portrait* (pantalla con orientación vertical). Si no se define un *layout* para una orientación Android cargara el *layout* por defecto. Estos *layouts* deben tener el mismo nombre diferenciándose por el sufijo.

Se pueden crear de las dos maneras siguientes:

- Forma genérica (permite crear *layouts* para distintas variaciones):



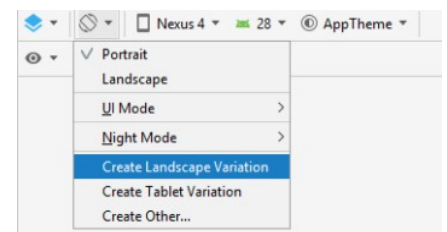
Para un mismo nombre de *layout* se puede escoger el tipo de variación que se desea (el nombre del *layout* debe ser el mismo que el del *layout* del que se desea crear la variación):



Una vez creado el nuevo *layout* se puede apreciar que en el directorio de recursos de *layouts* aparecen dos *layouts* para *activity_main*. El *layout* por defecto y el *layout* que cargará, de forma automática, cuando la disposición sea horizontal: es el que tiene el sufijo *land* (de *Landscape*).


- En el editor de *layout*: el botón que permite cambiar la orientación tenemos disponible la opción *Create Landscape Variation* que ofrece la opción de crear, de forma directa, una variación horizontal o incluso mediante la opción *Create other* crear otro tipo de variación.

Esto permite mucha versatilidad porque incluso se le pueden asignar eventos distintos a los mismos componentes según su *layout*.



Tarea 1

Crea un *layout* con dos orientaciones y añádele a cada uno de ellas un botón. En este botón, y mediante la propiedad *onClick*, muestren al pulsarlos un *toast* con la orientación de la pantalla.

	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

2.3. Evitar cambio de layout

Si lo que se quiere es forzar a una única orientación (por ejemplo horizontal), tenemos las opciones siguientes:

- En el fichero *AndroidManifest* añadiendo al elemento *activity* la propiedad `screenOrientation`:

```
<activity .... android:screenOrientation="landscape">
```

- Mediante código en el método *onCreate* mediante la instrucción:

```
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
```

Las principales alternativas son:

CONSTANTE	DESCRIPCIÓN	
SCREEN_ORIENTATION_FULL_SENSOR	Se orienta según el sensor de orientación	
SCREEN_ORIENTATION_LANDSCAPE	Siempre en horizontal ignorando el sensor de orientación	
SCREEN_ORIENTATION_PORTRAIT	Siempre en vertical ignorando el sensor de orientación	
SCREEN_ORIENTATION_USER_LANDSCAPE	Siempre en horizontal	Si el sensor de orientación está habilitado, este decide hacia qué lado se orienta.
SCREEN_ORIENTATION_USER_PORTRAIT	Siempre en vertical	

Podemos consultar todas las posibilidades disponibles tanto para el fichero XML como mediante código en:

<https://developer.android.com/reference/android/R.attr.html#screenOrientation>

2.4. Control de orientación

Si se necesita saber la orientación del dispositivo se pueden usar uno de los métodos siguientes:

- Obteniendo su orientación:


```
if (getResources().getConfiguration().orientation == Configuration.ORIENTATION_LANDSCAPE) { // Horizontal
    Toast.makeText(this, "Horizontal", Toast.LENGTH_SHORT).show();
} else { // Vertical
    Toast.makeText(this, "Vertical", Toast.LENGTH_SHORT).show();
}
```

- En base al tamaño de la pantalla: de esta manera se puede, además, obtener el tamaño de la pantalla.

```
Display display = getWindowManager().getDefaultDisplay();
Point p = new Point();
display.getSize(p); // tamaño usable de la pantalla (no incluye ciertos elementos como puede ser la statusBar)
if (p.x > p.y) { // Apaisado p.x → tamaño de la pantalla en el eje X p.y → tamaño de la pantalla en el eje y
    Toast.makeText(this, "Orientación Horizontal -> " + p.x + ":" + p.y, Toast.LENGTH_SHORT).show();
} else { // Vertical
    Toast.makeText(this, "Orientación Vertical -> " + p.x + ":" + p.y, Toast.LENGTH_SHORT).show();
}
display.getRealSize(p); // tamaño real de la pantalla
Toast.makeText(this, "Tamaño real -> " + p.x + ":" + p.y, Toast.LENGTH_SHORT).show();
```

Una opción equivalente a la anterior sería:

```
DisplayMetrics dm = new DisplayMetrics();
WindowManager windowManager = (WindowManager) getSystemService(Context.WINDOW_SERVICE);
windowManager.getDefaultDisplay().getMetrics(dm); // métricas de la pantalla con tamaño reducido
if (dm.widthPixels > dm.heightPixels) { // Apaisado
    Toast.makeText(this, "Orient. Horizontal -> " + dm.widthPixels + ":" + dm.heightPixels, Toast.LENGTH_SHORT).show();
} else { // Vertical
    Toast.makeText(this, "Orient. Vertical -> " + dm.widthPixels + ":" + dm.heightPixels, Toast.LENGTH_SHORT).show();
}
windowManager.getDefaultDisplay().getRealMetrics(dm); // métricas de la pantalla con tamaño real
Toast.makeText(this, "Tamaño real -> " + dm.widthPixels + ":" + dm.heightPixels, Toast.LENGTH_SHORT).show();
Log.i("Info", "Densidad de la pantalla en dpi: " + dm.densityDpi);
```

	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

2.5. Mantener datos en cambios de orientación

Cuando se cambia la orientación de una *activity* esta se destruye y se construye una nueva para la nueva orientación. Esto también ocurre en general ante un cambio de configuración del sistema (un cambio en el idioma, por ejemplo). Si por algún motivo queremos asegurarnos que alguna información presente en la actividad se pase a la nueva se pueden usar los métodos *callback*:

- *onSaveInstanceState*: almacena información en un objeto tipo *Bundle* antes de pausar el *activity* (es el mismo objeto que se le pasa a *onCreate* pero cuando se lanza la aplicación tiene un valor null).
- *onRestoreInstanceState*: permite recuperar la información almacenada. Se llama justo después de *onStart*.

Estos métodos solo son llamados cuando hay cambios de configuración y no cuando se pulsa el botón *Back* u otras situaciones similares.

Hay que tener en cuenta que si un componente (*EditText*, *Button*, ...) tiene un *id* único *Android* usa ese *id* para mantener, de forma automática, su estado ante cambios de orientación, lenguaje, ...

Un ejemplo de su utilización es:

```
@Override
public void onSaveInstanceState(Bundle outState) {
    // Guardamos un dato de tipo String
    outState.putString("DATO", "Dato a guardar");
    super.onSaveInstanceState(outState);
}

@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    // Recuperamos el dato, de tipo String, que habíamos guardado
    String dato = savedInstanceState.getString("DATO");
}
```


Tarea 2

Inserta dos *editText*, un *textView* y un *seekBar*.

Prueba a rotar el dispositivo varias veces.

Los valores de los dos primeros *editText* y del *seekbar* deben permanezcan al cambiar de orientación del dispositivo (pruébalo configurando ambos componentes tanto con *id* como sin *id*).

En el *textView* se deben ir concatenando el contenido de los *editText*. Por ejemplo el contenido del *textView* será el acumulativo del contenido los *editText* antes de cada una de las rotaciones.

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

3. Fragments²

Los *fragments* surgen a partir de la versión 3.0 de *Android* como una solución ante la aparición de dispositivos con una pantalla de gran tamaño, como pueden ser las *tablets*. Una interfaz gráfica diseñada para un móvil no tiene por que adaptarse bien a una *tablet*, varias pulgadas más grande.

Se podría decir que un *fragment* es un fragmento, una parte, ... de interfaz que puede usarse en más de una *Activity* de la aplicación y que dispone de su propio ciclo de vida. Un *fragment* no existe por si solo sino que debe estar siempre integrado en una *activity* y estas pueden contener más de un *fragment*.

Aunque un *fragment* tiene un ciclo de vida propio este se ve afectado por el ciclo de vida de la actividad contenedora. Por ejemplo si esta se pausa todos los *fragment* que contenga también se pausaran.

