Capítulo 22 - Fragments en Kotlin

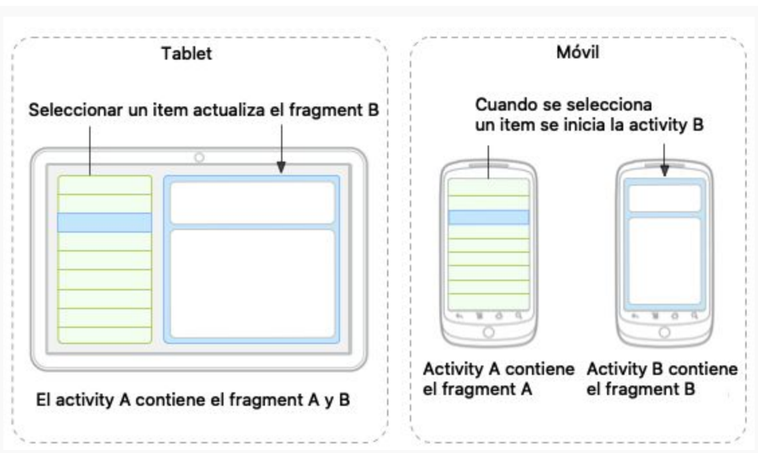
20 noviembre, 2019 17 Min de Lectura

En este capítulo volveremos un poco atrás, a uno de las bases de Android que, aunque su popularidad ha disminuido un poco, es imprescindible conocer su funcionamiento para los siguientes capítulos del blog (y para muchos de los desarrollos personales que podamos tener).

¿Qué es un fragment?

Los **fragments**, representan un componente básico a la hora de desarrollar una app. A día de hoy conocemos las *activities*, las cuales explicamos en el capítulo 12. Ahora imaginemos que el fragment es una variante de la *activity* pero con menos potencia, pero más manejable, es decir, su comportamiento es muy similar (tienen un *layout* y una clase para programar) pero tienen que estar dentro de una *activity*. La diferencia reside en que este componente es más reutilizable, y se puede añadir junto a más *fragments* en una misma *activity*.

Su popularidad empezó con la moda de las tablets, eran pantallas muy grandes que permitían aprovechar más los *fragments* añadiendo varios en una sola activity, mostrando más información que en un móvil.



Ejemplo de carga de fragments en distintos dispositivos.

Están disponibles desde el API 11 (**Android 3.0**) y desde entonces han salido tres versiones distintas.

Mostrar 10 registros Buscar:

Librería	Paquete
Nativo	android.app.Fragment
Support Library	android.support.v4.app.Fragment
AndroidX Library	androidx.fragment.app.Fragment

