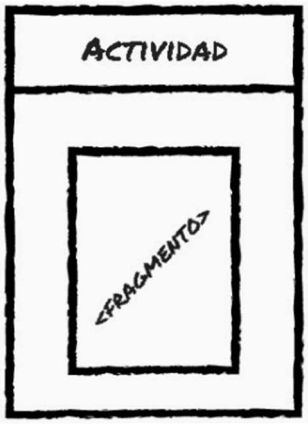
Fragmentos

Un Fragment representa una parte reutilizable de la IU de tu app. Un fragmento define y administra su propio diseño, tiene su propio ciclo de vida y puede administrar sus propios eventos de entrada. Los fragmentos no pueden existir por sí solos. Deben estar alojados por una actividad u otro fragmento.



Un **fragmento** es una sección «**modular**» de interfaz de usuario embebida dentro de una **actividad anfitriona**, el cual permite versatilidad y optimización de diseño. Se trata de miniactividades contenidas dentro de una actividad anfitriona, manejando su propio diseño (un ***recurso layout*** propio).

Modularidad

Los fragmentos introducen la modularidad y la capacidad de reutilización en la IU de tu actividad, ya que te permiten dividir la IU en fragmentos separados.

Estas nuevas entidades permiten **reutilizar código** y ahorrar tiempo de diseño a la hora de desarrollar una aplicación. Los fragmentos facilitan el despliegue de tus aplicaciones en cualquier tipo de tamaño de pantalla y orientación.

En pantallas más grandes, te recomendamos que la app muestre un panel lateral de navegación estático y una lista en un diseño de cuadrícula.

En pantallas más pequeñas, te recomendamos que la app muestre una barra de navegación inferior y una lista en un diseño lineal.

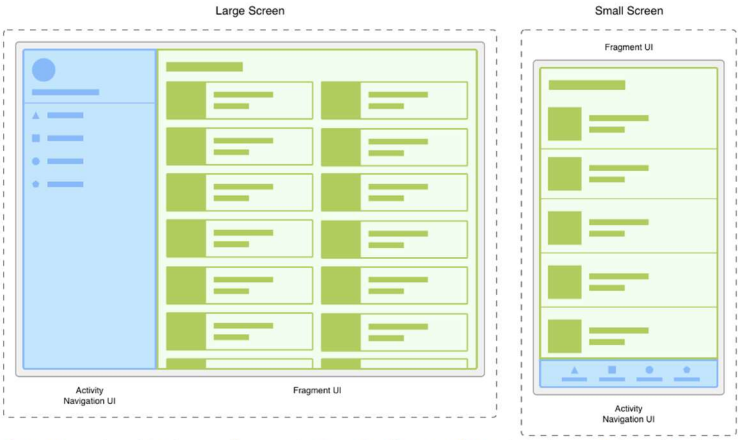
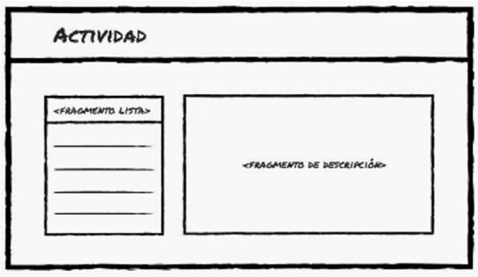


Figura 1: Dos versiones de la misma pantalla en pantallas de tamaños diferentes. En la parte izquierda, una pantalla grande contiene un panel lateral de navegación controlado por la actividad y una lista de cuadrícula controlada por el fragmento. A la derecha, una pantalla pequeña contiene una barra de navegación inferior controlada por la actividad y una lista lineal controlada por el fragmento.

Los Fragmentos permiten crear diseños de **interfaces de usuario de múltiples vistas**. *¿Qué quiere decir eso?*, que los fragmentos son imprescindibles para generar actividades con **diseños dinámicos**, como por ejemplo el uso de pestañas de navegación, etc.



Se creará un fragmento por cada una de las opciones de nuestro panel lateral, y en la actividad pondremos un panel lateral de navegación y un fragmento a la derecha, cuyo contenido iremos cambiando dinámicamente.