Los métodos principales del ciclo de vida de un *fragment* son:

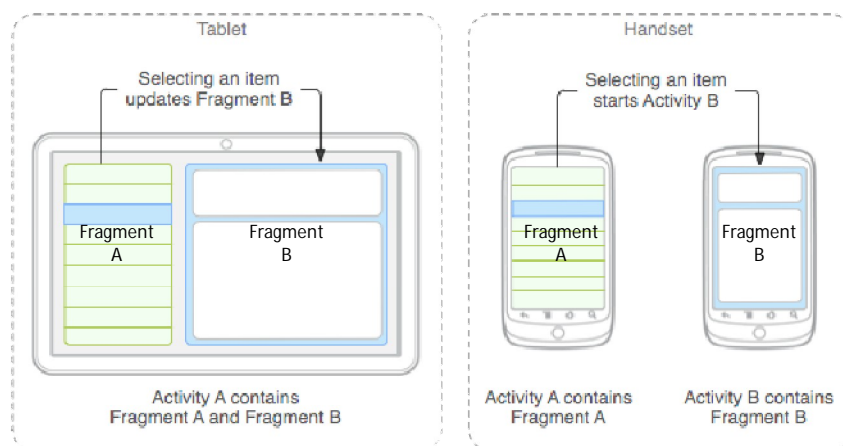
- *onCreate*: al igual que con una *activity* este método es invocado por *Android* cuanto se crea el *fragment*. Es aquí donde se deben inicializar los componentes, del *fragment*, que se vayan usar.
- *onCreateView*: es invocado cuando se desea inicializar la interfaz. Se debe retornar un *View* con la interfaz del *fragment* o *null* si no posee interfaz.
- *onPause*: es invocado cuando el *fragment* se pausa, es decir no se puede interaccionar con él. Este es el lugar para almacenar los datos que se desean conservar entre ejecuciones del *fragment*.

Los *fragments* al igual que las *activities* están compuestos de un fichero *layout XML*, que contendrá la interfaz de ese *fragment*, y una clase java, la cual contendrá la parte de lógica del *fragment*.


Los *fragments* permiten mucha más modularidad y versatilidad a la hora de diseñar interfaces gráficos. Como ejemplo podemos tomar el de la documentación de *Android*².

El mismo resultado obtenido usando *fragments* se podrían obtener definiendo distintos *layouts* para cada configuración de pantalla. Pero además de ser mucho más engorroso se repetiría gran cantidad de código.

Aquí se ve un caso claro en el que dependiendo de la situación puede ser más interesante usar una *activity* con dos *fragment* que se visualizan de forma simultánea (la *tablet* de la imagen siguiente) o dos *activities*, cada una con un único *fragment* (el móvil de la imagen).



² <https://developer.android.com/guide/components/fragments>

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

A modo de ejemplo vamos a desarrollar una aplicación que posea dos *fragments*:

- Fragment A: deberá mostrar tres botones
- Fragment B: tendrá una etiqueta de texto que indicará cual ha sido el último botón pulsado.

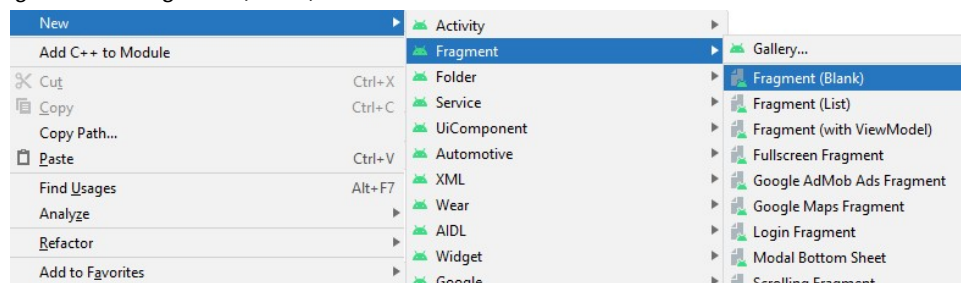
Dependiendo de la orientación del dispositivo se mostrara:

- Posición horizontal: se mostraran de forma simultánea los dos *fragments*, el *fragment A* a la izquierda y el *fragment B* a la derecha ocupando cada uno de ellos la mitad de la pantalla.
- Posición vertical: Comenzara mostrando el *fragment A* y al pulsar un botón cambiara a una nueva *activity* que mostrara *fragment B*.

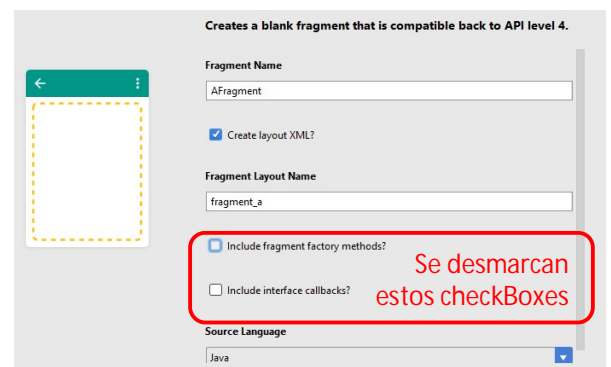
3.1. Creación de los fragments

Para crear los *fragments* creamos un proyecto con el paquete `com.example.appfragments` donde se van a seguir los siguientes pasos:

- Crear los *fragments*: creamos dos *fragments* denominados *AFragment* y *BFragment* mediante el asistente: *New* → *fragment* → *fragment (blank)*.



En el asistente se debe rellenar el nombre del *fragment* y se puede modificar tanto el nombre del *layout* sugerido como el lenguaje en que se va a desarrollar el *fragment*. En el caso de estar presentes, desmarcaremos las dos opciones disponibles: *Include fragment factory methods* e *include interface callbacks*. Una vez creados los dos *fragments* se generará las clases *AFragment* y *BFragment* y sus *layouts* asociados: *fragment_a.xml* y *fragment_b.xml*.




En el método *onCreateView* de las clases creadas se le asigna el *layout* al *fragment*. El *container* es el *view* de la actividad anfitriona y es donde se establece el *fragment*, no lo usaremos.

Si quieres ver más sobre el ciclo de vida de un *fragment* puedes leer el siguiente enlace:

<https://developer.android.com/guide/components/fragments.html?hl=es>

- Como cada *fragment* tiene un *layout* asociado. Se deberá modificar su *layout* con los componentes deseados. En nuestro ejemplo el *fragment AFragment* tendrá como *layout* asociado el fichero XML *fragment_a.xml* al que deberemos insertarle tres botones.

Por defecto un *fragment* tiene como *layout* un *FrameLayout* que ocupara toda la actividad. Se recomienda, si se desea usar más de un componente, añadirle un *ConstraintLayout* y colocar dentro de él los componentes.

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Con *fragments* se recomienda usar como contendor principal un *FrameLayout* para evitar posibles problemas cuando se realiza la sustitución y transición entre *fragments*.

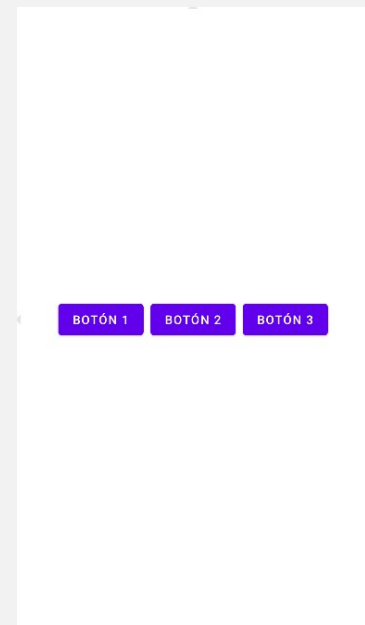
```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/id_fragment_a_botones"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".AFragment">


    <androidx.constraintlayout.widget.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:layout_editor_absoluteY="81dp">

        <Button
            android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginEnd="8dp"
            android:text="Botón 1"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toStartOf="@+id/button2"
            app:layout_constraintHorizontal_bias="0.5"
            app:layout_constraintHorizontal_chainStyle="packed"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <Button
            android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Botón 2"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toStartOf="@+id/button3"
            app:layout_constraintHorizontal_bias="0.5"
            app:layout_constraintStart_toEndOf="@+id/button"
            app:layout_constraintTop_toTopOf="parent" />

        <Button
            android:id="@+id/button3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:text="Botón 3"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.5"
            app:layout_constraintStart_toEndOf="@+id/button2"
            app:layout_constraintTop_toTopOf="parent" />
    </androidx.constraintlayout.widget.ConstraintLayout>
</FrameLayout>
```

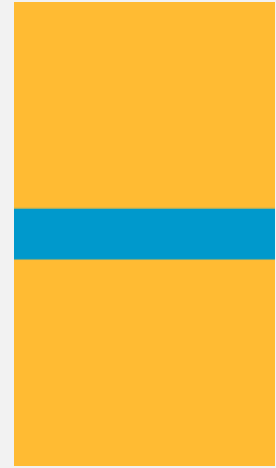


	RAMA:	Informática	CICLO:	Desenvolvememto de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

- Así mismo creamos el *fragment BFragment* y modificamos su fichero de *layout*, *fragment_b.xml* insertándole un *textView*:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_orange_light"
    tools:context=".BFragment">