Mostrando desde 1 hasta 3 de 3 registros Anterior Siguiente

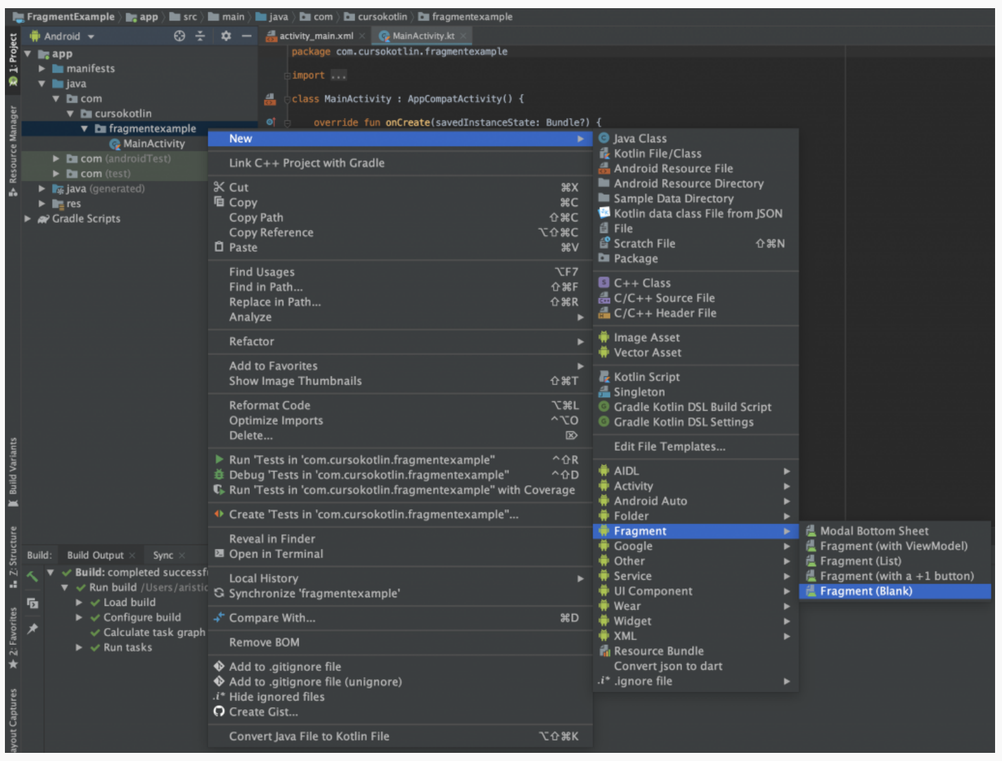
Si miramos la tabla anterior, vemos que la primera librería es la nativa (los *fragments* que salieron en el API 11), aunque son perfectamente funcionales, estás **deprecados**. Es decir, Google piensa que su código ya no está optimizado para las nuevas versiones, y por eso sacó los nuevos Fragments en la **Support Library**, que será la que usaremos. Terminamos con **Android X**, que pertenece a las nuevas librerías de Google pero las veremos más tarde, aunque si quieres informarte más puedes hacerlo aquí. Terminamos la tabla con el paquete, que significa el tipo de **import** que tenemos que hacer en nuestra clase para que lo detecte (lo veremos mejor durante a lo largo del artículo).

Aunque ya no se usan tan a menudo, todavía son imprescindibles para varios componentes que ves en el día a día de tus aplicaciones favoritas, por lo que es muy necesario entender completamente su funcionamiento para el diseño avanzado de vistas.

Let's code!

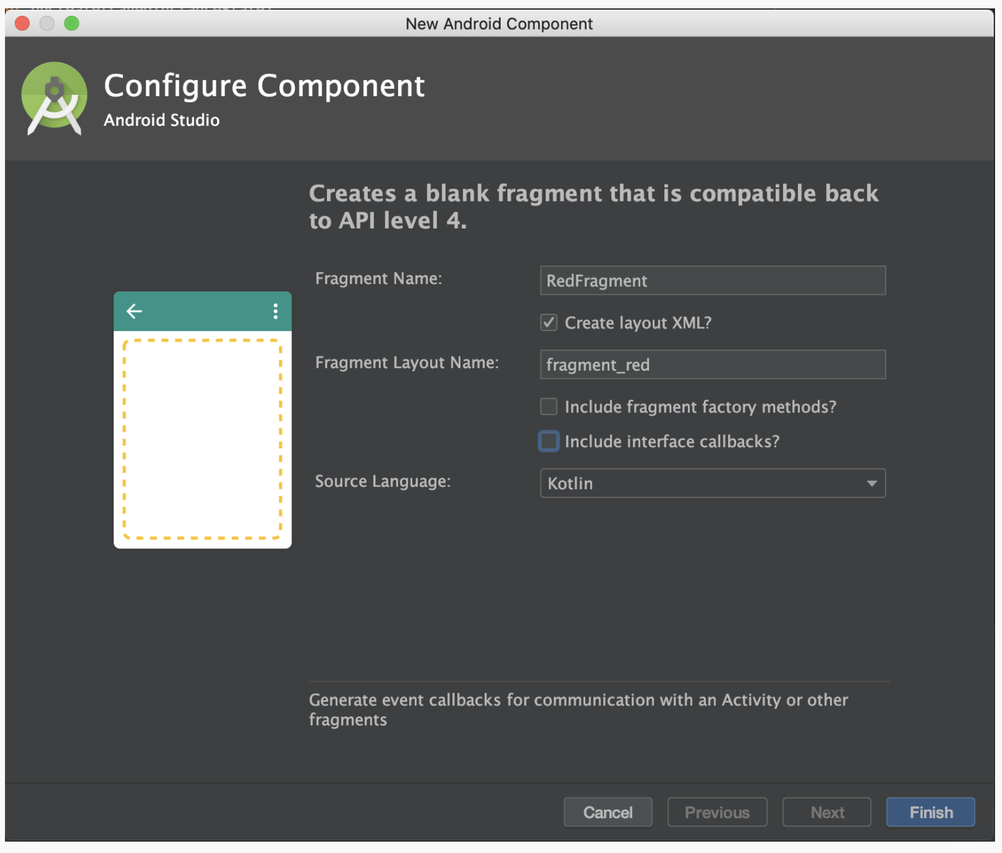
Vamos a crear un nuevo proyecto llamado **FragmentExample**. La idea de esta app será muy sencilla. En nuestra *activity* principal incluiremos dos *fragments* que ocuparán la mitad de la pantalla respectivamente. Cada una tendrá un botón que contactará con la *activity* para mostrar un mensaje.

