Apuntes

Descargar estos apuntes

B7 Resumen Fragments

Índice

Fragment 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) y ciclo de vida.

- `onAttach():` . Es invocado cuando el fragmento ha sido asociado a la actividad anfitriona.
- `onActiviyCreated()` . Se ejecuta cuando la actividad anfitriona ya ha terminado la ejecución de su método *onCreate()*.
- `onCreate()` . Este método es llamado cuando el fragmento se está creando. En el puedes inicializar todos los componentes.
- `onCreateView()` . Se llama cuando el fragmento será dibujado por primera vez en la interfaz de usuario. En este método crearemos el view que representa al fragmento para retornarlo hacia la actividad.
- `onStart()` . Se llama cuando el fragmento esta visible ante el usuario. Obviamente depende del método *onStart()* de la actividad.
- `onResume()` . Es ejecutado cuando el fragmento está activo e interactuando con el usuario. Esta situación depende de que la actividad anfitriona este primero en su estado Resume.
- `onStop()` . Se llama cuando un fragmento ya no es visible para el usuario debido a que la actividad anfitriona está detenida o porque dentro de la actividad se está gestionando una operación de fragmentos.
- `onPause()` . Al igual que las actividades, onPause se ejecuta cuando se detecta que el usuario dirigió el foco por fuera del fragmento.
- `onDestroyView()` . Este método es llamado cuando la jerarquía de views a la cual ha sido asociado el fragmento ha sido destruida.
- `onDetach()` . Se llama cuando el fragmento ya no está asociado a la actividad anfitriona.

Subclases de Fragment

- DialogFragment - Muestra un cuadro de dialogo flotante.
- ListFragment - Muestra una lista de elementos.
- PreferenceFragment - Muestra una lista de preferencias.

Crear un Fragment

```
class FragmentUno: Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?): View? {
        return inflater.inflate(R.layout.fragment_uno,container,false)
    }
}
```

Se ha creado un layout asociado a este fragment, o se puede usar uno de Android.

Agregar un Fragment a una Activity

A la hora de agregar un fragmento a una actividad lo podremos realizar de dos maneras:

1. **fragment estático o final** Declarar el fragmento en el layout de la activity.

```
<androidx.fragment.app.FragmentContainerView
    android:id="@+id/fragment_uno"
    android:name="com.ejemplos.b3.ejemplofragmentv1.FragmentUno"
    android:layout_weight="0.5"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"/>
```

2. **fragment dinámico** Agregar directamente el Fragment mediante programación Android. Se podrá eliminar o sustituir por otro fragment u otro contenido.

Agregar Fragment Dinámico

FragmentContainerView

```
    <androidx.fragment.app.FragmentContainerView
        android:name="com.ejemplos.b3.ejemplofragmentv1.FragmentUno"
        android:id="@+id/fragment_uno"
        android:layout_weight="0.5"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"/>
```

Nos creamos una clase para el fragment y un layout para gestionar el aspecto.

```
class MainActivity : AppCompatActivity()
{
    override fun onCreate(savedInstanceState: Bundle?)
    {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val framentManager=supportFragmentManager
        val fragmentTransaction=framentManager.beginTransaction()
        val fragmentDos=FragmentDos()
        fragmentTransaction.add(R.id.fragment_container,fragmentDos)
        fragmentTransaction.commit()
    }
}
```

Binding con Fragments

Gestionar Fragments.

FragmentManager

Cada transacción es un conjunto de cambios que se realizan al mismo tiempo. Podremos realizar dichos cambios a través de los métodos `add()`, `replace()`, `remove()` terminando la transacción con el método `commit()`.
Para añadir la transacción a la pila de retroceso de la activity utilizaremos el método `addToBackStack()` para cada transacción que realicemos.

Comunicar Fragments y Activitys

Comunicación de fragmentos, debemos tener en cuenta las siguientes premisas:

- Los fragmentos no pueden ni deben comunicarse directamente.
- La comunicación entre fragmentos debe hacerse a través de los asociados activity.
- Los fragmentos no necesitan conocer quién es su actividad principal.

Comunicación mediante ViewModel

ViewModel.