    <TextView
        android:id="@+id/textFragmentB"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical|center"
        android:background="@android:color/holo_blue_dark"
        android:textSize="60sp" />
</FrameLayout>
```



- Modificamos la clase *java BFragment* generada por el asistente obteniendo el *TextView* del *Layout* añadiéndole un atributo de instancia *txtB* de tipo *TextView* y cambiando el método *onCreateView* el siguiente:

```
public TextView txtB;

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    // Inflate the layout for this fragment
    View view=inflater.inflate(R.layout.fragment_b, container, false);
    this.txtB=view.findViewById(R.id.textFragmentB);
    return view;
}
```

Este método permite obtener e iniciar el componente *TextView* que es dónde se colocará el resultado. Se ejecuta una vez que la *activity*, donde está el *fragment*, ha terminado de crearse y ya existen los componentes.

3.2. Asociar los Fragments con las activities que los usan

Una vez definidos los *fragments* deberemos asociar estos con las *activities* que los van usar. Esta asociación se puede definir de forma estática en el XML (mediante el componente *FragmentContainerView*) o se puede crear de forma dinámica mediante código. Nosotros veremos el primer caso.

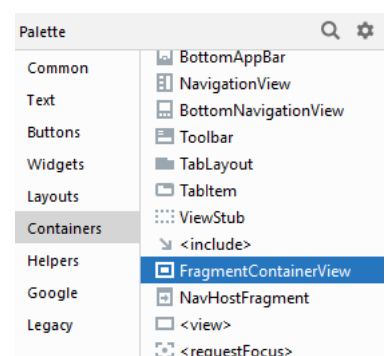
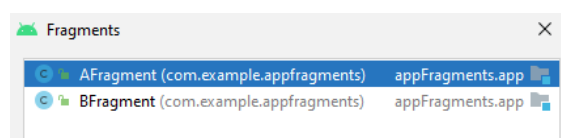
Recordemos la distribución de los *fragments*:


- Si el dispositivo está en posición horizontal en *MainActivity* estarán los dos *fragments* de forma simultánea.
- Si el dispositivo está en posición vertical se la aplicación muestra *fragmentA* y al pulsar un botón, de los presentes en el *fragmentA*, cambiara a una nueva *activity* que mostrara el *fragmentB*.

Realizaremos los siguientes pasos:

- Arrastramos, desde la paleta de componentes, en la sección *Containers*, el componente *<FragmentContainerView>* al *layout* vertical de *MainActivity*.

Una vez arrastrado al *layout* Android Studio pide especificar el *fragment* que se quiere insertar. En nuestro caso, para empezar, insertaremos *AFragment*.



	RAMA:	Informática	CICLO:	Desenvolvememto de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Se deberá especificar que el *fragment* ocupe todo el *layout* mediante los *constraints* necesarios para ello.

En estos momento aunque ya se ha añadido el fragment, y con él su *Layout* asociado, a la actividad este se ve en blanco. Para que se pueda visualiza en el editor de layouts hay que indicarle el fragment que se quiera visualizar en la atributo *Layout*.

Si no aparece ese atributo se puede añadir directamente en el fichero XML donde se define el fragment:

```
tools:layout="@layout/fragment_a".
```

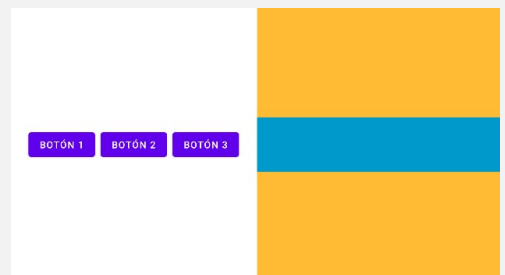
Por tanto el *layout* de la actividad principal, *main_layout.xml*, en posición vertical (*portrait*) debe ser similar al siguiente ejemplo:


```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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=".MainActivity">
    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/prin_layoutvertical_fragmentA"
        android:name="com.example.appfragments.AFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:layout="@layout/fragment_a" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Substituir por el valor adecuado. Mientras que la última parte es el fragment a añadir las partes anteriores representan el paquete del proyecto.

- De forma análoga, en el *layout activity_main.xml* para la orientación horizontal (*Landscape*) se añaden dos *fragments* ocupando cada uno una mitad. Al primer *fragment*, situado en la mitad izquierda, se le asocia el *fragment AFragment* mientras que al segundo el *fragment BFragment*.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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">
    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/prin_layouthorizontal_fragmentA"
        android:name="com.example.appfragments.AFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintWidth_percent="0.5"
        tools:layout="@layout/fragment_a" />
    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/prin_layouthorizontal_fragmentB"
        android:name="com.example.appfragments.BFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginBottom="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />
```



	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

```

app:layout_constraintHorizontal_bias="1.0"
app:layout_constraintStart_toEndOf="@+id/prin_layouthorizontal_fragmentA"
app:layout_constraintTop_toTopOf="parent"
tools:layout="@layout/fragment_b" />
</androidx.constraintlayout.widget.ConstraintLayout>

```

- Se define una nueva *activity* secundaria que mostrara el *fragment BFragment* cuando el dispositivo este en un modo vertical y se pulse un botón. Para ello se debe añadir, al archivo AndroidManifest.xml, la siguiente línea en la definición de la actividad: `android:screenOrientation="portrait"`

El código del *layout* de esta actividad será:

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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=".Secundaria">
    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/sec_layoutvertical_fragmentB"
        android:name="com.example.appfragments.BFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:layout="@layout/fragment_b" />
</androidx.constraintlayout.widget.ConstraintLayout>

```

Nombre de la actividad secundaria

Substituir por el valor adecuado. Mientras que la última parte es el fragment a añadir las partes anteriores representan el paquete del proyecto.

- En el método *onResume* de la *activity* secundaria añade el siguiente código: se recoge el nombre del botón, que se ha pasado desde la actividad principal, en un *intent*, y lo visualizara en el *textview* del *FragmentB*. Tenemos las posibilidades:

```

// Posibilidad 1: Se obtiene el TextView situado en el FragmentB
Fragment fB= getSupportFragmentManager().findFragmentById(R.id.sec_layoutvertical_fragmentB);
TextView txtB1=((BFragment)fB).getView().findViewById(R.id.textFragmentB);
txtB1.setText(getIntent().getStringExtra("nombreBoton"));

// Posibilidad 2: Como FragmentB se visualiza en la activity se puede acceder al TextView desde el Layout de la activity
final TextView txtB2 = findViewById(R.id.textFragmentB);
txtB2.setText(getIntent().getStringExtra("nombreBoton"));

```


- Se establece, en los botones del *AFragment*, las misma propiedad *onclick* para todos los botones: `android:onClick="clickBoton"`. El método *clickBoton* se debe implementar en el *MainActivity*.

```

public void clickBoton(View v){
    Button b=(Button)v;
    // Si el dispositivo esta en horizontal los dos fragments se visualizan a la vez por lo que se puede obtener BFragment
    if (getResources().getConfiguration().orientation == Configuration.ORIENTATION_LANDSCAPE){
        Fragment fB= getSupportFragmentManager().findFragmentById(R.id.prin_layouthorizontal_fragmentB);
        ((BFragment)fB).txtB.setText(b.getText().toString()); // Y le asignamos el texto
    } else { //Si esta en vertical lanzamos el activity secundario ya que es ahí donde esta BFragment
        Intent intent=new Intent(this,Secundaria.class);
        intent.putExtra("nombreBoton", b.getText().toString());
        startActivity(intent);
    }
}

```

Si al especificar el método *clickBoton* en la propiedad *onClick* en los botones de *Afragment* da error este se puede evitar creando el método *clickBoton* sin contenido en el *fragment*.

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

4. Dialogs

Los diálogos son elementos que pausan la *activity* actual y aparecen en primer plano sobre la *activity* actual solicitando una acción al usuario.

Android dispone de varios diálogos predefinidos y además permite su creación a medida.

La clase base para los diálogos es *Dialog*³ pero *Google* no recomienda instanciarla de forma directa y, en vez de ello, utilizar una de las siguientes subclases:

- *AlertDialog*: Puede mostrar un título, hasta tres botones, una lista de elementos seleccionables o un *layout* a medida.
- *DatePickerDialog* y *TimePickerDialog*: Son diálogos para obtener la fecha y la hora.
- *ProgressDialog*: Diálogo de muestra una barra de progreso y un texto opcional o una vista. *Google* tampoco recomienda su uso en favor de usar un *ProgressBar* en el *layout*.

<http://developer.android.com/intl/es/reference/android/widget/ProgressBar.html>

Aunque estos diálogos se pueden usar de forma directa *Google* recomienda, para un uso más versátil, crear nuevos diálogos heredando de *DialogFragment* ya que de esta forma, además del propio diálogo, disponemos de la versatilidad y del ciclo de vida de los *fragments* con lo que podemos manejar mas situaciones: giros, reinicios, ...