Una vez tengamos creado nuestro proyecto, iremos a la ruta donde está la clase **MainActivity** y en el paquete (*nombreDeTuPaquete/fragmentexample*) haremos clic derecho y seleccionaremos **New>Fragment>Fragment (Blank)**.



Añadiendo nuestro primer fragment.

A continuación nos saldrá una pantalla de confirmación que nos permitirá añadir el nombre y tendrá varias opciones distintas a las que habitualmente encontramos al crear una *activity*. Sólo marcaremos la primera, **Create layout XML?** El resto las desactivaremos, pues nos creará código por defecto que en esta ocasión no nos interesa, pues lo iremos haciendo nosotros poco a poco. Lo llamaremos **RedFragment**.



Ejemplo de opciones seleccionadas al crear un fragment.

Al aceptar tendremos un código similar a lo siguiente.

```
class RedFragment : Fragment() {

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_red, container, false)
    }

}
```

Lo primero que haremos será darle un diseño muy sencillo. Definiremos los dos colores que usaremos en la app. Iremos a *app/res/values/colors.xml*. Si no existe lo crearemos con clic derecho encima de values (new>Values resource file). Dentro veremos tres colores creados, **no los borres,** puesto que el sistema siempre necesitará esos tres para pintar la app. Añadiremos dos más, **red** y **blue**.

```
<color name="blue">#80D8FF</color>
<color name="red">#FF80AB</color>
```

Si no has trabajado con recursos, esto no es más que un «atajo» para definir los colores que usaremos en la app. Se trata de definir un nombre (en este caso red y blue) y asignarles un código de color hexadecimal. Si quieres probar otros colores, basta con hacer clic en el recuadro del color que aparece al principio de la línea y jugar con el editor visual.

Ahora volvemos al *fragment* y accedemos a su *layout*, que se llama **fragment_red**.

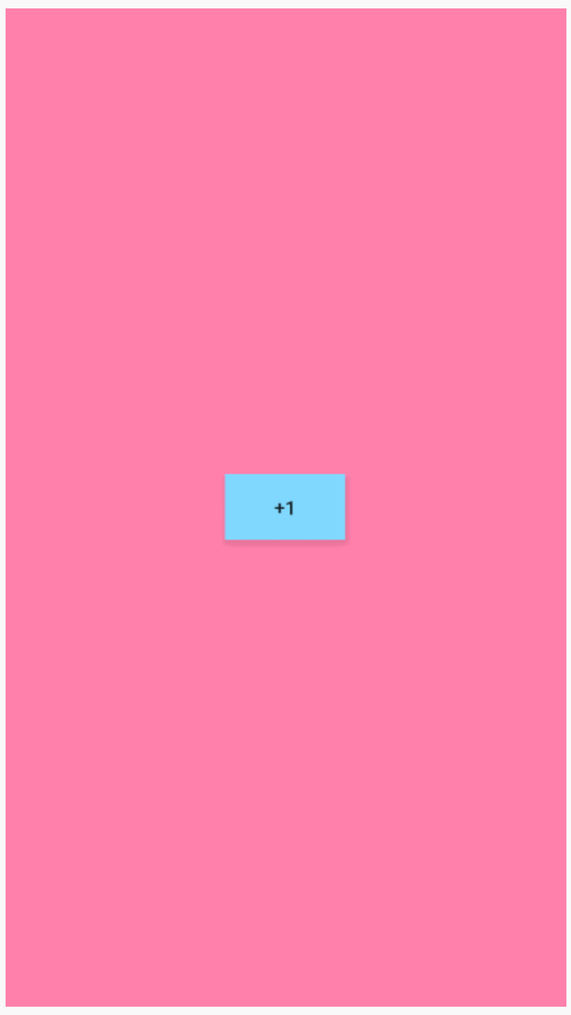
El diseño será muy sencillo, un *background* de color rojo y un botón azul en el centro.