Se crea una clase que derive de ViewModel parecida a la siguiente, y teniendo en cuenta que tipo de datos queremos pasar entre los fragments, en este caso un String aunque podría ser un objeto o incluso colecciones de estos:

```
class ItemViewModel : ViewModel() {
        private val liveData=MutableLiveData<String>()
        val getItem: LiveData<String> get() = liveData
        fun setItem(item: String) {
           liveData.value = item
        }
}
```

📌Como podemos ver en el código anterior, aparece un elemento nuevo para exponer los datos que lo hemos llamado liveData de tipo `MutableLiveData` que a su vez extiende de `LiveData` , este elemento es un titular de datos que es capaz de ser observado para enviar solo actualizaciones de datos cuando su observador está activo, puede contener cualquier tipo de datos, y además de eso, es consciente del ciclo de vida para mandar las actualizaciones de datos solamente si el observador está activo.
Para observar un elemento LiveData, tenemos la clase `Observer` , que a través de su objeto podremos saber si está en estado activo (su ciclo de vida está en el estado STARTED o RESUMED) o inactivo (en cualquier otro caso). LiveData solo notifica a los observadores activos sobre las actualizaciones.

```
class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_main)
       val fM: FragmentManager = supportFragmentManager
       val fT: FragmentTransaction = fM.beginTransaction()
       fT.add(R.id.fragment_uno, FragmentPrimario())
       fT.add(R.id.fragment_dos,FragmentSecundario())
       fT.commit() }
}
```

El Fragment primario podría ser como el que sigue, **FragmentPrimario.kt**:

```
class FragmentPrimario : Fragment() {
   private val model:ItemViewModel by activityViewModels()

   override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        saveInstanceState: Bundle?
    ): View?
 {
    val view: View = inflater.inflate(R.layout.fragment_primario,
                                      container,
                                      false)
              view.findViewById<Button>(R.id.boton).setOnClickListener{
              model.setItem(requireActivity().
                    findViewById<EditText>(R.id.texto).text.toString())
              }
      return view
    }
}
```

Para poder incluir el delegado `activityViewModels`, se tiene que añadir la siguiente dependencia (a día de hoy):

```
 implementation 'androidx.fragment:fragment-ktx:1.3.2'
```

El Fragment con el detalle, **FragmentSecundario.kt**:

```
class FragmentSecundario:Fragment() {
    private val model:ItemViewModel by activityViewModels()
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        saveInstanceState: Bundle?): View? {
        val view: View = inflater.inflate
                 (R.layout.fragment_secundario, container, false)
        val nameObserver = Observer<String>{cadena ->
                view.findViewById<TextView>(R.id.texto).text=cadena}
        model.getItem.observe(requireActivity(), nameObserver)
        return view
    }
}
```

ListFragment

```
class MyListFragment: ListFragment() {
    private val model:ItemViewModel by activityViewModels()
    private val valores =
              arrayOf<String>("item1", "item2", "item3", "item4",
                              "item5", "item6", "item7", "item8")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        listAdapter = ArrayAdapter<Any?>(requireActivity(),
                      android.R.layout.simple_list_item_1,
                      valores)
    }
    override fun onListItemClick(l: ListView, v: View,
                                 position: Int, id: Long) {
        super.onListItemClick(l, v, position, id)
        model.setItem(valores[position])
    }
}
```

> 🎓 Otro caso distinto, podemos tenerlo cuando **el fragment se comunica a través de la activity**, mediante el ViewModel.
> La actividad principal con la carga del fragment con la lista y con el observador sobre el ViewModel, quedará así **MainActivity.kt**:

```
    class MainActivity : AppCompatActivity() {
2       //Instanciamos el ViewModel teniendo en cuenta que
        //para la actividad se asocia el delegado viewModels
        private val model: ItemViewModel by viewModels()
        override fun onCreate(savedInstanceState: Bundle?) {
           super.onCreate(savedInstanceState)
            setContentView(R.layout.activity_main)
            val fM: FragmentManager = supportFragmentManager
            var fT: FragmentTransaction = fM.beginTransaction()
10          fT.add(R.id.contenedor_fragment, MyListFragment())
            fT.commit()

            //Construimos un delegado con el observador que se encargará de
            //cargar el FragmentSecundario con la información
            //pasada en un Bundle.
            val nameObserver = Observer<String>{cadena ->
                val bundle=Bundle()
                bundle.putString("DATO",cadena)
19              var fragmentSecundario=FragmentSecundario()
20              fragmentSecundario.arguments=bundle
                fT=fM.beginTransaction()
                fT.add(R.id.contenedor_fragment,fragmentSecundario )
                fT.commit()
                fT.addToBackStack(null)
            }
        //Se pone en observación el ViewModel
        //con el delegado construido anteriormente.
          model.getItem.observe(this, nameObserver)
        }
    }
```

