Programación multimedia y dispositivos móviles

Actividad 3.1.3. Adaptación de la app Parques

Francisco José García Cutillas | 2FPGS_DAM

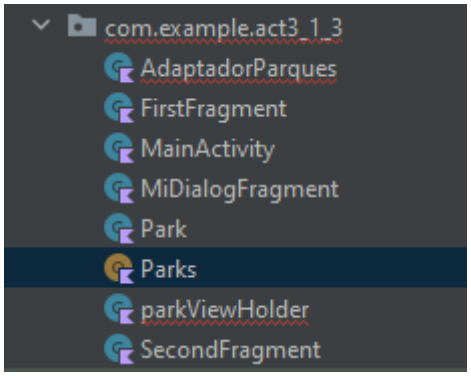
Índice

Ejercicio 1

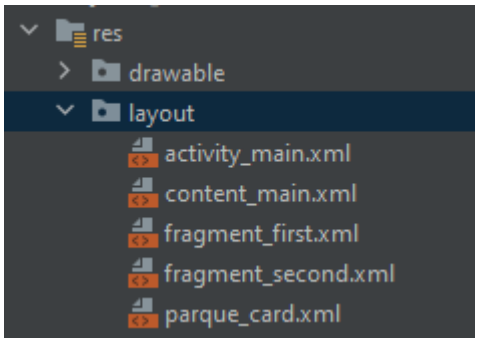
Partiendo de la plantilla de actividad básica y del proyecto sobre parques de la unidad anterior, debes adaptar tu aplicación para que se base en esta plantilla.

Para la realización de esta tarea vamos a ir explicando paso por paso las modificaciones necesarias que debemos realizar en el proyecto parques de la unidad anterior para adaptarlo ahora a una aplicación basada en la plantilla “basic activity”

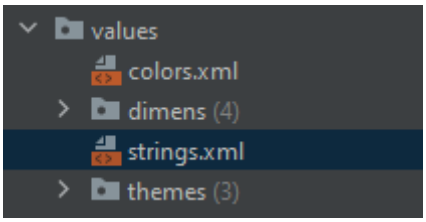
1. Copiamos al directorio com.example.act3_1_3 las clases “AdaptadorParques”, “MiDialogFragment”, “parkViewHolder”, “Park” y el objeto “Parks”.



2. Copiamos el layout “parque_card.xml” en el directorio res -> layout

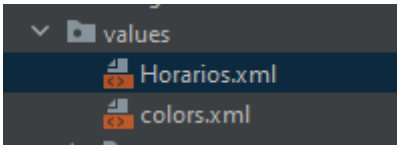


3. En el directorio res -> values, en el fichero “strings.xml”, copiamos el contenido del otro proyecto, y se lo añadimos al de éste. Nos dará un error con la propiedad “name” porque ya existe en este proyecto, por lo que borramos la del proyecto anterior.



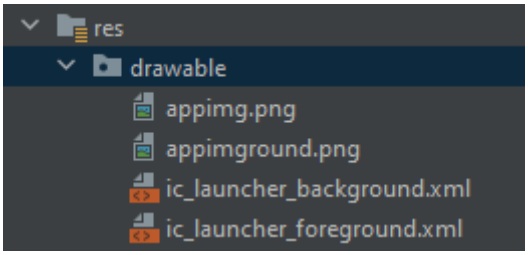

```
<string name="Name">Name</string>
<string name="Description">Description</string>
<string name="PhoneNumber">PhoneNumber</string>
<string name="website">website</string>
<string name="OpeningTime">Opening Time</string>
<string name="ClosingTime">Closing Time</string>
<string name="Activities">Activities</string>
<string name="Sports">Sports</string>
<string name="ChildrenPark">Children\'s Park</string>
<string name="Bar">Bar</string>
<string name="Save">Save</string>
<string name="Pets">Pets</string>
<string name="inicio">Inicio</string>
<string name="add_park">Add Park</string>
<string name="main">Main</string>
<string name="about">About</string>
<string name="SaveData">Save Data</string>
<string name="AskSaveData">Do you want to save data?</string>
<string name="dataSaved">Data has been saved</string>
<string name="actionCancelled">Action has been cancelled</string>
<string name="deletePark">Delete Park</string>
<string name="askDeletePark">"Do you want to delete "</string>
```

4. Dentro de res -> values, creamos un nuevo values resource file con el nombre “Horarios” y en él vamos a copiar los horarios que teníamos en el anterior proyecto.