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

    <Button
        android:id="@+id/btnPlus"
```

```
        android:text="+1"
        android:layout_gravity="center"
        android:background="@color/blue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</FrameLayout>
```



Diseño del primer fragment.

Ahora Repetiremos el proceso, crearemos otro *fragment* llamado BlueFragment que tendrá un diseño opuesto, es decir el color del fondo será azul y el del botón rojo.

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

    <Button
        android:id="@+id/btnPlus"
        android:text="+1"
        android:layout_gravity="center"
        android:background="@color/red"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</FrameLayout>
```

Ahora que tenemos los dos *fragments* creados vamos a volver a nuestro **MainActivity** e iremos a su *layout*, llamado **activity_main**. Una vez dentro lo editamos y lo dejaremos así.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.cursokotlin.fragmentexample.RedFragment"
        android:id="@+id/first"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
    <fragment android:name="com.cursokotlin.fragmentexample.BlueFragment"
        android:id="@+id/second"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
</LinearLayout>
```

Analicemos el código anterior. Lo primero ha sido crear un *linearLayout* que lo que permite es colocar componentes de forma horizontal o vertical dependiendo del valor que pongamos en la propiedad **android:orientation**. Luego hemos creado dos componentes **fragment** que contiene un atributo **name** donde debemos poner la ruta de nuestro *fragment*, una **id** única para hacer referencia a cada uno de los fragments.

Los atributos **layout_height** y **layout_width,** como sabemos son obligatorios, pero en este caso hemos puesto 0dp en el **width**. Esto se debe a que una propiedad de los *linearLayout* es que si ponemos una anchura o altura a 0dp y le ponemos la propiedad **layout_weight**, podemos dividir en porcentaje el tamaño disponible. Es decir, si le ponemos **layout_weight=»1"** a cada uno de los componentes del *linearLayout* ambos ocuparán un porcentaje exacto de pantalla (en este caso 50%). Si cambiásemos y pusiéramos un peso de 2 en uno de los componentes, este ocupará un 66’6% mientras que el otro ocupará el 33,3% restante.

Si ejecutamos nuestra app ahora mismo, veríamos algo así.

Diseño actual de nuestra app.

Muy bonito, pero ahora mismo la app no hará nada. Para poder continuar debemos entrar y entender más el funcionamiento de los fragments.

Ciclo de vida de los fragments

El ciclo de vida de un fragment, no es más que los métodos por los que pasa desde que se crea hasta que se destruye, a modo de símil nuestro ciclo de vida si fueran funciones sería *nacer()*, *crecer()*, *reproducirse()* y *morir()*. Pues es básicamente lo mismo.

Ciclo de vida de los fragments

En la imagen anterior, veremos el ciclo de vida. Antes de explicarlo debemos entender que nuestro fragment extiende de una clase superior llamada *Fragment*, por ello, aunque no veamos estos métodos dentro de nuestra clase, se están ejecutando por detrás. Además, podemos sobreescribirlos y modificar su comportamiento para que haga lo que necesitemos, lo veremos a continuación.

Estados de creación

Lo primero que tenemos que hacer para iniciar el ciclo de vida, es añadir el fragment. En el ejemplo anterior, este evento se reproduce al ejecutar el *MainAcitivity*, pues en su *layout*, contiene dos fragments. Es en ese momento cuando empieza el ciclo, el siguiente paso será la función **onAttach()** que por decirlo de algún modo «ligará» nuestra *activity* a nuestro *fragment*, dándoles la oportunidad de que se puedan comunicar. Pasamos a **onCreate()** que se llamará al haber creado la instancia del fragment, es decir, lo estamos «construyendo».

