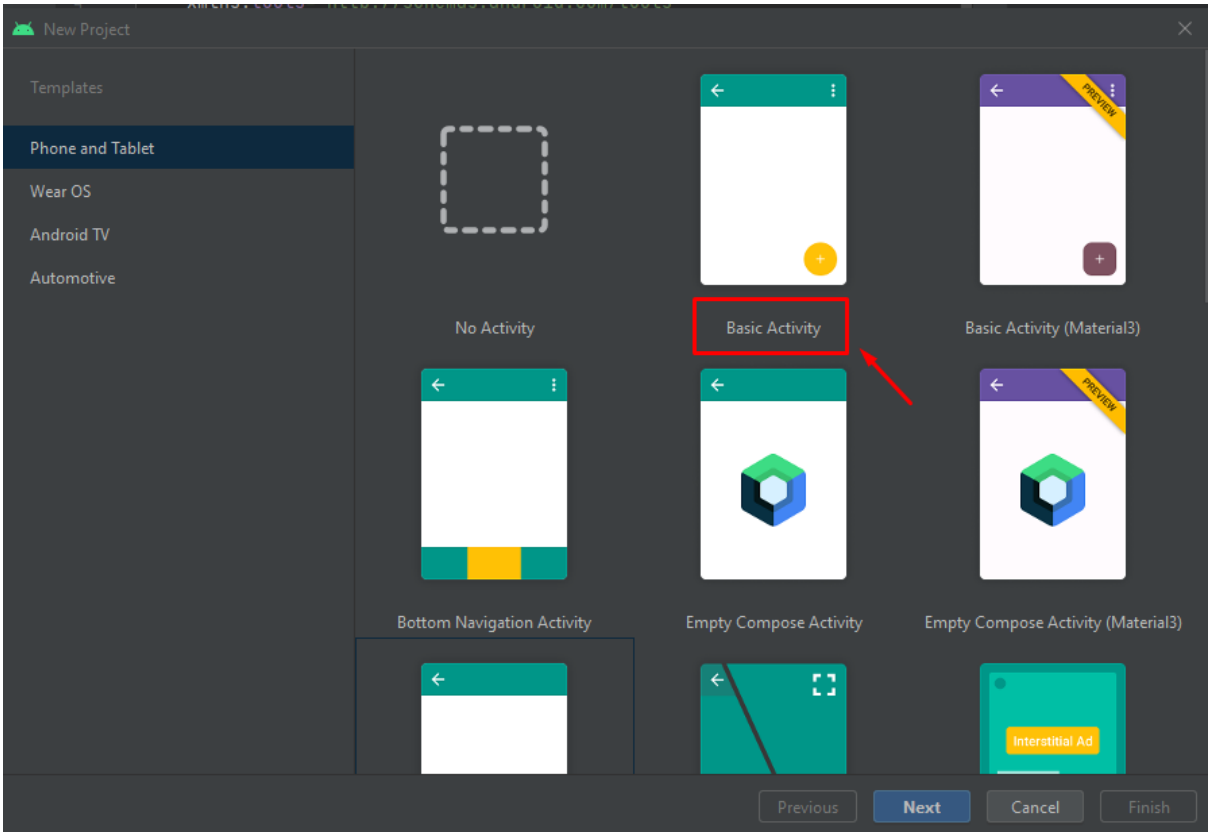
Guía Examen PMDM

Creamos proyecto

Creamos nuestro proyecto seleccionando **Basic Activity**, ya que este ya contiene Navegación y dos fragments, suficiente para lo que queremos hacer y nos pide en el examen. Aprovechamos lo que ya hay creado y borramos lo que nos sobre.



RecyclerView

En el segundo fragment se cargará un recyclerView. Para ello empezamos creando las siguientes clases:

- **DiscoModel**: Clase que solo contiene valores. Se debe colocar el data delante del class.
- **DiscoProvider**: Clase que contiene la información de los discos.
- **DiscoAdapter**: Adaptador para inflar el layout (esto lo dejamos para lo último).