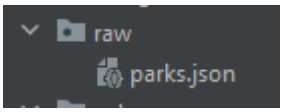


```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="horas">
        <item>24h</item>
        <item>8:00</item>
        <item>8:30</item>
        <item>9:00</item>
        <item>9:30</item>
        <item>10:00</item>
        <item>10:30</item>
        <item>11:00</item>
        <item>11:30</item>
        <item>12:00</item>
        <item>12:30</item>
        <item>13:00</item>
        <item>13:30</item>
        <item>14:00</item>
        <item>14:30</item>
        <item>15:00</item>
        <item>15:30</item>
        <item>16:00</item>
        <item>16:30</item>
        <item>17:00</item>
        <item>17:30</item>
        <item>18:00</item>
        <item>18:30</item>
        <item>19:00</item>
        <item>19:30</item>
        <item>20:00</item>
        <item>20:30</item>
        <item>21:00</item>
        <item>21:30</item>
        <item>22:00</item>
        <item>22:30</item>
    </string-array>
</resources>
```

5. Copiamos en res-> drawable los dos iconos de la aplicación



6. Dentro de res, creamos un directorio de tipo Android resource directory, en el que vamos a copiar el fichero parks.json que es el que contiene los parques de muestra cuando se inicia la aplicación.




```
{
  "parks": [
    {
      "name": "Parc de Capçalera. València.",
      "desc": "Zona verde en el antiguo cauce del Túria, con senderos para caminar o ir en bicicleta.",
      "phone": "963257881",
      "website": "https://jardins.valencia.es/va/jardin/parc-de-capcalera",
      "closingTime": "24h",
      "openingTime": "24h",
      "sports": true,
      "children": true,
      "bar": true,
      "pets": true
    },
    {
      "name": "Parque del Retiro. Madrid",
      "desc": "Jardín histórico y parque público.",
      "phone": "914 00 87 40",
      "website": "https://www.esmadrid.com/informacion-turistica/parque-del-retiro",
      "closingTime": "21:00",
      "openingTime": "18:00",
      "sports": true,
      "children": false,
      "bar": true,
      "pets": true
    },
    {
      "name": "Parc Güell. Barcelona",
      "desc": "Parque público con jardines y elementos arquitectónicos.",
      "phone": "934 09 18 31",
      "website": "https://parkguell.barcelona",
      "closingTime": "19:30",
      "openingTime": "9:30",
      "sports": false,
      "children": false,
      "bar": false,
      "pets": false
    },
    {
      "name": "Parque de María Luisa. Sevilla.",
      "desc": "Parque junto a la Plaza de España, con gran cantidad de plantas, aves y zonas de juego.",
      "phone": "955 47 32 32",
      "website": "https://www.sevilla.org/servicios/medio-ambiente-parques-jardines/parques/parques-y-jardines-historicos/parque-de-maria-luisa",
      "closingTime": "22:00",
      "openingTime": "8:00",
      "sports": true,
      "children": true,
      "bar": true,
      "pets": true
    }
  ]
}
```