Creando un fragmento embebido en una actividad

Los fragmentos son representados en el API de Android por la clase **Fragment**, por lo que cada vez que vayamos a crear un fragmento personalizado debemos crear una nueva clase que herede sus propiedades y comportamientos (**New > Fragment > Fragment (Black)**).

Como veis a cada Fragment se le asocia, una clase de Kotlin y un xml, similar a las Activitys, y ya viene con cierto código precargado.

Al crear los Fragments en la clase Kotlin del fragmento precarga bastante código que podría ser útil en otros usos. Pero en nuestro ejemplo, simplemente tener en cuenta lo siguiente:

a) Hay que modificar el método **onCreateView**() para crear el binding y devolver el View del fragmento correspondiente, y añadir los import o dependencias que reclame.
b) Si quieres añadir eventos a elementos de un Fragment, se deben hacen en **onViewCreated** (no se puede directamente en el onCreateView, porque así lo han diseñado)

Veamos un ejemplo básico (fíjate en las partes marcadas en marrón):

```
class EjemploFragment: Fragment() {

      private lateinit var binding: FragmentEjemploBinding

      override fun onCreateView(
            inflater: LayoutInflater, container: ViewGroup?,
            savedInstanceState: Bundle?
      ): View? {
         binding = FragmentEjemploBinding.inflate(inflater, container,
false)
         val view = binding.root

         ...

         return view
     }
}
```

Vemos como creamos el binding a través del método **inflate**, el cual recibe como parámetros: un objeto de tipo LayoutInflater el cual es proveído por la clase anfitriona, el **contenedor** (en este caso el view de la actividad anfitriona) donde será insertado el fragmento y un booleano indicando si el view que se producirá debe adherirse al contenedor (en nuestro caso usaremos **false** para indicar que no deseamos adherir el view del fragmento).

Si ahora vemos el archivo XML creado, en nuestro caso: fragment_ejemplo.xml, veremos que tiene un componente **TextView** con la cadena «***Hello blank fragment***«:

```
<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="view.EjemploFragment">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/hello_blank_fragment" />

</FrameLayout>
```

Una vez creada la clase y el layout que representa nuestro fragmento debemos **añadirlo a la actividad**.

En el archivo layout de la actividad, incluiremos un componente **XML** llamado **<FragmentCointerView>**, usando el atributo android:name, de esta manera en la actividad se cargará directamente el fragmento indicado.

Veamos un ejemplo del XML de la actividad:

```
<?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:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.fragment.app.FragmentContainerView
    android:id="@+id/miFragmento"
    android:name="view.EjemploFragment"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1" />

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

Creando un fragmento dinámico

Existen casos de fragmentos que no pueden agregarse al layout de una actividad hasta que la aplicación este corriendo, bien sea porque necesitan la información de la base de datos para sus views o porqué todavía el usuario no ha solicitado ver esa parte de la interfaz. En estos casos no es posible embeber un fragmento dentro de la actividad.

Para crear un fragmento dinámicamente en tiempo de ejecución haremos exactamente lo mismo que hicimos anteriormente. Crearemos la clase del fragmento y generamos su layout (**New > Fragment > Fragment (Black)**).

En la clase Kotlin, hay que modificar el método **onCreateView**() para crear el binding y devolver el View del fragmento correspondiente (igual que hicimos en el caso anterior).

Una vez creada la clase y el layout que representa nuestro fragmento, en el layout de la actividad, incluiremos un componente **XML** llamado **<FrameLayout>**. Es importante usar un atributo id para el fragmento, ya que esto permitirá diferenciarlo de otros fragmentos a la hora de hacer la búsqueda de alguno en especial.

También incluiremos un botón a la actividad con el **id** «**btAddFragment**» y el texto «**Añadir Fragmento**«. Su función es añadir el fragmento en tiempo de ejecución, en el componente **FrameLayout** existente de la actividad, una vez se ha presionado el botón.

Veamos un ejemplo del XML de la actividad:

```
<?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:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

   <Button
    android:id="@+id/btAddFragment"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Añadir Fragmento"/>

    <FrameLayout
    android:id="@+id/miFragmento"
    android:layout_width="0dp"
    android:layout_height="wrap_content"></FrameLayout>

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

Antes de nada, es importante entender que un fragmento puede ser añadido, reemplazado y eliminado en tiempo de ejecución. A esas tres operaciones se les llama **Transacciones** y en Kotlin una transacción es representada con la clase **FragmentTransaction**.