Creamos también el siguiente layout:

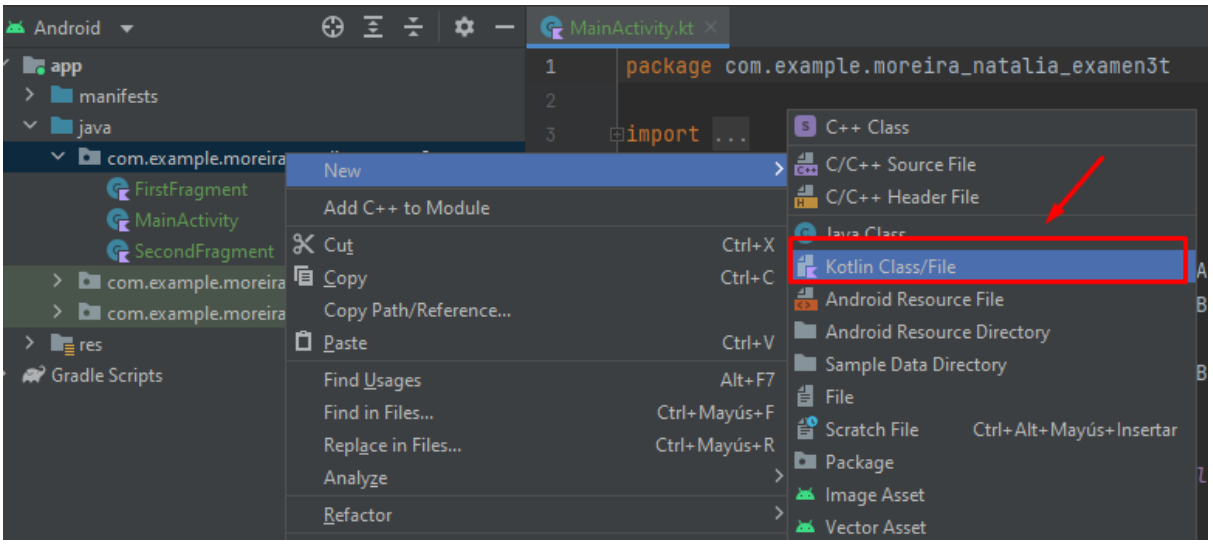
- **disco**.

Creamos la vista:

- **DiscoViewHolder**

Y apovechamos el **SecondFragment.kt** y **fragment_second.xml** que ya vienen creados.

Para crear las clases, hacemos clic derecho encima de nuestra aplicación y elegimos **New/ Kotlin Class/File**



DiscoModel

```
package com.example.moreira_natalia_examen3t.Model

/* Clase que solo contiene valores, por eso el data delante de class
* https://www.develou.com/data-classes-en-kotlin/
*/

data class DiscoModel (
    val foto: Int?,
    val titulo: String,
    val artista: String,
    val id: Int
    )
```