En el **FragmentSecundario.kt** se deberá recuperar el bundle.

```
class FragmentSecundario:Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        saveInstanceState: Bundle?
    ): View? {
        val bundle=arguments
        val view: View = inflater.inflate(
            R.layout.fragment_secundario,
            container, false)
        val texto=bundle?.getString("DATO")
        ...
        return view
    }
}
```

Comunicación mediante Interfaces

La implementación de una interface, es la otra manera correcta para pasar información y controlar algún evento. Se trata de crear una interface en el fragmento y exigir a la activity que la implemente. De esta manera cuando el fragmento reciba un evento también lo hará la activity, que se encargara de recibir los datos de ese evento y compartirlos con otros fragmentos.

Primero crearemos la interface con el método que necesitemos, **PasoCadenaInterface.kt**

```
interface PasoCadenaInterface {
    fun informacionCadena(dato:String)
}
```

La actividad principal **MainActivity.kt**:

```
class MainActivity : AppCompatActivity(),
                     PasoCadenaInterface {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val fM: FragmentManager = supportFragmentManager
        var fT: FragmentTransaction = fM.beginTransaction()
        fT.add(R.id.contenedor_fragment, MyListFragment())
        fT.commit()
    }

    override fun informacionCadena(dato: String) {
        val bundle=Bundle()
        bundle.putString("DATO",dato)
        var fragmentSecundario=FragmentSecundario()
        fragmentSecundario.arguments=bundle
        val  fT=supportFragmentManager.beginTransaction()
        fT.add(R.id.contenedor_fragment,fragmentSecundario )
        fT.commit()
        fT.addToBackStack(null)
    }
}
```

Fragment que pasará el dato, **MyListFragment.kt**:

```
class MyListFragment: ListFragment() {

    lateinit var pasoCadenaInterface:PasoCadenaInterface

    private val valores =
              arrayOf<String>("item1", "item2", "item3", "item4",
                              "item5", "item6", "item7", "item8")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        listAdapter = ArrayAdapter<Any?>(
            requireActivity(),
            android.R.layout.simple_list_item_1,
            valores)
    }

    override fun onListItemClick(l: ListView,
                                 v: View,
                                 position: Int,
                                 id: Long) {
        super.onListItemClick(l, v, position, id)
        pasoCadenaInterface.informacionCadena(valores[position])
    }

    override fun onAttach(context: Context) {
        super.onAttach(context)
        pasoCadenaInterface=context as PasoCadenaInterface
    }
}
```

Información gestión de tareas.

Jetpack Navigation

Navigation

Principios

Para que el usuario pueda entender correctamente el funcionamiento de la app. Podemos encontrar tres elementos fundamentales:

- **Gráfico de navegación**: Es un recurso XML que contiene toda la información relacionada con la navegación.
- **NavHost**: Es un contenedor vacío que muestra los destinos de tu gráfico de navegación. Por defecto está implementado el `NavHostFragment` para fragments y que será el que usemos.
- **NavController**: Es el objeto que administra la navegación de la app dentro de un NavHost.

Para usar estas características deberemos incluir las librerías:

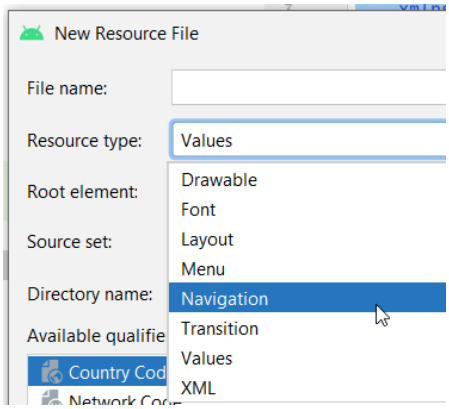
```
implementation 'androidx.navigation:navigation-ui-ktx:2.3.5'
implementation 'androidx.navigation:navigation-fragment-ktx:2.3.5'
```

NavHost

```
    <androidx.fragment.app.FragmentContainerView
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:defaultNavHost="true"
        app:navGraph="@navigation/navegacion_fragments"
        android:id="@+id/contenedor"/>
```

NavGraph

Para crear el `NavGraph` debemos crear un nuevo recurso de tipo navigation ***res->new->android resource file***.