Clase MiDialogFragment.kt

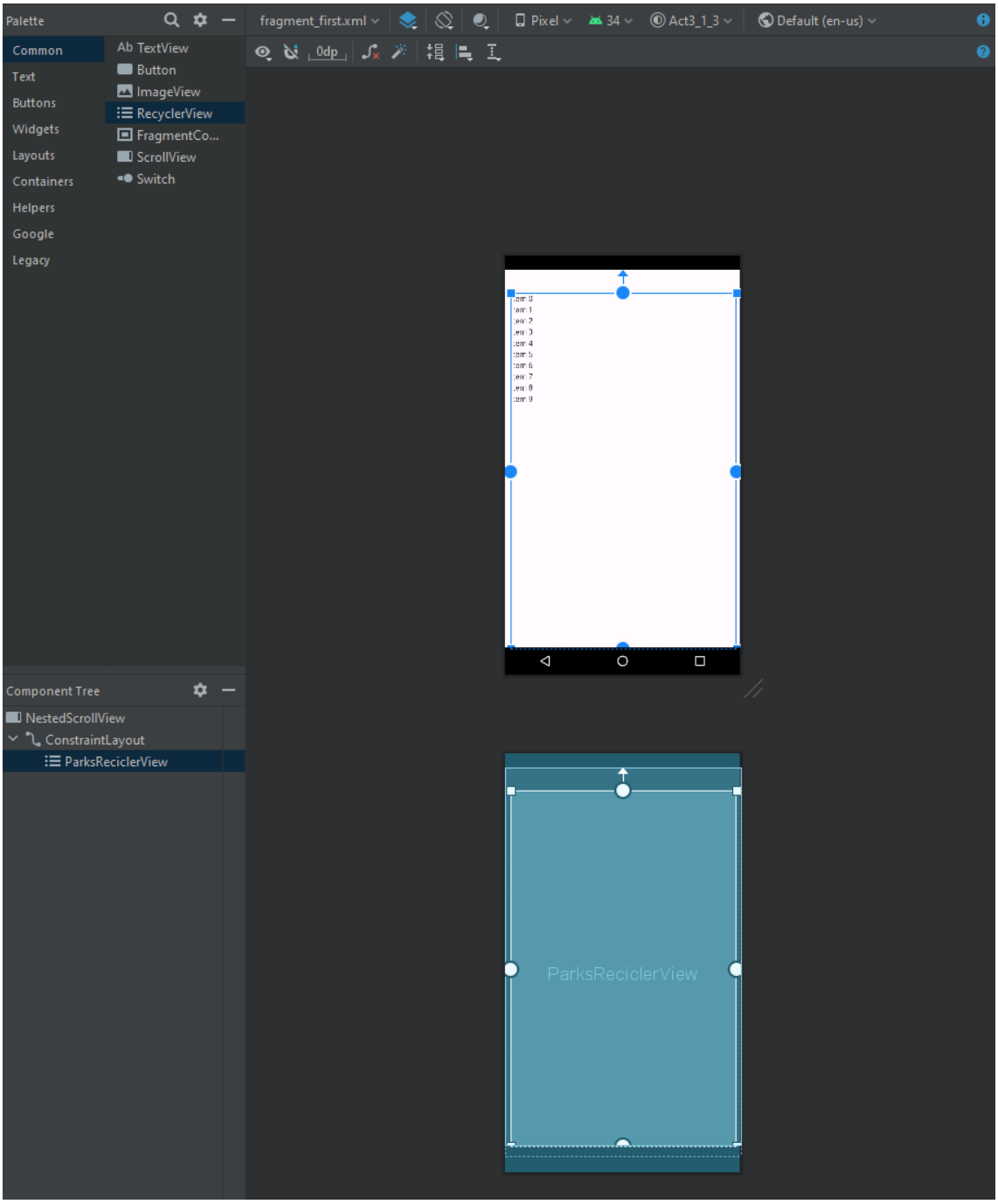
7. En la clase “MiDialogFragment” vamos a añadir al constructor un nuevo parámetro que hemos llamado “fragmento” de tipo Fragment. También hemos modificado los cast de las variables listener de los botones Ok y Cancel, que ya no son activity como antes, sino fragmento.

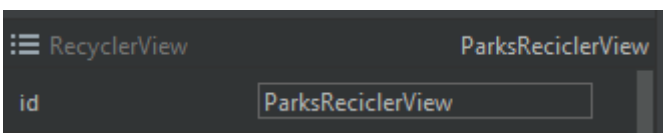
```
class MiDialogFragment(var title: String ="Título por defecto",
                       var content: String ="Contenido por defecto",
                       var fragmento : Fragment) : DialogFragment() {
```

```
builder.setTitle(title).setMessage(content)
    .setPositiveButton(android.R.string.ok) { _, _ ->
        val listener = fragmento as OnOKOrCancelListener?
        listener!!.onPositiveClick()
    }
    .setNegativeButton(android.R.string.cancel) { _, _ ->
        val listener = fragmento as OnOKOrCancelListener?
        listener!!.onCancelClick()
    }
```

Layout fragment_first.xml

8. En el layout “fragment_first.xml” vamos a borrar los componentes que se crean por defecto, y añadimos un componente de tipo ReciclerView, poniéndole de id “ParksReciclerView”.