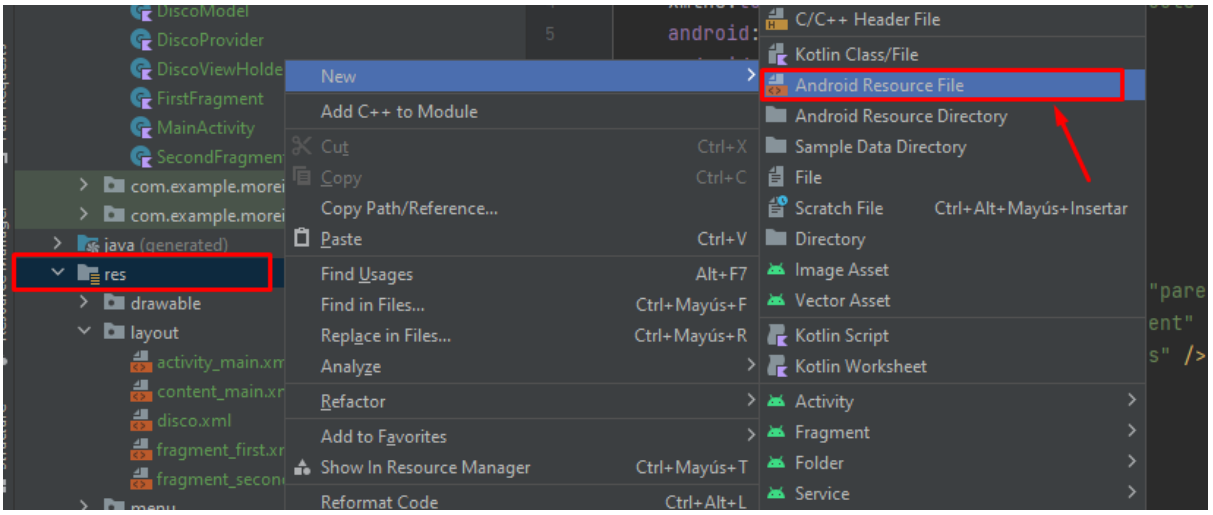
DiscoProvider

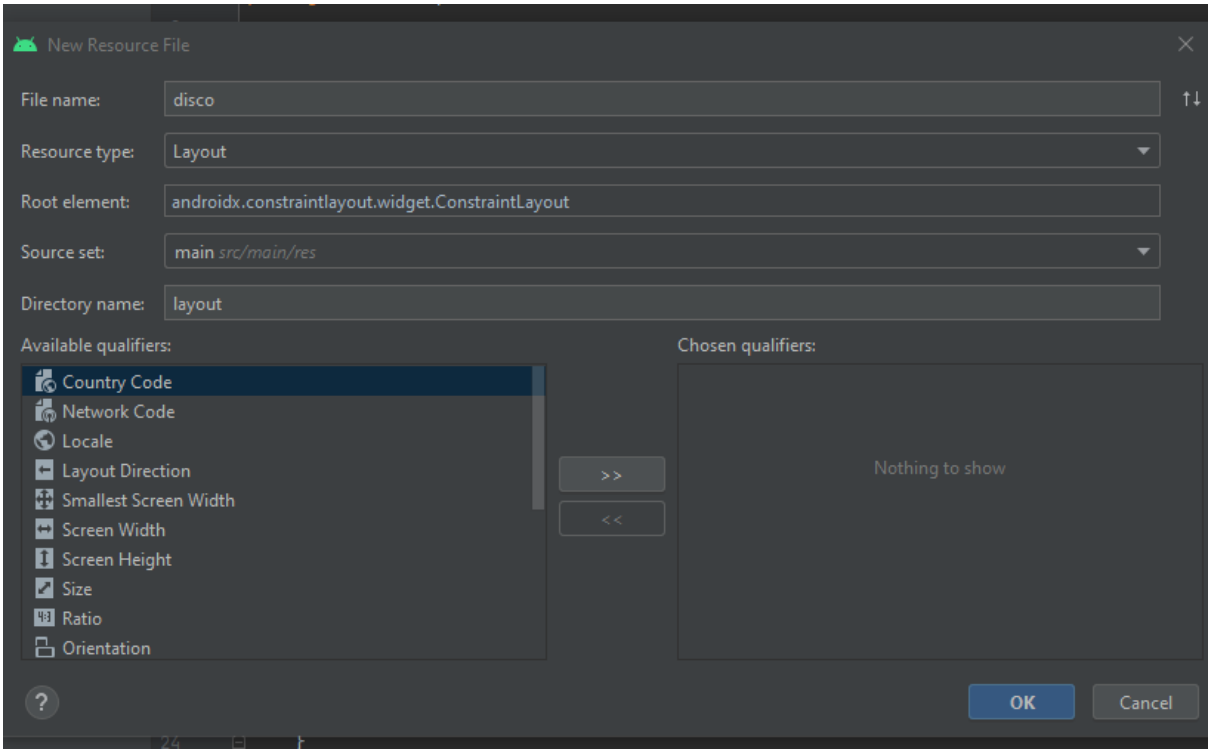
```
class DiscoProvider {
    companion object{
        val listaDisco : MutableList<DiscoModel> = mutableListOf(
            DiscoModel(
                foto = R.drawable.abbeyroad,
                titulo = "AbbeyRoad",
                artista = "The Beatles",
                id = 0
            ), DiscoModel(
                foto = R.drawable.exileonmainst,
                titulo = "Exile on Main Street",
                artista = "The Rolling Stones",
                id = 1
            ),
            DiscoModel(
                foto = R.drawable.velvetunderground,
                titulo = "The Velvet Underground",
                artista = "The Velvet Underground and Nico",
                id = 2
            ),
            DiscoModel(
                foto = R.drawable.areyouexperienced,
                titulo = "Are You Experienced",
                artista = "Jimi Hendrix",
                id = 3
            ),
            DiscoModel(
                foto = R.drawable.backinblack,
                titulo = "Back in Black",
                artista = "AC/DC",
                id = 4
            ), DiscoModel(
                foto = R.drawable.appetitefordestruction,
                titulo = "Appetite for Destruction",
                artista = "TGuns N' Roses",
                id = 5
            ),
```