4.1. AlertDialog

Diálogo que puede mostrar los siguientes tres elementos:

- Un título.
- Hasta tres botones.
- Una lista o un *layout* personalizado.

Existen tres tipos de botones que se asocian con un tipo de respuesta. Dentro del mismo diálogo solo puede haber uno de cada tipo. Los tipos pueden ser:

- Respuesta positiva: se acepta y continúa la acción.
- Respuesta negativa: se cancela la acción.
- Respuesta neutra: si no se quiere proceder con la acción pero tampoco cancelarla. Por ejemplo una acción tipo "Recuérdemelo después".

Veamos en un ejemplo de su uso:

- Lo primero es crear las cadenas en el fichero *strings.xml* que usaremos:

```
<string name="app_name">Dialogos</string>
<string name="titulo">Escoge tu pastilla</string>
<string name="mensaje">¿Quiere tomar las pastilla roja?</string>
<string name="boton_positivo">¡Adelante!</string>
<string name="boton_negativo">Nooo, paso</string>
<string name="boton_neutro">Depende</string>
<string name="respuesta_ok">Bienvenido al mundo real</string>
<string name="respuesta_negativa">Tu te lo pierdes</string>
<string name="respuesta_neutra">Lo decido luego</string>
```


- Implementaremos, en *MainActivity*, los métodos que son invocados al pulsar el botón correspondiente:

```
public void opcionOk(String s) {
    Toast.makeText(this, s, Toast.LENGTH_SHORT).show();
}

public void opcionCancel(String s) {
    Toast.makeText(this, s, Toast.LENGTH_SHORT).show();
}

public void opcionNeutra(String s) {
    Toast.makeText(this, s, Toast.LENGTH_SHORT).show();
}
```

³ <https://developer.android.com/guide/topics/ui/dialogs.html>

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

- Crearemos ahora una clase que denominaremos *DialogoAlerta* que hereda de *DialogFragment* en la cual se va a crear el cuadro de diálogo. El código de dicha clase puede ser similar al siguiente:

```
public class DialogoAlerta extends DialogFragment { // Hereda de DialogFragment
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) { // Función llamada por el sistema al crear el diálogo
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity()); // Se encarga de configurar el diálogo.
        builder.setMessage(R.string.mensaje); // Especificamos el mensaje del diálogo
        builder.setTitle(R.string.titulo); // Y el título del diálogo

        // Se añade el botón de respuesta positiva con su evento
        builder.setPositiveButton(R.string.boton_positivo, new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) { // Click en el botón positivo.
                ((MainActivity)getActivity()).opcionOk(getString(R.string.respuesta_ok));
            }
        });

        // Se añade el botón de respuesta negativa
        builder.setNegativeButton(R.string.boton_negativo, new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) { // Click en el botón negativo
                ((MainActivity)getActivity()).opcionCancel(getString(R.string.respuesta_negativa));
            }
        });

        // Se añade el botón de respuesta neutra
        builder.setNeutralButton(R.string.boton_neutro, new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) { // Click en el botón neutro
                ((MainActivity)getActivity()).opcionNeutra(getString(R.string.respuesta_neutra));
            }
        });


        return builder.create(); // Se devuelve el diálogo al sistema para que lo muestre
    }
}
```

- En *MainActivity* crearemos y mostraremos el diálogo al pulsar un botón:

```
Button boton=findViewById(R.id.boton);
boton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        DialogoAlerta dAlerta = new DialogoAlerta(); // Se crea el diálogo y se muestra
        dAlerta.show(getSupportFragmentManager(), "Alerta"); // "Alerta" es un TAG único
    }
});
```

Además de usar una clase para definir nuestro dialogo se puede realizar también desde cualquier zona del código. Por ejemplo como respuesta a la pulsación del segundo botón definido anteriormente:

```
Button boton2=(Button)findViewById(R.id.boton2);
boton2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
        builder.setMessage(R.string.mensaje); // Especificamos el mensaje del diálogo
        builder.setTitle(R.string.titulo); // Y el título del diálogo
        builder.setPositiveButton(R.string.boton_positivo, new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) { // Click en el botón positivo.
                Toast.makeText(MainActivity.this, "Botón ok", Toast.LENGTH_SHORT).show();
            }
        });
        builder.show();
    }
});
```


	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

```
    }
    });
```

Otras posibilidades del *AlertDialog* son:

- Crear una lista de selección de elementos.
- Crear un *layout* a medida
- Enviar de vuelta la respuesta a la *activity* llamante.

Para ver como se realizan estas posibilidades se puede ver la documentación:

<https://developer.android.com/guide/topics/ui/dialogs.html#DialogFragment>

4.2. DatePickerDialog

Este diálogo se utiliza para obtención de una fecha. Su uso es muy similar al caso anterior.

Veamos un ejemplo de su utilización:

- Creamos, en *MainActivity*, el siguiente método que se ejecutará al aceptar una fecha seleccionada.

```
public void fecha(String s) {
    Toast.makeText(this, s, Toast.LENGTH_SHORT).show();
}
```

- Se crea la clase *DialogoFecha* que hereda de *DialogFragment* e implementa el listener *OnDateSetListener*:

```
public class DialogoFecha extends DialogFragment implements DatePickerDialog.OnDateSetListener {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Cogemos la fecha actual para inicializar el calendario. Se puede usar un constructor para inicializarlo con otra fecha
        final Calendar c = Calendar.getInstance();
        int anio = c.get(Calendar.YEAR);
        int mes = c.get(Calendar.MONTH);
        int dia = c.get(Calendar.DAY_OF_MONTH);
        // Crea el diálogo y lo devuelve
        return new DatePickerDialog(getActivity(), this, anio, mes, dia);
    }
    public void onDateSet(DatePicker view, int year, int monthOfYear, int dayOfMonth) {
        // Al pulsar el botón se muestra un mensaje con la fecha
        Toast.makeText(view.getContext(), dayOfMonth+"/"+(monthOfYear+1)+"/"+year,
            Toast.LENGTH_SHORT).show();
        // O también se puede ejecutar un método que se encuentra en otra activity
        ((MainActivity) getActivity()).fecha( dayOfMonth+"/"+(monthOfYear+1)+"/"+year); // Los meses van de 0 a 11
    }
}
```


- El tercer botón creado, en *MainActivity*, se utilizara para lanzar el *DatePickerDialog*:

```
Button boton3=(Button)findViewById(R.id.boton3);
boton3.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        DialogoFecha dFecha= new DialogoFecha();
        dFecha.show(getSupportFragmentManager (), "Fecha");
    }
});
```

Se puede, en el constructor, pasarle una fecha para que aparezca al crearse el diálogo.

Aunque no *Google* no lo recomienda puede usarse el diálogo de fecha y hora sin usar *DialogFragment*:

```
final Calendar c = Calendar.getInstance();
int mYear = c.get(Calendar.YEAR);
int mMonth = c.get(Calendar.MONTH);
int mDay = c.get(Calendar.DAY_OF_MONTH);
```

	RAMA:	Informática	CICLO:	Desenvolvememto de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

```

DatePickerDialog datePickerDialog = new DatePickerDialog(MainActivity.this, new DatePickerDialog.OnDateSetListener() {
    @Override
    public void onDateSet(DatePicker view, int year, int monthOfYear, int dayOfMonth) {
        tvFecha.setText(dayOfMonth + "-" + (monthOfYear + 1) + "-" + year);
    }
}, mYear, mMonth, mDay);
datePickerDialog.show();

```

4.3. TimePickerDialog

El uso de *TimePickerDialog* es muy similar a *DatePickerDialog*.