Modificación de la clase FirstFragment.kt

9. Pasamos a modificar la clase que gestiona este layout que es la clase “FirstFragment”. A esta clase le vamos a añadir que implemente la interfaz MiDialogFragment.OnOKorCancelListener, lo que nos obliga a implementar sus métodos.

```
class FirstFragment : Fragment(), MiDialogFragment.OnOKOrCancelListener {

    override fun onPositiveClick() {
        TODO( reason: "Not yet implemented")
    }

    override fun onCancelClick() {
        TODO( reason: "Not yet implemented")
    }
}
```

10. Añadimos también a la clase “FirstFragment” el método onCreate en el que vamos a meter todos los parques contenidos en el json al reciclerView cada vez que se inicie la actividad. También se ha modificado el primer parámetro del método populate, en el que con requireActivity() nos referimos al contexto de la actividad en la que se encuentra el primer fragmento.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    // Poblamos la lista de parques
    Parks.populate(requireActivity(), R.raw.parks)
```

11. En el método onCreateView vamos a añadir el código necesario para el RecyclerView. Al igual que anteriormente hemos modificado el this que recibía LinearLayoutManager por requireActivity(). Los métodos de clic corto y clic largo se implementan más adelante.

```
// 1. Asociamos el LayoutManager
binding.ParksReciclerView.layoutManager = LinearLayoutManager(requireActivity())

// 2. Determinamos que tiene tamaño fijo (optimización)
binding.ParksReciclerView.setHasFixedSize(true)
// 3. Asignamos el adaptador al RecyclerView, proporcionándole
//    dos funciones lambda, que invocarán los callbacks definidos en esta clase,
//    para gestionar el click y el click largo sobre un item del RecyclerView.
binding.ParksRecyclerView.adapter = AdaptadorParques(
    { park: Park, v: View -> itemClicked(park, v) },
    { park: Park, v: View -> itemLongClicked(park, v) }
)
```

12. Borramos este código que era el generado por defecto.

```
binding.buttonFirst.setOnClickListener {
    findNavController().navigate(R.id.action_FirstFragment_to_SecondFragment)
}
```

13. Insertamos también la variable que nos almacenará temporalmente un parque cuando vayamos a eliminarlo y la inicializamos a null.

```
// Propiedad privada para almacenar temporalmente un item
// se utiliza cuando se va a eliminar un parque
private var itemToRemove:Park?

init{
    itemToRemove=null
}
```

14. Método para el clic corto.

```
private fun itemClicked(park: Park, v: View) {

    val bundle = bundleOf( ...pairs: "park" to park)
    v.findNavController().navigate(R.id.action_FirstFragment_to_SecondFragment, bundle)

}
```

15. Añadimos también el método para el clic largo.

```
private fun itemLongClicked(park: Park, v: View): Boolean {
    // Con el click largo eliminamos el parque de la vista

    // Establecemos el elemento a eliminar
    itemToRemove=park

    // e Invocamos al diálogo para preguntar si deseamos eliminar el registro
    val miDialogo = MiDialogFragment(
        "Delete Park",
        "Do you want to delete "+ park.name +"?",  fragmento: this)
    miDialogo.show(requireActivity().supportFragmentManager,  tag: "miDialogo")
    return true

}
```

16. El código para los métodos para los botones de aceptar y cancelar sería el mismo que en la actividad anterior. Al igual que en casos anteriores el contexto lo proporcionamos con el método requireActivity().

```
override fun onPositiveClick() {
    // Si se pulsa en Aceptar en el diálogo
    // eliminamos el item.
    itemToRemove?.also { it: Park
        val index=Parks.remove(it)

        // Y actualizamos el adaptador del RecyclerView
        // En lugar de DataSetChanged, utilizamos el método notifyItemRemoved
        // indicando la posición eliminada. De este modo, se aplica una
        // animación en el borrado.
        binding.ParksRecyclerView.adapter?.notifyItemRemoved(index)
    }
}

override fun onCancelClick() {
    // Si pulsa cancelar mostramos un toast
    Toast.makeText(
        requireActivity().applicationContext,
        resources.getString(R.string.actionCancelled), Toast.LENGTH_SHORT
    ).show()
}
```

Layout fragment_second.xml

17. En este caso vamos a copiar todo el código xml contenido en el layout de la actividad anterior “activity_edit_park.xml” y modificamos el contexto, que en este caso pasa a ser “SecondFragment”.