Debemos ver que en la parte del Host se ha añadido el elemento **NavHost** al crear el contenedor de fragmentos enlazado, *por nombre*, con el recurso en la línea `app:navGraph="@navigation/..."`).

NavController

Cada `NavHost` tiene su propio NavController correspondiente.

```
  val navController=
    NavHostFragment.findNavController(this)
if (navController.currentDestination?.id == R.id.fragment1)
    navController.navigate(R.id.action_fragment1_to_fragment2)
```

NavController también permite pasar datos entre fragment mediante bundle, el proceso es similar al usado en otras explicaciones.

```
class MainActivity : AppCompatActivity() {
    lateinit var navHostFragment:NavHostFragment
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        navHostFragment = supportFragmentManager.
           findFragmentById(R.id.contenedor) as NavHostFragment
    }

     fun cancelar()
     {
         val navController = navHostFragment.navController
         navController.navigate(R.id.action_global_fragment1)
     }
}
```

En este ejemplo, los Fragments con el botón **aceptar** que realizará la acción de moverse al siguiente fragment y el botón **cancelar** que lanzará la acción global.

```
class Fragment1: Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view=inflater.inflate(R.layout.fragment1,
                                  container,
                                  false)
        view.findViewByIdMaterialButton(R.id.aceptar).
                 setOnClickListener {
                     aceptar()
        }
        view.findViewByIdMaterialButton(R.id.cancelar).
                 setOnClickListener{
                     (requireActivity() as MainActivity).cancelar()
        }
        return view
    }
    fun aceptar()
    {
     val navController= NavHostFragment.findNavController(this)
     if (navController.currentDestination?.id == R.id.fragment1)
          navController.navigate(R.id.action_fragment1_to_fragment2)
    }
}
```

Permisos con Fragments

Para realizar la petición de permisos al usuario en las clases que heredan de Fragment, registramos la actividad para esta tarea mediante el método `registerForActivityResult`, con un contrato especifico para pedir permisos `ActivityResultContracts.RequestPermission`. Luego usaremos el `ActivityResultLauncher` creado, para lanzar la tarea.

```
    ...
    lateinit  var registerPermisosStorage:ActivityResultLauncher<String>
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        registerPermisosStorage=
            registerForActivityResult(
                    ActivityResultContracts.RequestPermission())
            {
                if(it ==true) tomarGaleria()
            }
    }
    ...
    binding.imagen.setOnLongClickListener {
        registerPermisosStorage.launch(
            Manifest.permission.READ_EXTERNAL_STORAGE)
        true
    }
    ...
```

DialogFragment

DialogFragment

Diálogo de Alerta

```
class DialogoAlerta:DialogFragment(){
    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        super.onCreateDialog(savedInstanceState)
        val builder=MaterialAlertDialogBuilder(requireActivity())
        builder.setMessage("Esto es un mensaje de alerta")
            .setTitle("INFORMACIÓN DE ALERTA")
            .setPositiveButton("Avisado",
                DialogInterface.OnClickListener{
                    dialogo,id->dialogo.cancel()})
        return builder.create()
    }
}
```

Para crearlo y lanzarlo:

```
     val dialogoAlerta=DialogoAlerta()
        dialogoAlerta.show(supportFragmentManager,"DialogoAlerta")
```

Si por algún motivo se necesita pasar información al dialogo, podremos crear un constructor al que le llegue la información mediante Bundle o de otro modo.En el siguiente código se puede ver un posible caso:

```
class DialogoAlerta(bundle: Bundle):DialogFragment(){
    lateinit var bundle: Bundle
    init{
      this.bundle=bundle
    }
    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        super.onCreateDialog(savedInstanceState)
        val builder=MaterialAlertDialogBuilder(requireActivity())
        builder.setMessage(bundle.getString("DATO"))
            .setTitle("INFORMACIÓN")
            .setPositiveButton("Acceptar",
                DialogInterface.OnClickListener{
                    dialogo,id->dialogo.cancel()})
        return builder.create()
    }
}
```

Y en la creación del dialog fragment:

```
 val bundle=Bundle()
        bundle.putString("DATO","AVISO DE PROXIMA CONSULTA MÉDICA")
        val dialogoAlerta=DialogoAlerta(bundle)
        dialogoAlerta.show(supportFragmentManager,"DialogoAlerta")
```