Luego llegaremos a **onCreateView()**, que si nos fijamos en nuestros fragments, este método aparecerá con una palabra clave al principio (**override**), esto le dice al sistema que va a hacer lo que debe de hacer como norma general, pero que después vamos a modificar su comportamiento añadiendo funcionalidades extras, en este caso estamos devolviendo la vista que contendrá dicho fragment. Cuando la vista se ha terminado de crear, llegaremos a **onActivityCreated()**, que nos avisa de que ya está todo disponible. Este es un buen lugar donde añadir las funciones *onClick()* o inicializar variables del fragment.

Estado de creación

El método **onStart()**, se lanzará casi justo después del método anterior, y nos avisará cuando podamos trabajar con el fragment.

Estado de resumen

El siguiente es básicamente igual, **onResume()** actuará de modo similar pero será llamado más veces, por ejemplo si minimizamos la aplicación en el móvil y la volvemos a abrir, la función **onResume()** se volverá a llamar, pero **onStart()** no.

Estado de pausa

Cuando una nueva *activity* se pone por encima de la que aloja el *fragment*, pero esta primera activity no cubre totalmente la vista del *fragment*, empieza el método **onPause()**, pero aún en este estado la información que hay en el *fragment* se mantiene.

Estado de parada

Si se repite el proceso anterior, pero la pantalla es tapada totalmente o el *fragment* ha sido quitado de la *activity* se cambia al estado **onStop()**. El *fragment* detenido sigue activo, pero se destruirá si la *acitivity* que lo contiene se destruye.

Estados de destrucción

Contiene tres funciones distintas que se irán llamado de manera secuencial una vez los procesos van terminando. El primero será **onDestroyView()** que se llamará cuando la vista vaya a ser destruida, al terminar pasará por **onDestroy()** que pondrá fin a la vida del *fragment*, terminando por **onDetach()**, que hará lo contratio a la función **onAttach()** que vimos al principio, básicamente desliga la *activity con el fragment*.

Si nos fijamos básicamente en un bucle de crear y destruir, pero es necesario que entendamos que hace cada una de estas funciones para ver el siguiente paso.

Comunicación activity - fragment

Hay veces, como en este ejemplo que necesitamos que nuestra *activity* haga algo, ya sea para controlar y centralizar la lógica o porque hay distintas funciones que los fragments no pueden hacer. Por ello debemos entender que la forma más sencilla que tenemos de hacer esto es a través de los **listeners**.

Un *listener* no es más que una función que se llama en un sitio, pero avisa a otro de que ha sido llamado. Para que lo entendamos un *listener* es como la alarma de un bar, hay alguien que la contrata (en este caso la *activity*) pero se reproducirá cuando entre un desconocido en el bar (el *fragment*), es decir, el *listener* hará de comunicador entre *activity* y *fragment* (aunque se puede utilizar en muchísimos más casos).

Para ello lo primero que haremos será ir al directorio donde tenemos nuestra *activity* y *fragments* y con el botón derecho sobre el directorio, ir a **new>Kotlin File Class** y nos saldrá un diálogo para configurar nuestro fichero. Lo llamaremos **OnFragmentActionsListener** y en **kind** seleccionaremos **interface**.

Ejemplo de creación de interfaz.

Nuestra interfaz fija un contrato entre quien los usa, si nosotros lo añadimos a nuestro *activity*, todos los métodos que estén en la interfaz, deben añadirse a dicho *activity*. Crearemos una función en la interfaz, pero a diferencia de las clases con las que trabajaremos, solo definiremos la función con sus parámetros de entrada o salida si tuvieran, pero no la lógica, pues esta se tiene que implementar donde implementemos la interfaz.