```
    app:layout_scrollFlags="scroll"
    tools:context=".SecondFragment">
```

Clase SecondFragment.kt

18. En el método onCreateView vamos a añadir el siguiente código, en el que vamos a rellenar el Spinner y a controlar el evento del botón de guardado.

```
// Poblamos el Spinner de la hora de cierre (CPE1)
// (el de apertura de ha hecho en el XML)
populateSpinner()

// Gestor del evento onClick sobre el botón de guardar
binding.btSave.setOnClickListener { it: View!

    // Preparamos el diálogo de confirmación

    // Hemos añadido los recursos de tipo string saveData y AskSaveData
    // con los valores "Save Data" y "Do you want to save Data".
    val miDialogo = MiDialogFragment(
        resources.getString(R.string.SaveData),
        resources.getString(R.string.AskSaveData),
        fragmento: this)

    miDialogo.show(requireActivity().supportFragmentManager, tag: "miDialogo")

}

return binding.root
```

19. El método populateSpinner() también lo reutilizamos de la actividad anterior, pero en este caso en lugar de this, lo sustituimos por el método requireActivity().

```
// Método especifico para poblar el spinner para la hora de cierre
// (la hora de apertura se ha hecho mediante el XML)
private fun populateSpinner() {
    // Creamos un ArrayAdapter a partir de un recurso de tipo array
    // Requiere tres parámetros: El contexto, el recurso, y el
    // diseño de la entrada (utilizaremos el proporcionado por la
    // propia plataforma.
    ArrayAdapter.createFromResource(
        requireActivity(),
        R.array.horas,
        android.R.layout.simple_spinner_dropdown_item
    ).also { adapter ->
        // Si tenemos el adaptador preparado, seleccionamos el diseño
        // para la lista de opciones (lo proporciona por la plataforma)
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_item)
        // Y finalmente, añadimos el adaptador al spinner
        // Al id del cual accedemos a través del binding
        binding.SpinnerClosingTime.adapter = adapter
    }
```

20. Mediante este código vamos a obtener el parque y establecemos la imagen de fondo.

```
val park : Park? = arguments?.getSerializable( key: "park") as Park?

binding.Image.setImageResource(R.drawable.appimg)
```

21. Comprobamos que el parque no sea nulo, y si no lo es, establecemos a cada campo el valor que tiene para este parque en concreto.

```
park?.also { it: Park


    CurrentPark=it

    binding.Name.setText(it.name)
    binding.Description.setText(it.desc)
    binding.Phone.setText(it.phone)
    binding.webSite.setText(it.webiste)

    for (i in 0 ≤ .. ≤ binding.SpinnerOpeningTime.adapter.count)
        if (binding.SpinnerOpeningTime.adapter.getItem(i).equals(it.openingTime)) {
            binding.SpinnerOpeningTime.setSelection(i)
            break
        }

    for (i in 0 ≤ .. ≤ binding.SpinnerClosingTime.adapter.count)
        if (binding.SpinnerClosingTime.adapter.getItem(i).equals(it.closingTime)) {
            binding.SpinnerClosingTime.setSelection(i)
            break
        }

    if (it.sports?:false) binding.cbSport.isChecked=true
    if (it.children?:false) binding.cbChildren.isChecked=true
    if (it.Bar?:false) binding.cbBar.isChecked=true
    if (it.Pets?:false) binding.cbMascotas.isChecked=true

}
```

22. En el método onViewCreated hemos borrado la funcionalidad del botón que llevaba por defecto.

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

}
```

23. En esta clase también vamos a añadir la variable CurrentPark para guardar temporalmente el parque que se está modificando.

```
// Propiedad para guardar temporalmente el parque que
//  se está editando. Se utiliza cuando se guarda el parque.
private var CurrentPark:Park?

init{
    // Inicialización del parque actual
    CurrentPark=null
}
```

Clase MainActivity.kt

24. Modificamos la acción al pulsar sobre el botón flotante, para que nos lleve del primer fragmento al segundo.

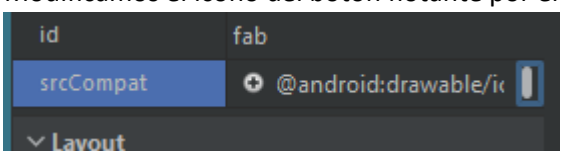
```
binding.fab.setOnClickListener { view ->
    navController.navigate(R.id.action_FirstFragment_to_SecondFragment)
}
```