- Creamos, en *MainActivity*, el siguiente método que se ejecutará al aceptar una hora seleccionada.

```

public void hora(String s) {
    Toast.makeText(this, s, Toast.LENGTH_SHORT).show();
}

```

- Se crea la clase *DialogoHora* que hereda de *DialogFragment* e implementa el *listener OnTimeSetListener*:

```

public class DialogoHora extends DialogFragment implements TimePickerDialog.OnTimeSetListener {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Se coge la hora actual para inicializar el reloj. Se puede usar un constructor para inicializarlo con otra hora
        final Calendar c = Calendar.getInstance();
        int hora = c.get(Calendar.HOUR_OF_DAY);
        int minuto = c.get(Calendar.MINUTE);
        return new TimePickerDialog(getActivity(), this, hora, minuto, true); // Crea el diálogo y lo devuelve
    }

    @Override
    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
        // Al pulsar el botón se muestra un mensaje con la hora
        Toast.makeText(view.getContext(), hourOfDay + ":" + minute, Toast.LENGTH_SHORT).show();

        // O también se puede ejecutar un método que se encuentra en otra activity
        ((MainActivity) getActivity()).hora( hourOfDay + ":" + minute);
    }
}

```

- Se utiliza el cuarto botón creado en *MainActivity* para lanzar el *TimePickerDialog*:

```

Button boton4=(Button)findViewById(R.id.boton4);
boton4.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        DialogoHora dHora= new DialogoHora();
        dHora.show(getSupportFragmentManager(), "Hora");
    }
});

```

Para más información se puede consultar el siguiente enlace:

<https://developer.android.com/guide/topics/ui/controls/pickers.html?hl=es>


Aunque no Google no lo recomienda puede usarse el diálogo de fecha y hora sin usar *DialogFragment*:

```

final Calendar c = Calendar.getInstance();
int mHour = c.get(Calendar.HOUR_OF_DAY);
int mMinute = c.get(Calendar.MINUTE);

TimePickerDialog timePickerDialog = new TimePickerDialog(MainActivity.this, new TimePickerDialog.OnTimeSetListener() {
    @Override
    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
        tvHora.setText(hourOfDay + ":" + minute);
    }
}

```

	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

```
}, mHour, mMinute, false);
timePickerDialog.show();
```

5. CalendarView

Calendar nos va a permitir seleccionar una fecha desde un calendario. Dispone, entre otros, de los siguientes métodos:

Método	Descripción
getDate()	Fecha en milisegundos desde el 1 de enero 1970.
setDate()	Establece la fecha en milisegundos desde el 1 de enero 1970.
getFirstDayOfWeek()	Día de comienzo de la semana. 0 para el lunes y 1 para el domingo.
setFirstDayOfWeek	Establece el día de comienzo de la semana



El atributo XML *firstDayOfWeek* permite establecer el primer día de la semana: 0: Lunes 1: Domingo. Se puede realizar también mediante código.

Un ejemplo de su funcionamiento es el siguiente:

Ejemplo

```
CalendarView calendarView=findViewById(R.id.calendarView);
final Calendar fecha=Calendar.getInstance(); // Se obtiene la fecha mediante Calendar del paquete java.util
fecha.add(Calendar.MONTH,-3); // a la fecha actual se le restan tres meses
calendarView.setDate(fecha.getTimeInMillis()); // se establece en el widget del calendario la fecha deseada
calendarView.setFirstDayOfWeek(1); // se establece el primer día de la semana: 0: Lunes 1: Domingo
calendarView.setOnDateChangeListener(new CalendarView.OnDateChangeListener() {
    @Override
    public void onSelectedDayChange( CalendarView calendarView, int year, int month, int day) {
        fecha.set(year,month, day); // los parámetros son de la fecha seleccionada en el calendario

        SimpleDateFormat formatoFecha = new SimpleDateFormat("dd/MM/yyyy ");
        String cadFecha = formatoFecha.format(fecha.getTime());

        // Se muestra la fecha cada vez que se selecciona una en el calendario
        Toast.makeText(getApplicationContext(),cadFecha,LENGTH_SHORT).show();
    }
});
```

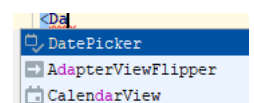
6. DatePicker


Permite seleccionar los días, meses y años mediante un componente de tipo calendario.

En las últimas versiones de Android Studio este componente no aparece en la paleta de componentes. Para añadirlo desde el editor XML se comienza a escribir el elemento `<DatePicker>` seleccionando *DatePicker* en el menú emergente completando los valores de ancho y alto y estableciendo su propiedad *id*.

Dispone, entre otros, de los siguientes métodos:

Método	Descripción
getFirstDayOfWeek()	Consigue el día de comienzo de la semana
setFirstDayOfWeek	Establece el día de comienzo de la semana
getDayOfMonth()	Consigue el día del mes actual del componente
getMonth()	Consigue el mes actual del componente
getYear()	Consigue el año del mes actual del componente
init(int year, int monthOfYear, int dayOfMonth, DatePicker.OnDateChangedListener onDateChangedListener)	Establece el método callback que gestiona cuando se cambia la fecha estableciendo además el año, mes y día que se mostrara en el componente.



	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

DatePicker dispone de los siguientes atributos XML interesantes:

- *DatePickerMode*: permite cambiar su apariencia: tiene los posibles valores de *calendar* y *spinner*.
- *firstDayOfWeek*: se utiliza para especificar el primer día de la semana.

Además dispone de varias propiedades XML para cambiar tanto los colores como la apariencia del componente.

Para más información ver: <https://developer.android.com/reference/android/widget/DatePicker.html>

Ejemplo

```
Calendar c=Calendar.getInstance();
int anho=c.get(Calendar.YEAR);
int mes=c.get(Calendar.MONTH);
int dia=c.get(Calendar.DAY_OF_MONTH);

final TextView textView=(TextView)findViewById(R.id.textView);
final DatePicker dp=(DatePicker)findViewById(R.id.datePicker);

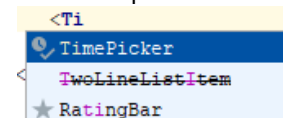
dp.init(anho, mes, dia, new DatePicker.OnDateChangedListener() {
    @Override
    public void onDateChanged(DatePicker view, int year, int monthOfYear, int dayOfMonth) {
        textView.setText(dayOfMonth + "/" + (monthOfYear + 1) + "/" + year);
    }
});
```



7. TimePicker

Permite seleccionar las horas y minutos una hora mediante un componente (tanto *TimePicker* como *DatePicker* se verán en el tema siguiente se verán como diálogos).

En las últimas versiones de Android Studio este componente no aparece en la paleta de componentes. Para añadirlo desde el editor XML se comienza a escribir el elemento `<TimePicker` seleccionando *TimePicker* en el menú emergente completando los valores de ancho y alto y estableciendo su propiedad *id*.




Dispone, entre otros, de los siguientes métodos:

Api >=23	Api <23	Descripción
<code>getHour()</code>	<code>getCurrentHour()</code>	Consigue la hora actual del componente
<code>getMinute()</code>	<code>getCurrentMinute()</code>	Consigue los minutos actuales del componente
<code>setHour()</code>	<code>setCurrentHour()</code>	Establece la hora actual del componente
<code>setMinute()</code>	<code>setCurrentMinute()</code>	Establece los minutos actuales del componente
<code>is24HourView()</code>		Comprueba si el componente está en modo 24 horas
<code>setIs24HourView(true/false)</code>		Establece si el componente se muestra en modo 24 horas o en modo 12 horas

TimePicker dispone del atributo XML *timePickerMode* que permite cambiar su apariencia: tiene los posibles valores de *clock* y *spinner*.

Para más información ver: <https://developer.android.com/reference/android/widget/TimePicker.html>

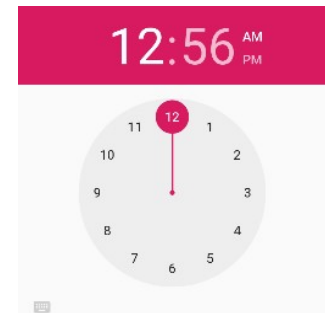
	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Ejemplo

```

Calendar cal=Calendar.getInstance();
final int hour = cal.get(Calendar.HOUR_OF_DAY);
final int minute = cal.get(Calendar.MINUTE);
final TextView textView=(TextView)findViewById(R.id.textView);
final TimePicker timePicker=(TimePicker)findViewById(R.id.timePicker);
timePicker.setIs24HourView(true);
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M){
    timePicker.setHour(hour);
    timePicker.setMinute(minute);
} else {
    timePicker.setCurrentHour(hour);
    timePicker.setCurrentMinute(minute);
}
timePicker.setOnTimeChangedListener(new TimePicker.OnTimeChangedListener() {
    @Override
    public void onTimeChanged(TimePicker timePicker, int hora, int minuto) {
        textView.setText(hora+":"+minuto);
    }
});

```




Tarea 3:

Continúa con el ejercicio 1 añadiéndole las siguientes funcionalidades:

En la actividad principal añade dos botones y dos *textView*:

- El primer botón lanza una nueva *activity* que permite escoger una fecha. Esta actividad dispone además de dos botones centrados en el *layout*. El primero cancela la selección de fecha volviendo a la actividad principal. El segundo envía la fecha escogida a la actividad principal visualizándose esta en la primera de las *textView* añadidos.
- El segundo botón hace exactamente lo mismo pero en vez de para una fecha para una hora. Esta tiene que estar en el formato de 24h y con dos dígitos tanto para la hora como para los minutos.

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

8. Navegación por pestañas⁴

Una forma muy típica de navegar por una aplicación es mediante pestañas. Estas pestañas normalmente se colocan justo por debajo de la *ActionBar* y permite cambiar de pestaña tanto pulsado sobre ellas como mediante un gesto de desplazamiento realizado con los dedos.