```
1.  interface OnFragmentActionsListener {
2.      fun onClickFragmentButton()
3.  }
```

Hemos añadido una función llamada **onClickFragmentButton()** que no recibe parámetros ni devuelve nada. Ahora implementaremos esta interfaz en la clase **MainActivity**. Para ello lo añadiremos en la primera línea, quedando nuestra *activity* así.

```
class MainActivity : AppCompatActivity(), OnFragmentActionsListener {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

}
```

Si intentamos compilar ahora no nos dejará, pues si miran debajo del nombre de la clase tendrá una línea roja que nos avisa de un error.

Error de implementación en nuestra MainActivity.

Este error nos está diciendo que hemos implementado una interfaz pero no hemos añadido sus métodos, para ello haremos lo siguiente, iremos al menú superior **Code>Implement methods** y nos saldrá un diálogo con todos los métodos que debemos implementar, en este caso solo es uno.

Implementando métodos en Kotlin.

Nada más añadirlo vemos como se añade una nueva función a nuestra clase.

```
override fun onClickFragmentButton() {
    TODO("not implemented") //To change body of created functions use Fil
}
```

Como ya hemos hablado anteriormente, esta función tiene la palabra reservada **override** delante, y en su interior tiene un **TODO** que no es más que un recordatorio de que debemos hacer algo con él, así que podemos borrar esa línea.

Dentro de este método añadiremos un *Toast*, que si lo recuerdan de otros capítulos se trata de un mensaje que saldrá de la parte inferior de la pantalla al llamar al método.

```
override fun onClickFragmentButton() {
    Toast.makeText(this, "El botón ha sido pulsado", Toast.LENGTH_SHORT).
}
```

Con esto no tendríamos que hacer nada más en el *activity*. Volvemos a los fragments, yendo a **BlueFragment**.

Lo primero que haremos será añadir nuestro listener al *fragment*, para ello empezaremos declarando una variable **listener**, que sea del mismo tipo que la interfaz.

```
private var listener: OnFragmentActionsListener? = null
```

Esta variable la pondremos al inicio de la clase, y podrá ser **null**, así nos obligamos a comprobar su contenido antes de intentar ejecutar cualquier acción.

Luego para inicializarla usaremos uno de los métodos que vimos en el ciclo de vida, la función **onAttach()**.

```
override fun onAttach(context: Context) {
    super.onAttach(context)
    if (context is OnFragmentActionsListener) {
        listener = context
    }
}
```

Analicemos la función anterior. La primera línea llama a la función **super()**, que no es más que la forma que le decimos al sistema de que, aunque queramos que haga más cosas, ejecute el código que tenga que ejecutarse al llamar a esta función. Es decir, queremos añadir cosas pero que no se cargue lo que haría ese método normalmente.

Luego encontramos un *if()*, que comprobará si el contexto que llega a la función **onAttach()**, tiene implementada la interfaz que hemos creado. En este caso el contexto será de **MainActivity**, pues es esta clase la que crea el *fragment*. Para finalizar igualamos el *listener* que hemos declarado en la parte superior de la clase al contexto.

El último paso será que haremos será desconectar el *fragment* de la *activity* cuando vaya a morir, así que volveremos a hacer uso de otro método del ciclo de vida.