NOTA: las cosas que nos salgan en rojo al crear las clases, ni caso, es porque nos falta por crear cosas. Después al tener todo se importan.

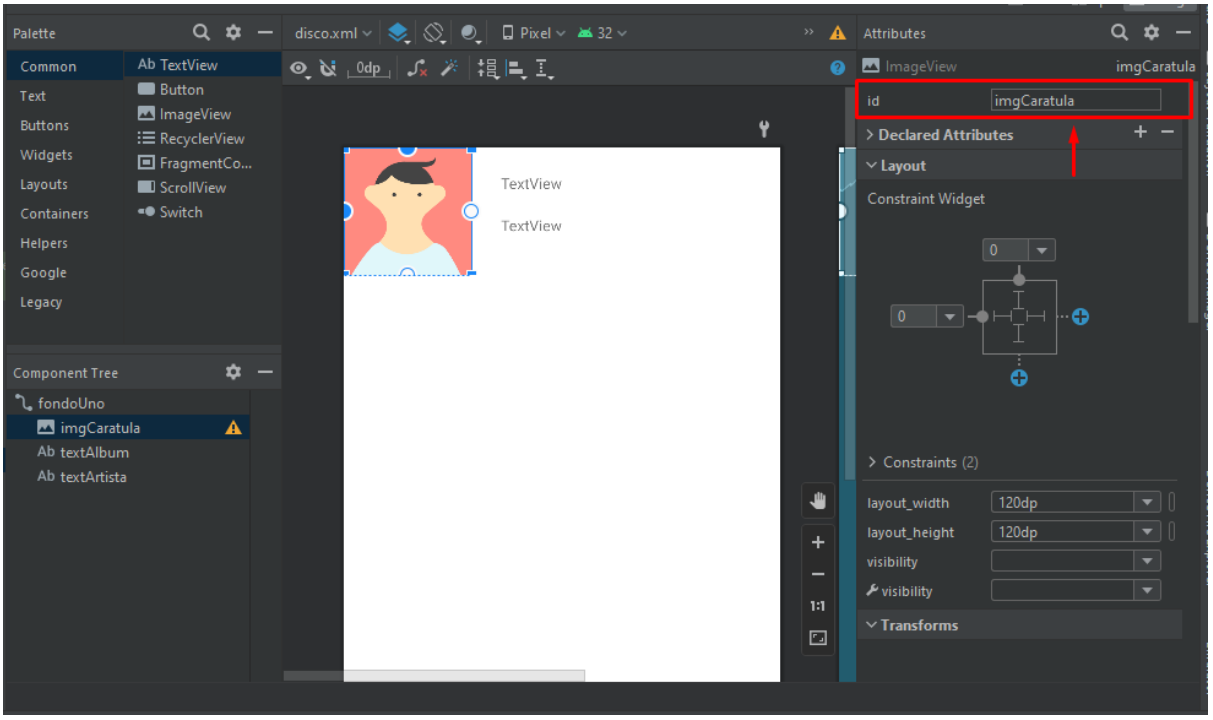
Layout disco

Clic derecho en **res** y elegimos **New/ Android Resource File**





y ahí creamos nuestro diseño.
Añadimos una imagen y dos textView, importante cambiar el id.

