UA03. PROGRAMANDO...

3.3.5. Fragment

Las actividades suelen entenderse como una pantalla completa en la que distribuir elementos gráficos como imágenes, botones y demás controles. Los fragmentos llegaron para dar mayor flexibilidad a la composición de las pantallas. Pueden ocupar toda o solo una parte de ella.

El **fragment** podría entenderse como una unidad de presentación, como una **subactividad**. En sí mismo, sería como un componente gráfico más.

Imaginemos un fragment llamado **DetallesClienteFragment** que muestre los datos de un cliente. Podríamos utilizar ese fragment, solo o en composición con otros, en varias actividades de la aplicación, sin ninguna repetición de código. El paso de un fragment a otro es más eficiente que el paso entre actividades y, además, **no es necesario declarar los fragmentos en el manifest**.

Los fragments, como las activities, tienen un **ciclo de vida** que debemos comprender y respetar si queremos que todo funcione como deseamos. Además, **un fragment siempre irá embebido en una activity**, que será la responsable de dirigir sus fragments.

Ejemplo

Selecciona **File > New > New Project > Basic Activity**. Se creará una app con una activity que servirá de contenedor para dos **fragments** diferentes. Además, veremos cómo hace

uso del relativamente reciente componente **Navigation**, que nos facilita la navegación entre fragments.

Quizá lo primero es ver cómo luce, así que lancemos la app en un emulador o terminal y juguemos con ella. Estudiando el código, veremos cómo Android Studio ha creado tres clases:

- MainActivity
- FirstFragment
- SecondFragment

El código de **MainActivity** no tiene nada de particular, nada que indique cómo se mostrarán los fragments. Pero pulsemos **Ctrl** y hagamos clic sobre **activity_main** para ver el layout de la actividad.

Vemos el **CoordinatorLayout** y la barra de herramientas **Toolbar**. Más abajo veremos un include hacia otro layout, que será donde se defina el contenido de la ventana. Mientras presionamos **Ctrl**, hacemos clic sobre **@layout/content_main**. Veremos el código **XML**:

```
<?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"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior'

    <fragment
        android:id="@+id/nav_host_fragment_content_main"
        android:name="androidx.navigation.fragment.NavHostFragme
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:defaultnavhost="true"
        app:layout_constraintbottom_tobottomof="parent"
        app:layout_constraintend_toendof="parent"
        app:layout_constraintstart_tostartof="parent"
        app:layout_constrainttop_totopof="parent"

```

```
        app:navgraph="@navigation/nav_graph">
</fragment>
</androidx.constraintlayout.widget.constraintlayout>
```

La etiqueta **fragment** servirá como contenedor de los fragments que queremos mostrar. El atributo **name** referencia a la clase **NavHostFragment**, que tendrá la funcionalidad de permitir la navegación entre fragments. Cada **NavHostFragment** tiene un **NavController** que define los posibles caminos de navegación y que se define mediante el gráfico de navegación. En el código podemos ver que el gráfico de navegación se define con el atributo **app:navGraph** y que, en este caso, su valor es **@navigation/nav_graph**. Pulsemos **Ctrl** y clic sobre **nav_graph** para ver el gráfico de navegación:

```
<?xml version="1.0" encoding="utf-8"?>
<navigation
    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:id="@+id/nav_graph"
    app:startdestination="@id/FirstFragment">

    <fragment
        android:id="@+id/FirstFragment"
        android:name="com.example.fragmentsapp.FirstFragment"
        android:label="@string/first_fragment_label"
        tools:layout="@layout/fragment_first">

        <action
            android:id="@+id/action_FirstFragment_to_SecondFragm
            app:destination="@id/SecondFragment">
        </action>
    </fragment>
    <fragment
        android:id="@+id/SecondFragment"

```

```
        android:name="com.example.fragmentsapp.SecondFragment"
        android:label="@string/second_fragment_label"
        tools:layout="@layout/fragment_second">

        <action
            android:id="@+id/action_SecondFragment_to_FirstFrag
            app:destination="@id/FirstFragment">
        </action>
    </fragment>
</navigation>
```

Observa cómo el grafico tiene un elemento raíz que define su **id** y establece el fragment que será visible al inicio con **app:startDestination**, que en este caso es el **FirstFragment**. Debajo veremos una lista de fragmentos definidos con la etiqueta con varias propiedades, como la clase definida con **android:name** y el layout con **tools:layout**. Una etiqueta hija define la acción de navegación de una etiqueta a otra. Igual que con los layouts, podemos ver una representación gráfica de la navegación entre fragments, pues tenemos las opciones en la esquina superior derecha: **Code**, **Split** y **Design** que ya conocemos de los puntos anteriores.

Veamos ahora el código del primer fragmento, haciendo **Ctrl** y clic sobre **FirstFragment**:

```

package com.example.fragmentsapp

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.navigation.fragment.findNavController
import com.example.fragmentsapp.databinding.FragmentFirstBinding

/**
 * A simple [Fragment] subclass as the default destination in th
```

```
 */
class FirstFragment : Fragment() {
    private var _binding: FragmentFirstBinding? = null
    // This property is only valid between onCreateView and
    // onDestroyView.
    private val binding get() = _binding!!
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        _binding = FragmentFirstBinding.inflate(inflater, contai
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: B
        super.onViewCreated(view, savedInstanceState)
        binding.buttonFirst.setOnClickListener {
            findNavController().navigate(R.id.action_FirstFragme
        }
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

Vemos que **FirstFragment** hereda de **Fragment** y sobrescribe **onCreateView** y **onViewCreated**. En la primera especifica el layout que diseña su contenido; en la segunda, establece un **OnClickListener** para el botón **button_first**. Cuando se pulse sobre el botón, se ejecutará el código **findNavController().navigate(action)**.

El **NavController** conoce el gráfico de navegación, y sabe que esa acción requiere navegar hacia el fragment **SecondFragment**. El segundo fragment es igual de sencillo: simplemente, hace el recorrido contrario.

```
1    override fun onViewCreated(view: View, savedInstanceState: B
2        super.onViewCreated(view, savedInstanceState)
3        binding.buttonSecond.setOnClickListener {
4            findNavController().navigate(R.id.action_SecondFragm
5        }
6    }
```

Ciclo de vida

Como se ha comentado anteriormente, los Fragments tienen su propio ciclo de vida.

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.

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.

Create

El ciclo de vida de un fragment comienza al ejecutar la actividad donde contenga los 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().** Este método aparece con una palabra clave al principio (**override**), esto permite modificar el comportamiento por defecto añadiendo funcionalidades extras. Por ejemplo, devolver la vista que contiene dicho fragment.

Cuando la vista se ha terminado de crear, llegaremos a **onViewCreated()**, que nos avisa de que ya está todo disponible. Este es un buen lugar donde añadir las funciones **onClick()** o inicializar variables del fragment.

Start

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

Resume

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.

Pause

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.

Stop

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.

Destroy

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 contrario a la función **onAttach()** que vimos al principio, básicamente desliga la activity con el fragment.