```
override fun onDetach() {
    super.onDetach()
    listener = null
}
```

Cuando la función **onDetach()** sea llamada, volveremos a hacer **null** el *listener* para asegurarnos de que no haya algún error por falta de comunicación. Ahora si tenemos funcionando nuestro listener.

La siguiente pregunta sería ¿Cómo ejecutamos el *listener*? Para ello haremos lo habitual, crearemos un *onClickListener()* para que cuando el botón del *fragment* se ejecute lo llame.

Si miramos el *layout* que creamos anteriormente veremos que el nombre de su id es **btnPlus**, así que le asignaremos el *onClick* en el método *onViewCreated* que como verás también forma parte del ciclo de vida.

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    btnPlus.setOnClickListener { listener?.onClickFragmentButton() }
}
```

Hemos hecho que al hacer clic en el botón, este llame a la función *onClickFragmentButton()* de la interfaz, que hará que nuestra *activity* lance el *toast*. Nuestro *fragment* completo quedaría de la siguiente manera.

```
class BlueFragment : Fragment() {
```

```
    private var listener: OnFragmentActionsListener? = null

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.fragment_blue, container, false)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        btnPlus.setOnClickListener { listener?.onClickFragmentButton() }
    }

    override fun onAttach(context: Context) {
        super.onAttach(context)
        if (context is OnFragmentActionsListener) {
            listener = context
        }
    }

    override fun onDetach() {
        super.onDetach()
        listener = null
    }
}
```

Repetiremos lo mismo con **RedFragment.**

```
class RedFragment : Fragment() {
    private var listener: OnFragmentActionsListener? = null

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.fragment_red, container, false)
    }

    override fun onActivityCreated(savedInstanceState: Bundle?) {
        super.onActivityCreated(savedInstanceState)
        btnPlus.setOnClickListener { listener?.onClickFragmentButton() }
    }

    override fun onAttach(context: Context) {
        super.onAttach(context)
        if (context is OnFragmentActionsListener) {
            listener = context
        }
    }

    override fun onDetach() {
        super.onDetach()
        listener = null
    }
}
```

Ejecutamos la aplicación y pulsamos los botones, veremos que sale el *toast*.

Añadiendo fragments por código

Con el ejemplo anterior hemos conseguido añadir fragments desde el **xml**, lo que nos permite añadirlos de una forma fácil y rápida. Compliquemos un poco las cosas ahora y veamos cómo podemos añadir un *fragment* mediante código (por ejemplo al pulsar un botón) o incluso reemplazarlo y cambiarlo dependiendo de la selección del usuario.

Para ello modificaremos el **xml** de nuestra clase **MainActivity**.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/purple"
    android:orientation="vertical">

    <Button
        android:id="@+id/btnRed"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:padding="8dp"
        android:layout_margin="5dp"
        android:background="@color/red"
        android:text="Fragment rojo" />

    <Button
        android:id="@+id/btnBlue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="8dp"
        android:layout_gravity="center_horizontal"
        android:layout_margin="5dp"
        android:background="@color/blue"
        android:text="Fragment azul" />
```

```
27.
28.      <FrameLayout
29.          android:id="@+id/fragmentContainer"
30.          android:layout_width="match_parent"
31.          android:layout_height="match_parent" />
32.  </LinearLayout>
```

Hemos añadido dos botones en la *activity* principal que serán los que se encarguen de cambiar el *fragment* mediante código y además hemos añadido un *FrameLayout* que será el contenedor de dichos *fragments*.

Seguramente el XML nos de un error, pues he añadido un nuevo color, vamos a **Res>Values>Colors** y añadimos el violeta.

```
1.  <color name="purple">#673AB7</color>
```

Vamos a la clase **ActivityMain** y crearemos un método llamado *loadFragment(fragment: Fragment)*.

```
1.  private fun loadFragment(fragment:Fragment) {
2.      val fragmentTransaction = supportFragmentManager.beginTransaction()
3.      fragmentTransaction.add(R.id.fragmentContainer, fragment)
4.      fragmentTransaction.commit()
5.  }
```

Esta función recibe un *fragment* que será el que queramos cargar en nuestro *FrameLayout* del xml. Primero creamos un objeto **fragmentTransaction**, este objeto se crea con **supportFragmentManager** que es el que se encarga de gestionar los *fragments* y posee la mayoría de los métodos que necesitamos en la gestión.

Para cargar un *fragment* haremos siempre lo mismo, necesitamos avisar al sistema de que vamos a hacer un cambio, añadirlo y guardar dicho cambio. Eso es exactamente lo que hace nuestra función anterior. Empezamos avisando del cambio con *beginTransaction()*, luego llamamos a la función *add()* del *fragmentTransaction* y lo cerramos con *commit()*.

Para terminar necesitamos añadir la función *onClickListener()* a nuestros botones para añadir los fragments.