Para ello disponemos de los siguientes componentes:

- *CoordinatorLayout*⁵: es un tipo de *layout* que permite controlar, de forma automática, las animaciones entre los distintos componentes que se definen dentro de él, será el contenedor de la *toolbar* y las pestañas.
- *AppBarLayout*⁶: es un *LinearLayout* vertical que da soporte a muchas de las características de *Material design*. Para que funcionen todas sus características se debe definir dentro de un *CoordinatorLayout*.
- *Toolbar*: como se comentó en el tema anterior un *toolbar* representa una barra de opciones y es un contenedor que puede albergar, dentro de él, otros componentes como pueden ser botones, *RadioButtons*, *Spinners*, ... En este apartado su uso no es obligatorio.
- *TabLayout*⁷: es un *layout* horizontal que alberga pestañas.
- *ViewPager*⁸: es un *layout* que permite moverse entre pestañas mediante gestos de desplazamiento realizados con los dedos. Está ligado al uso de *fragments*.

En nuestro ejemplo cada una de las pestañas será un *fragment* y mediante un *ViewPager* podremos desplazarnos, usando los dedos, entre dichas pestañas.

8.1. *layout_scrollFlags*:

En un *tabLayout* el comportamiento normal de un *view*, por ejemplo la *toolbar*, es que aparezca cuando solo cuando hacemos *scroll* en la pantalla y llegamos a la parte superior de la misma.

Las siguientes opciones de *layout_scrollFlags* varían este comportamiento:

- *scroll*: debe especificarse siempre para permitir que los componentes se animen.
- *enterAlways*: el objeto que define la propiedad aparece cuando nos movemos en la página hacia arriba incluso si no se ha llegado al tope de la página.
- *enterAlwaysCollapsed*: si además de *enterAlways* se define *enterAlwaysCollapsed* junto con un tamaño mínimo (*minHeight*) cuando nos movemos hacia arriba en la página aparece el componente hasta su tamaño mínimo y es cuando se llega al tope cuando se muestra en su totalidad.
- *exitUntilCollapsed*: cuando se define esta propiedad junto con un tamaño mínimo (*minHeight*) cuando nos desplazamos hacia abajo la componente se oculta hasta el tamaño mínimo especificado.
- *snap*: el comportamiento varía dependiendo de cuando *scroll* se haya hecho. Si se ha hecho un *scroll* menos que la mitad del tamaño del componente su tamaño se reduce y si se ha hecho de más de la mitad el componente desaparece.

Para más información se puede consultar el siguiente enlace:

<https://guides.codepath.com/android/handling-scrolls-with-coordinatorlayout>


⁴ Basado en: <http://www.androidhive.info/2015/09/android-material-design-working-with-tabs/>

⁵ <https://developer.android.com/reference/android/support/design/widget/CoordinatorLayout>

⁶ <https://developer.android.com/reference/android/support/design/widget/AppBarLayout>

⁷ <https://developer.android.com/reference/android/support/design/widget/TabLayout>

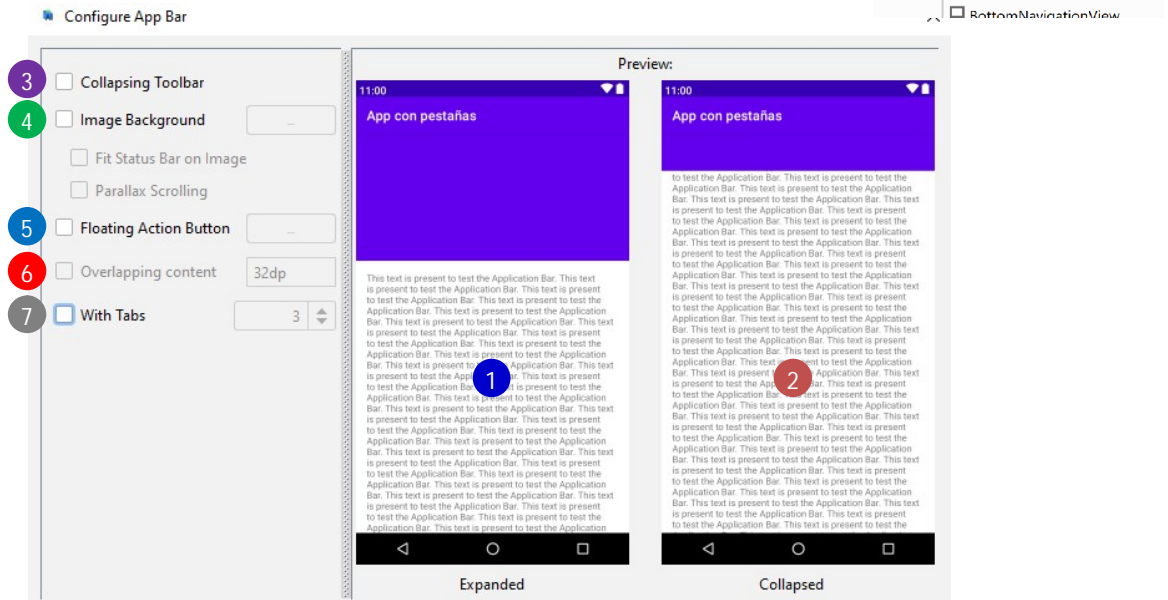
⁸ <https://developer.android.com/reference/android/support/v4/view/ViewPager>

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

8.2. Definición de las pestañas

Para añadir pestañas debemos añadir un componente `AppBarLayout` a nuestro *layout*.

Tras lo cual surge la siguiente ventana de configuración:



Donde:


1. Vista previa con los componentes expandidos.
2. Vista previa con los componentes sin expandir.

Se puede habilitar o deshabilitar que el componente *toolbar*, que se introduce de forma automática, se pueda contraer o no. Añade los valores `enterAlways` y `enterAlwaysCollapsed` a la propiedad `layout_scrollFlags` del contenedor. Si no se desea el componente *toolbar* se puede eliminar tanto editando el fichero XML del *layout* como desde el editor de *Layouts*.



Si se quiere que la *toolbar* no se pueda contraer tenemos que eliminar los valores que posee la propiedad `layout_scrollFlags`. `CollapsingToolbarLayout` y eliminar este.

4. Permite mostrar una imagen como fondo de `AppBarLayout`. No lo usaremos ya que no se pueden utilizar de forma simultánea con las pestañas. Usa un componente `CollapsingToolbarLayout` para poder contraer la *toolbar* con la imagen insertada.
5. Añade un `Floating Action Button` en la parte baja del *layout*. No lo usaremos.
6. Indica los `DPs` que el contenido del *layout* puede superponerse a la `AppBarLayout`. No lo usaremos.
7. Permite añadir un `TabLayout` que contendrá las pestañas. Marcaremos esta opción y pondremos el número de pestañas a 0 ya que añadiremos las pestañas mediante código y no en el fichero XML mediante elementos `TabItem`, que es lo que hace esta opción.

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Por lo que el fichero XML con el *layout* de la actividad principal quedaría de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <com.google.android.material.appbar.AppBarLayout
        android:id="@+id/appbar"
        android:layout_height="wrap_content"
        android:layout_width="match_parent">

        <androidx.appcompat.widget.Toolbar
            android:layout_height="?attr/actionBarSize"
            android:layout_width="match_parent"
            app:layout_scrollFlags="scroll|enterAlways">
        </androidx.appcompat.widget.Toolbar>

        <com.google.android.material.tabs.TabLayout
            android:id="@+id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:layout_scrollFlags="scroll|enterAlways"
            app:tabMode="scrollable">
        </com.google.android.material.tabs.TabLayout>

    </com.google.android.material.appbar.AppBarLayout>

    <androidx.core.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="com.google.android.material.appbar.AppBarLayout$ScrollingViewBehavior">

        <androidx.constraintlayout.widget.ConstraintLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            tools:context=".MainActivity" />
    </androidx.core.widget.NestedScrollView>

</androidx.coordinatorlayout.widget.CoordinatorLayout>
```


Podemos ver que la estructura generada está contenida en un *CoordinatorLayout*, sustituyendo a *ConstraintLayout*. Dentro de él se sitúa una *AppBarLayout* que contiene un *Toolbar*, esto es opcional, y un *TabLayout* en donde insertaremos las pestañas. Así mismo define un *ConstraintLayout*, para insertar el contenido de la página, dentro de un *NestedScrollView*, que es un contenedor, visto en el tema dos, que nos permite tener contenido al que se le puede realizar un *scroll*.

Vamos a introducir ahora las pestañas y su contenido.

- En primer lugar deberemos, añadir las siguientes cadenas al fichero *strings.xml*. Estas cadenas serán el contenido de cada pestaña.