Otra situación relevante es que existe una entidad administradora de fragmentos llamada **FragmentManager**. Esta entidad es la que permite coordinar las transacciones de los fragmentos, por lo tanto, cuando reemplacemos nuestro fragmento debemos referirnos a él.

Ahora solo queda añadir las instrucciones necesarias dentro de **OnClickListener** del botón, para añadir nuestro fragmento en el elemento <<miFragmento>> de tipo FrameLayout creado en el XML de la actividad:

Paso 1: Crear nuevo fragmento

Instancia el nuevo fragmento de la clase **EjemploFragment**

```
//Paso 1: Crear un nuevo fragmento y añadirlo
val fragmentoEjemplo = EjemploFragment()
```

Paso 2: Crear la instancia de la transacción

Ahora crearemos una nueva transacción (una instancia de la clase **FragmentTransaction**), usando el método **beginTransaction**() del administrador de fragmentos (*supportFragmentManager*):

```
//Paso 2: Crear una nueva transacción
val fragmentTransaction =supportFragmentManager.beginTransaction()
```

Paso 3: Reemplazar el fragmento en la actividad

Reemplaza el fragmento en el layout de la actividad a través del método **replace**() de la transacción.

```
//Paso 3: Reemplazar fragmento en la actividad
fragmentTransaction.replace(R.id.miFragmento, fragmentoEjemplo)
```

Como ves, el método **replace**() recibe dos parámetros. El primero es el **identificador del contenedor** donde insertaremos el fragmento, en este caso es el FrameLayout de nuestra actividad al cual le asigné el id «**miFragmento**«. El segundo parámetro es la **instancia del fragmento** que queremos añadir.

Paso 4: Confirmar los cambios

Por último ordenamos que los cambios surtan efecto mediante el método **commit**() de la clase **FragmentTransaction**. Si estas familiarizado con **transacciones en SQL**, puedes relacionarlo con la instrucción **COMMIT**, ambas tienen un propósito muy similar.

```
//Paso 4: Confirmar el cambio
fragmentTransaction.commit()
```

El código final de nuestra actividad tendría la siguiente estructura:

```
class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
            insets
        }
        //
        binding.btAddFragment.setOnClickListener {
            val fragmentoEjemplo = EjemploFragment()

            val fragmentTransaction =supportFragmentManager.beginTransaction()
            fragmentTransaction.replace(R.id.miFragmento, fragmentoEjemplo)
            fragmentTransaction.commit()

        }
```

A continuación, vemos un ejemplo de cómo pasar parámetros, en este caso mediante un Bundle. Cuando pasamos datos de un activity a otro, ese bundle va implícito en el intent. En este caso se hace así, directamente creando un Bundle.

```
binding.btF2.setOnClickListener {
    val mBundle = Bundle()
    mBundle.putString("texto",binding.edCaja.text.toString())

    val fragment1 = EjemploFragment()
    fragment1.arguments = mBundle

    val fragmentTransaction = supportFragmentManager.beginTransaction()
    fragmentTransaction.replace(R.id.miFragmento, fragment1)
    fragmentTransaction.commit()
}
```

Estos parámetros se recuperan en método **onCreateView** del fragmento de la siguiente manera:

```
// Recuperar datos del bundle
val bundle = arguments
val message = bundle?.getString("texto")
```

A continuación, vemos un ejemplo de cómo añadir eventos a elementos de un Fragment, desde **onViewCreated:**

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    binding.btnFA.setOnClickListener(){
        val editProfileIntent = Intent(this.context, MainActivity::class.java)
        editProfileIntent.putExtra( name: "ValorFromIntent", value: "Desde Intent")
        startActivity(editProfileIntent)
        Toast.makeText(this.context, text: "Pulsado el botón de dentro del F1", Toast.LENGTH_SHORT).show()
    }
}
```

EJERCICIO: Implementar una navegación de Fragments y **ViewPager2** para crear vistas deslizantes con pestañas, siguiendo el video del siguiente tutorial https://www.youtube.com/watch?v=NHQcWU0emPY y se lo enseñes al profesor.