```
1.  btnRed.setOnClickListener { loadFragment(RedFragment()) }
2.  btnBlue.setOnClickListener { replaceFragment(BlueFragment()) }
```

A parte de añadir el *listener* estamos creando un objeto de tipo **RedFragment** o **BlueFragment** dependiendo el botón en el que hemos hecho clic.

Añadiendo fragments mediante código.

Creo que este es el ejemplo perfecto por el cual soy programador y no diseñador.

Reemplazando fragments

En el ejemplo anterior vimos cómo añadir *fragments*, pero si nos fijamos, cada vez que añadimos uno, el anterior va a morir. Ahora pensemos que estamos realizando un flujo de trabajo y son dos pantallas, en la primera el usuario tiene que introducir sus datos personales y en la segunda su dirección. Si trabajamos con fragments tenemos la posibilidad de emular el funcionamiento de las *activities*, es decir, podemos pulsar atrás (tanto en la *toolbar* cómo el botón del móvil) y que vuelva al anterior.

Para ello tendremos que hacer dos cambios en el método *loadFragment()*.

```
1.  private fun replaceFragment(fragment: Fragment){
2.      val fragmentTransaction = supportFragmentManager.beginTransaction()
3.      fragmentTransaction.replace(R.id.fragmentContainer, fragment)
4.      fragmentTransaction.addToBackStack(null)
5.      fragmentTransaction.commit()
6.  }
```

Lo primero que hemos hecho ha sido renombrarlo, pues ya no añade *fragments*, sino que los reemplaza. Esta vez el objeto **fragmentTransaction** no llama a la función *add()*, sino a *replace()*. Para terminar antes de hacer el *commit()*, hemos añadido *addToBackStack(null)*, que con esto le decimos al sistema que permita ir para atrás.

```
1.  class MainActivity : AppCompatActivity(), OnFragmentActionsListener {
2.
3.      override fun onClickFragmentButton() {
4.          Toast.makeText(this, "El botón ha sido pulsado", Toast.LENGTH_SHO
5.      }
6.
7.      override fun onCreate(savedInstanceState: Bundle?) {
8.          super.onCreate(savedInstanceState)
9.          setContentView(R.layout.activity_main)
10.
11.         btnRed.setOnClickListener { replaceFragment(RedFragment()) }
12.         btnBlue.setOnClickListener { replaceFragment(BlueFragment()) }
13.     }
14.
15.     private fun replaceFragment(fragment: Fragment){
16.         val fragmentTransaction = supportFragmentManager.beginTransaction
17.         fragmentTransaction.replace(R.id.fragmentContainer, fragment)
18.         fragmentTransaction.addToBackStack(null)
19.         fragmentTransaction.commit()
20.     }
21. }
```

Si probamos el ejemplo completo anterior y abrimos un par de *fragments*, al pulsar volver para atrás nos llevará al *fragment* anterior.

Continúa con el curso: Capítulo 23 – Jetpack y AndroidX

Te recuerdo que puedes seguirme en mis redes sociales en Aristi.Dev. Y si tienes dudas con este o cualquier otro artículo del blog únete al Discord de la comunidad y te ayudaremos.

Curso de Android en Kotlin

Etiquetado en: CURSO KOTLIN EN ESPAÑOL, curso programacion android en kotlin, kotlin

Mostrar Comentarios

Compartir Artículo:

https://cursokotlin.com/capitulo-22-fragments-en-kotli

También podría interesarte.

Kotlin Multiplatform con Ktor para consumir API

- 13 junio, 2024

Cómo ser Android Developer - Ruta de aprendizaje [ROADMAP]

- 4 enero, 2024

Estados (STATES) en Jetpack Compose

- 21 noviembre, 2023

Otras historias.

Capítulo 23 - Jetpack y AndroidX

Siguiente Historia

Historia anterior

Capítulo 21 - Gestión de permisos en Android