Diálogo de Selección Múltiple

```
class DialogoSeleccionMultiple:DialogFragment() {
    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        super.onCreateDialog(savedInstanceState)
        val builder= MaterialAlertDialogBuilder(requireActivity())
        val datos= arrayOf("Chino","Españo","Frances","Inglés","Turco")
        builder.setTitle("Seleccione idioma")
            .setMultiChoiceItems(datos,null,
            DialogInterface.OnMultiChoiceClickListener{dialogo,item,isChecked->
                Toast.makeText(requireActivity(),"Se ha seleccionado "
                +datos[item],Toast.LENGTH_SHORT).show()
            })
        return builder.create()
    }
}
```

Cada vez que se selecciona un elemento de la lista se ejecutará el escuchador `OnMultiChoiceClickListener`, llegando a este el item pulsado y si esta *checked* o *unchecked*.

Vistas Deslizantes y Tabs

ViewPager

Vistas deslizantes o paginación horizontal

1. El layout de la actividad donde se van a añadir los fragments, deberá de incluir un `ViewPager2`.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.viewpager2.widget.ViewPager2
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/ViewPager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

2. Se deberá crear una clase que manejará la carga de los fragments con el desplazamiento. Esta clase deberá heredar de la clase abstracta `FragmentStateAdapter`, por lo que obligará a implementar el método `createFragment()` a fin de aprovisionar instancias de Fragments a la paginación, y el método `getItemCount()`, que debe devolver el número exacto de fragments que se paginarán.

```
class FragmentPageAdapter(fragment: FragmentActivity):
                            FragmentStateAdapter(fragment)
{
    override fun getItemCount()=3
    override fun createFragment(position: Int): Fragment {
        return when (position) {
            0 -> MyListFragment()
            1 -> FragmentSecundario()
            2 -> FragmentTercero()
            else -> MyListFragment()
        }
    }
}
```

3. Por último se deberá conectar el `FragmentStateAdapter` a los objetos `ViewPager2`.

```
class MainActivity : FragmentActivity()
{
    lateinit var viewPager:ViewPager2
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        viewPager = findViewById(R.id.ViewPager)
        val pagerAdapter = FragmentPageAdapter(this)
        viewPager.adapter = pagerAdapter
    }
    override fun onBackPressed() {
        if (viewPager.currentItem == 0)  super.onBackPressed()
        else viewPager.currentItem = viewPager.currentItem - 1  }
}
```

TabsLayout

TabLayout.

Las pestañas se pueden insertar en la ToolBar (la manera más común) o independientes a esta.

```
    <com.google.android.material.appbar.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.MaterialComponents.Dark.ActionBar">
        <com.google.android.material.appbar.MaterialToolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"/>
        <com.google.android.material.tabs.TabLayout
            android:id="@+id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            style="@style/Widget.MaterialComponents.TabLayout.PrimarySurface"/>
    </com.google.android.material.appbar.AppBarLayout>
    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/viewpager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
```

Las nuevas líneas a añadir en la actividad principal para referenciar al TabLayout creado y para crear un objeto de tipo TabLayoutMediator encargado de relacionar el tabLayout con el ViewPager.

```
val tabLayout = findViewById<TabLayout>(R.id.tabs)
TabLayoutMediator(tabLayout, viewPager) { tab, position ->
             when (position) {
                0 -> tab.setIcon(R.drawable.ic_emoticon)
                1 -> tab.text="Tap 2"
                2 -> tab.setIcon(R.drawable.ic_place)
            }
        }.attach()
```

`TabLayoutMediator` escucha diferentes elementos, para controlar las distintas situaciones:

- `OnPageChangeCallback` de ViewPager2 para ajustar la pestaña cuando ViewPager2 se mueva.
- `OnTabSelectedListener` de TabLayout para ajustar VP2 cuando se mueve la pestaña.
- `AdapterDataObserver` de RecyclerView para recrear el contenido de la pestaña cuando cambia el conjunto de datos.

Un ejemplo del escuchador del ViewPager2 sería:

```
tabLayout.addOnTabSelectedListener (object:TabLayout.OnTabSelectedListener{
    override fun onTabSelected(tab: TabLayout.Tab?) {
            TODO("Not yet implemented")
    }
    override fun onTabUnselected(tab: TabLayout.Tab?) {
            TODO("Not yet implemented")
    }
    override fun onTabReselected(tab: TabLayout.Tab?) {
            TODO("Not yet implemented")
    }})
```