```
<string name="uno">Fragment uno</string>
<string name="dos">Fragment dos</string>
<string name="tres">Fragment tres</string>
```

- El contenido de cada pestaña será un *fragment*. En nuestro ejemplo crearemos tres *fragments* (usando el asistente): *UnoFragment*, *DosFragment* y *TresFragment* que contienen exactamente el mismo código y únicamente difieren en el texto mostrado. Este texto se visualizará en un *TextView* presente en cada *fragment*.

	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

No tocaremos el código generado en cada *fragment* pero si su *layout*. El *layout* de *fragment_uno.xml* será:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".UnoFragment">

    <TextView
        android:id="@+id/tvfragment1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center_vertical|center_horizontal"
        android:text="@string/uno"
        android:textSize="36sp" />

</FrameLayout>
```

No indicamos el *layout* del *fragment* dos y del *fragment* tres ya que se definen de la misma forma con las oportunas modificaciones.

- Una vez tenemos diseñados los *fragments* definimos en la *activity* principal las pestañas y el *ViewPager* que nos permite realizar el desplazamiento entre estas.

Añadimos la definición del *ViewPager* al fichero XML con el *layout* de la actividad principal:

```
<androidx.viewpager.widget.ViewPager
    android:id="@+id/viewpager"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"/>
```


- El *ViewPager* necesita un adaptador que le diga que *fragment* poner en cada gesto *swipe*. Para ello creamos una nueva clase que hereda de *FragmentPagerAdapter* y que sirve precisamente para eso. La denominamos *Adaptador*.

```
public class Adaptador extends FragmentPagerAdapter {
    private final List<Fragment> mFragmentManager = new ArrayList<>();
    private final List<String> mFragmentTitleList = new ArrayList<>();

    public Adaptador(FragmentManager manager) {
        super(manager);
    }

    // Devuelve distintos fragments dependiendo del índice que le pasa el ViewPager
    @Override
    public Fragment getItem(int position) {
        return mFragmentManager.get(position);
        // Si no se desea usar una lista, podríamos usar algo similar a:
        // switch (position) {
        //     case 0: return new UnoFragment();
        //     case 1: return new DosFragment();
        //     case 2: return new TresFragment();
        //     default: return null;
        // }
    }

    // Se debe poner aquí el n° de pestañas para una correcta indexación
    @Override
    public int getCount() {
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

```

        return mFragmentManager.size();
    }

    @Override
    public CharSequence getPageTitle(int position) {
        return mFragmentManager.get(position);
    }

    public void addFragment(Fragment fragment, String title) {
        mFragmentManager.add(fragment);
        mFragmentManager.add(title);
    }
}

```

- Con lo anterior ya podemos modifica *MainActivity* para que quede de la siguiente forma:

```

public class MainActivity extends AppCompatActivity {
    private TabLayout tabLayout; // Encargado de pestañas
    private ViewPager viewPager; // Encargado del Swipe

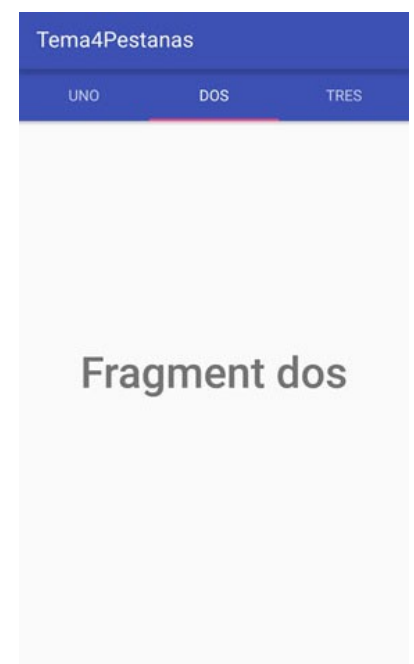
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);


        // Creamos el adaptador y le añadimos tres tabs
        Adaptador adapter = new Adaptador(getSupportFragmentManager());
        adapter.addFragment(new UnoFragment(), "Uno");
        adapter.addFragment(new DosFragment(), "Dos");
        adapter.addFragment(new TresFragment(), "Tres");

        // Obtenemos el viewPager y establecemos el adaptador
        viewPager = findViewById(R.id.viewpager);
        viewPager.setAdapter(adapter);

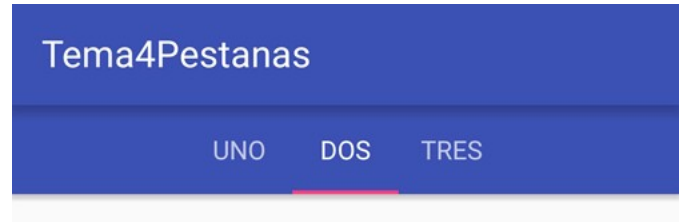
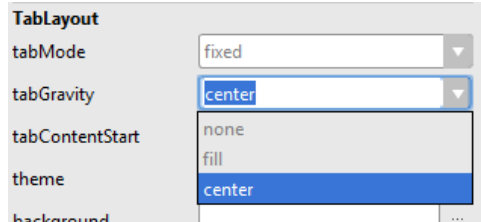
        // Obtenemos el tablayout y lo enlazamos con un viewPager
        tabLayout = findViewById(R.id.tabs);
        tabLayout.setupWithViewPager(viewPager);
    }
}

```



	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

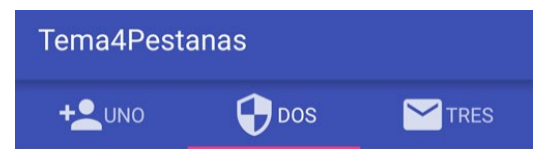
Por defecto, las pestañas disponibles se distribuyen desde el lado izquierdo. Si queremos solo ocupen el espacio necesario, centrándose en el medio, cambiaremos la propiedad *tabGravity*, de *tabLayout* a *center* (también se puede realizar mediante código desde el *TabLayout* con el método *setTabGravity* y los valores *TabLayout.GRAVITY_CENTER* y *TabLayout.GRAVITY_FILL*) y cambiando *tabMode* al valor *fixed*:



Se puede, además de un texto, mostrar un icono al lado de este. Para eso usando *tabLayout* se obtiene una pestaña y se le añade el un icono. Dependiendo de la versión de la biblioteca *Design Support Library* la disposición de los iconos puede cambiar de situarse a la izquierda del texto a situarse sobre el mismo. Para cambiar este comportamiento se puede utilizar el atributo *tabInlineLabel* de *tabLayout*.

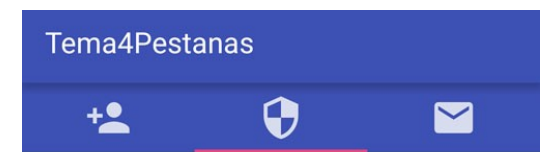
Añadir el siguiente código en *MainActivity* después de enlazar el *tabLayout* con *ViewPager* y de haber creado los iconos.

```
tabLayout.getTabAt(0).setIcon(R.drawable.add_user);
tabLayout.getTabAt(1).setIcon(R.drawable.protect);
tabLayout.getTabAt(2).setIcon(R.drawable.mensaje);
```



Si se desea que únicamente se muestren los iconos, ocultando el texto, disponemos de dos opciones:

- En *MainActivity* creamos el *fragment* pasándole la cadena vacía en el título.
- En el método *getPageTitle* del adaptador devolvemos *null*.



Se puede cambiar el color de los elementos de las pestañas mediante las siguientes propiedades XML de *TabLayout*.


- Color de fondo:
`android:background="@color/colorPrimary"`
- Color del texto de la pestaña no seleccionada:
`app:tabTextColor="@color/blanco"`
- Color del texto de la pestaña seleccionada:
`app:tabSelectedTextColor="@color/colorAccent"`
- Color de los iconos:
`app:tabIconTint="@color/blanco"`
- Color de la marca de la pestaña seleccionada y su ancho:
`app:tabIndicatorColor="@color/naranja"`
`app:tabIndicatorHeight="3dp"`

Siendo *@color/blanco* y *@color/naranja* colores definidos en el fichero *colors.xml*.

Se pueden establecer los mismos colores mediante código:

```
tabLayout.setBackgroundColor(getColor(R.color.colorPrimary)); // color de fondo
tabLayout.setSelectedTabIndicatorColor(getColor(R.color.naranja)); // color del indicador de pestaña seleccionada
tabLayout.setTabTextColors(getColor(R.color.blanco), getColor(R.color.colorAccent)); // color normal, color seleccionado
tabLayout.setTabIconTintResource(R.color.blanco); // color de los iconos
```

La forma de establecer el color del icono seleccionado se verá en un punto siguiente.

	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Pero ¿que pasa si el número de pestañas es muy grande?. Para probarlo de forma facil realizaremos las siguientes modificaciones a nuestro código:

- Cambiamos el codigo de *TresFragment* por el siguiente. Con este código conseguimos modificar de forma dinámica el contenido de *TextView* que posee.