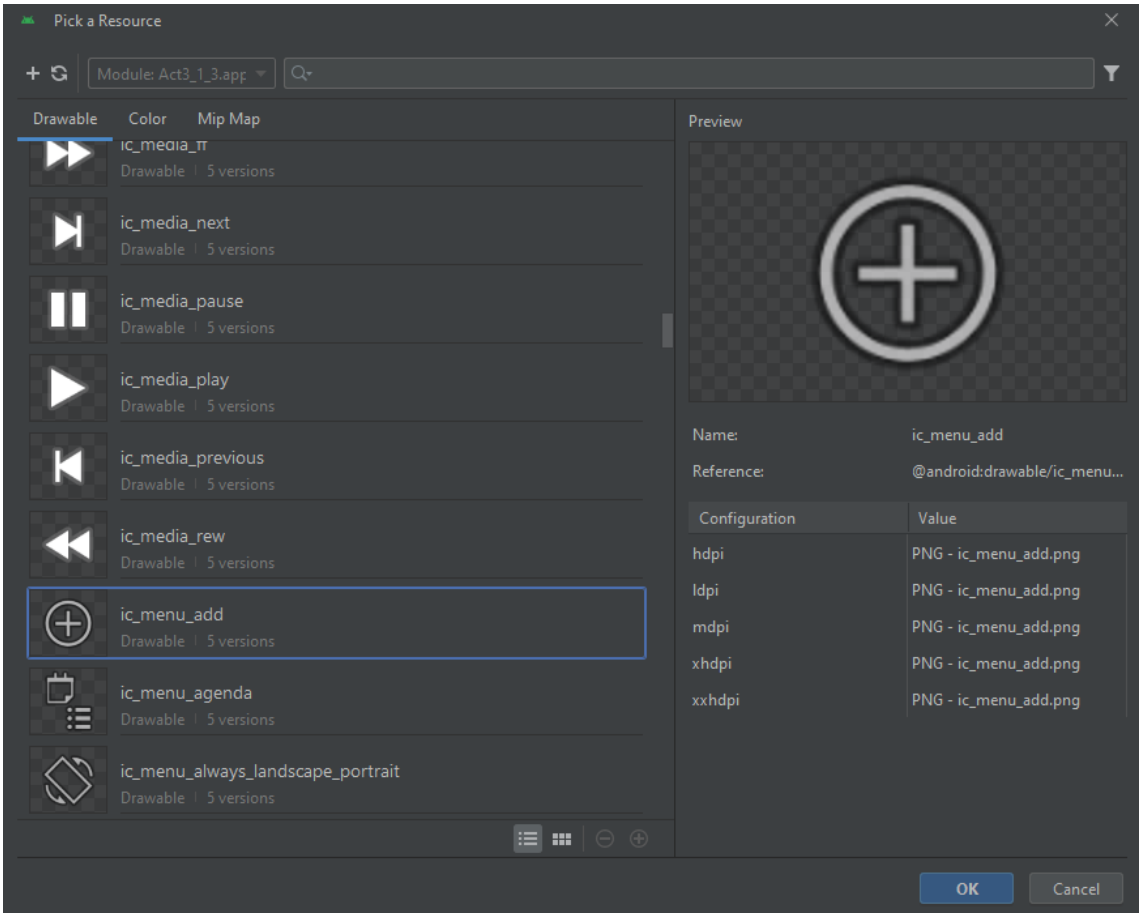
25. Para que en el SecondFragment no se muestre el botón flotante, añadimos:

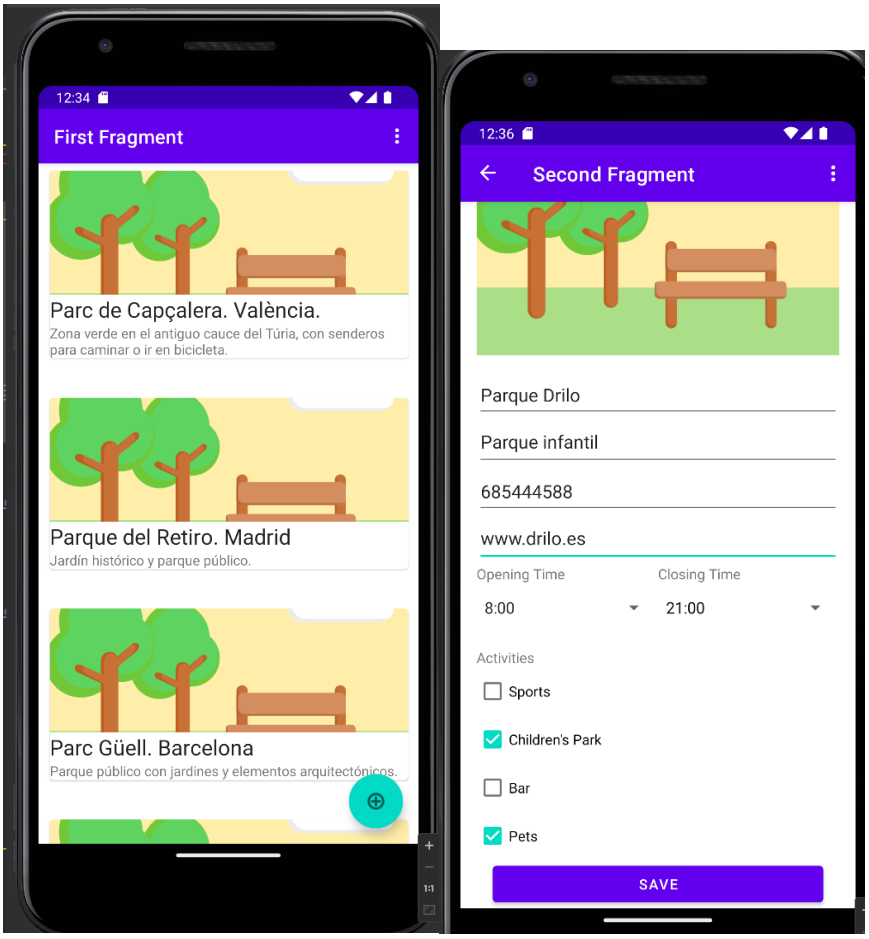
```
navController.addOnDestinationChangedListener{_, destination, _ ->
    when ((destination as FragmentNavigator.Destination).className){

        FirstFragment::class.qualifiedName -> {
            binding.fab.visibility = View.VISIBLE
        }
        else -> {
            binding.fab.visibility = View.GONE
        }
    }
}
```

Layout activity_main.xml

26. Modificamos el icono del botón flotante por el de ic_menu_add.




Pick a Resource
Module: Act3_1_3.app
Drawable
Color
Mip Map
Drawable | 5 versions
ic_media_next
Drawable | 5 versions
ic_media_pause
Drawable | 5 versions
ic_media_play
Drawable | 5 versions
ic_media_previous
Drawable | 5 versions
ic_media_rew
Drawable | 5 versions
ic_menu_add
Drawable | 5 versions
ic_menu_agenda
Drawable | 5 versions
ic_menu_always_landscape_portrait
Drawable | 5 versions
Preview
Name:
ic_menu_add
Reference:
@android:drawable/ic_menu...
Configuration
Value
hdpi
PNG - ic_menu_add.png
ldpi
PNG - ic_menu_add.png
mdpi
PNG - ic_menu_add.png
xhdpi
PNG - ic_menu_add.png
xxhdpi
PNG - ic_menu_add.png
OK
Cancel


12:34
First Fragment
Parc de Capçalera. València.
Zona verde en el antiguo cauce del Túria, con senderos para caminar o ir en bicicleta.
Parque del Retiro. Madrid
Jardín histórico y parque público.
Parc Güell. Barcelona
Parque público con jardines y elementos arquitectónicos.
12:36
Second Fragment
Parque Drilo
Parque infantil
685444588
www.drilo.es
Opening Time
Closing Time
8:00
21:00
Activities
Sports
Children's Park
Bar
Pets
SAVE


12:37
First Fragment
Jardín histórico y parque público.
Parc Güell. Barcelona
Parque público con jardines y elementos arquitectónicos.
Parque de Maria Luisa. Sevilla.
Parque junto a la Plaza de España, con gran cantidad de plantas, aves y zonas de juego.
Parque Drilo
Parque infantil