```
public class TresFragment extends Fragment {
    String cad=null;

    public TresFragment() { // Required empty public constructor
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

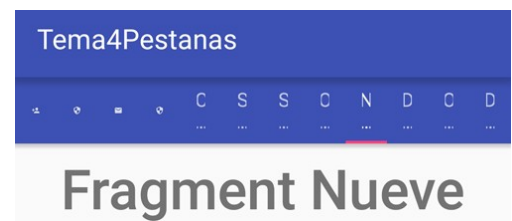
    public void setContenido(String cad) {
        this.cad=cad;
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View layout= inflater.inflate(R.layout.fragment_tres, container, false);
        if (this.cad==null) return layout;
        else {
            TextView tv=(TextView)layout.findViewById(R.id.tvfragment3);
            tv.setText(this.cad);
            return layout;
        }
    }
}
```

- Insertamos el siguiente código en *MainActivity* para añadir, dinámicamente, las siguientes pestañas:

```
TresFragment nuevaTab;
String[] tabs={"Cuatro", "Cinco", "Seis", "Siete", "Ocho",
              "Nueve", "Diez", "Once", "Doce"};

for (String t : tabs) {
    nuevaTab=new TresFragment();
    nuevaTab.setContenido("Fragment "+t);
    adapter.addFragment(nuevaTab, t);
}
```

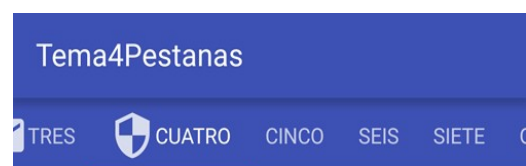
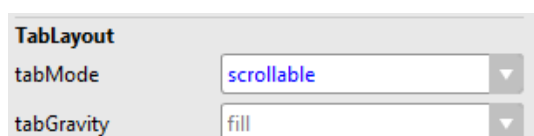



Si se añaden las pestañas tras establecer el adaptador deberemos notificar al adaptador que se han añadido pestañas mediante la instrucción:

```
adapter.notifyDataSetChanged();
```

Si ejecutamos el código veremos que todas las pestañas, se reparten el espacio disponible, imposibilitando su lectura.

¿Cómo solucionamos esto?. Cambiando la propiedad *tabMod*, de *tabLayout* de *fixed* a *scrollable*. De esta manera cada pestaña ocupara el espacio que necesite y podremos navegar por las pestañas desplazándolas con el dedo de izquierda a derecha (también se puede realizar mediante código desde el *TabLayout* con el método *setTabMode* y los valores *TabLayout.MODE_FIXED* y *TabLayout.MODE_SCROLLABLE*):



	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Si tenemos un *ToolBar* dentro del contendor *AppBarLayout* podemos trabajar con él como con cualquier componente *ToolBar*. Por ejemplo para añadirle un título se puede realizar mediante el siguiente código:

```
ToolBar toolbar=findViewById(R.id.toolbar);
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
    toolbar.setTitle("Componente ToolBar");
}
```

Se puede detectar el cambio de pestaña y ejecutar código asociado a ello mediante tanto desde el *ViewPager* como desde el *TabLayout*. Este último nos da más opciones:

- Desde *ViewPager*:

```
viewPager.addOnPageChangeListener(new ViewPager.OnPageChangeListener() {
    @Override
    public void onPageScrolled(int i, float v, int i1) {
    }

    @Override
    public void onPageSelected(int i) {
        Log.i("cambio", "ViewPager: pestaña "+i);
    }

    @Override
    public void onPageScrollStateChanged(int i) {
    }
});
```

- Desde *TabLayout*: por ejemplo cambiaremos el color del icono seleccionado.

```
tabLayout.addOnTabSelectedListener(new TabLayout.BaseOnTabSelectedListener() {
    @Override
    public void onTabSelected(TabLayout.Tab tab) {
        Log.i("cambio", "TabLayout pestaña "+tab.getPosition()+" "+tab.getText());
        tabLayout.getTabAt(tab.getPosition()).getIcon().setTint(getColor(R.color.naranja));
    }

    @Override
    public void onTabUnselected(TabLayout.Tab tab) {
        tabLayout.getTabAt(tab.getPosition()).getIcon().setTint(getColor(R.color.blanco));
    }

    @Override
    public void onTabReselected(TabLayout.Tab tab) {
    }
});
```


Esto puede por ejemplo si se quiere que al cambiar de pestaña la *AppBarLayout* aparezca siempre expandida.

Por ejemplo:

```
tabLayout.addOnTabSelectedListener(new TabLayout.BaseOnTabSelectedListener() {
    @Override
    public void onTabSelected(TabLayout.Tab tab) {
        AppBarLayout appBarLayout=findViewById(R.id.appbar);
        appBarLayout.setExpanded(true, true);
    }

    @Override
    public void onTabUnselected(TabLayout.Tab tab) {
    }

    @Override
    public void onTabReselected(TabLayout.Tab tab) {
    }
});
```

	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO	Programación Multimedia y Dispositivos Móviles					CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022		
	UNIDAD		COMPETENCIA					

9. Guías de interés

- *Resource manager*. Gestionar los recursos de una aplicación.
<https://developer.android.com/studio/write/resource-manager?hl=es>
- *Profile*. Comprobar el rendimiento y uso de recursos: memoria, CPU, ...
<https://developer.android.com/studio/profile?hl=es>
- Como publicar una aplicación
<https://developer.android.com/studio/publish?hl=es>
- Editor de temas
<https://developer.android.com/studio/write/theme-editor?hl=es>
- Cómo crear íconos de aplicaciones con *Image Asset Studio*
<https://developer.android.com/studio/write/image-asset-studio?hl=es>
- Crear gráficos vectoriales con *Vector Asset Studio*
<https://developer.android.com/studio/write/vector-asset-studio?hl=es>
- Crear imágenes *webp*⁹ y su uso
<https://developer.android.com/studio/write/convert-webp?hl=es>
- Mapas de bits de tamaño modificable: archivos *9-Patch*
<https://developer.android.com/studio/write/draw9patch?hl=es>

10. ¿Y ahora por dónde sigo?

La plataforma *Android* es realmente muy amplia en relación a la cantidad de componentes y elementos de la interfaz de usuario de que dispone. Además es un sistema dinámico en el cual en nuevas versiones aparecen nuevos componente o cambian los que ya hay (por ejemplo *DatePicker*).

Por esto es necesario estar al día y pensar muy bien la fase de diseño antes de ponerse a realizar cualquier aplicación ya que existen otros componentes muy útiles a la hora de diseñar aplicaciones. Por tanto es más que recomendable que antes de ponerse un programador a realizar cierta aplicación, echar un vistazo a los distintos componentes y, quizá aún más importante, que vea las recomendaciones de diseño de *Google*.

Esta es la web de guía de diseño:


<https://developer.android.com/intl/es/design/index.html>

Y en concreto en Patrones (*Patterns*) y en bloques de construcción (*Building Blocks*) existen un montón de indicaciones de cómo usar tanto los componentes y elementos vistos como los que aún no hemos visto o no tendremos tiempo de ver.

Las pestañas añadidas en el punto anterior pertenecen a *Design Support Library*. Podemos consultar los componentes que añade a Android consultando los siguientes enlaces.

- https://github.com/codepath/android_guides/wiki/Material-Design-Primer
- https://github.com/codepath/android_guides/wiki/Design-Support-Library

⁹ <https://es.wikipedia.org/wiki/WebP>

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Ejercicios

- Haz una calculadora de forma que cuando esté en vertical (*Portrait*) disponga solamente de las operaciones de suma resta multiplicación y división además de pasar a 0 (*reset*) y la coma decimal. Dicho teclado será un *fragment*.

Si dicha calculadora se pone en horizontal además del *fragment* anterior, aparecerá en otro *fragment* botones para halla el cuadrado, la raíz cuadrada, el inverso ($1/x$) el factorial es seno el coseno y la tangente. También habrá un inversor de signo (+/-)

Debe mantener el dato que tiene el usuario en el *EditBox*.

(Opcional) Añade más posibilidades de las especificadas.
- Retoma el ejercicio 5 del tema anterior (aplicación de películas) y modifica el interfaz. Si aún no lo has hecho empieza ya por esta versión.

Ahora debe estar distribuido en pestañas de la siguiente forma:

Listado completo, favoritos y salas serán pestañas. Añadir nuevo será una elemento en el *actionBar* con el icono + (añadir).

En el listado completo cada uno de los *items* debe responder a un menú contextual con solo una opción: Eliminar. Si se selecciona debe pedir confirmación con un *alert dialog*.

La petición de fecha de la película será ahora con un diálogo.

Debe funcionar tanto en horizontal como en vertical.
- (Opcional) Realiza una aplicación pensada por ti en la cual utilices además de los componentes vistos algunos no explicados en clase.