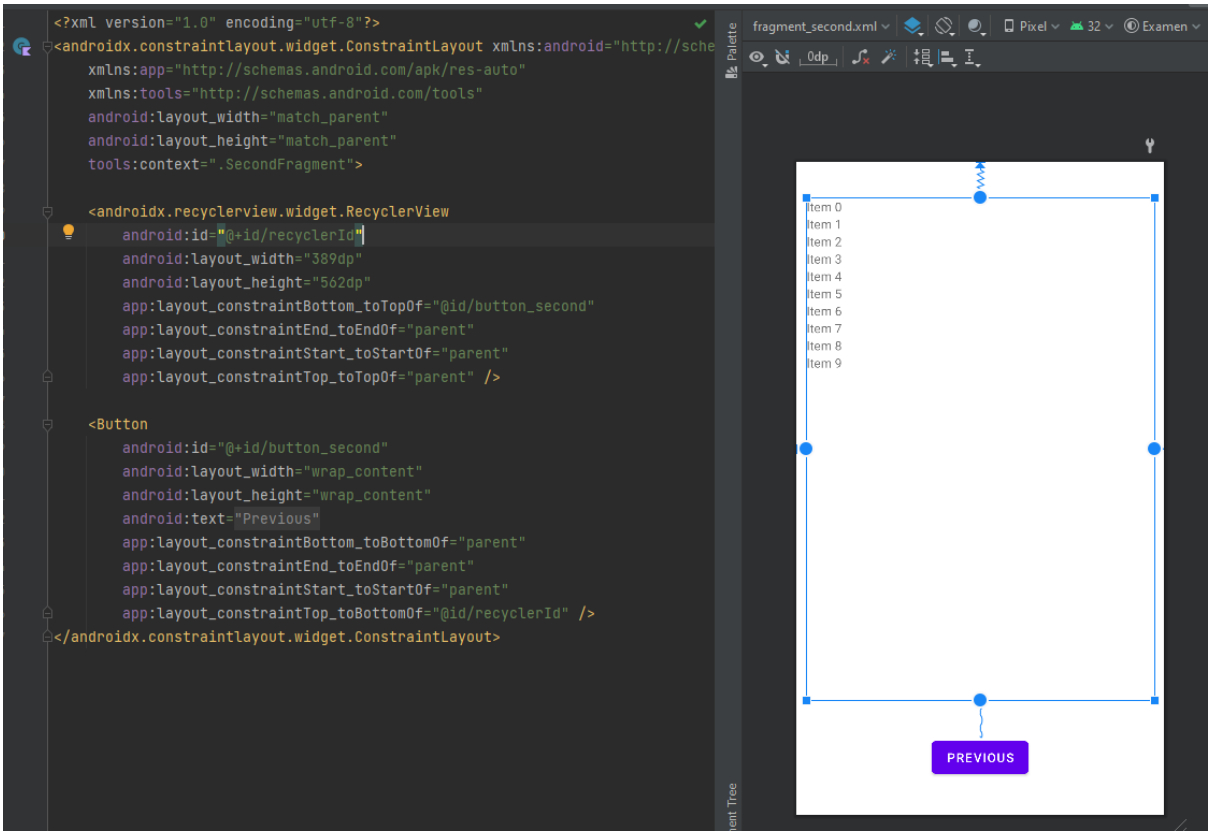


También añadimos un id para cambiar color de fondo e importante poner wrap_content en el height.
En ImageView ponemos una anchura y altura de 120dp para adaptar la imagen.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:an
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/fondoUno"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <ImageView
        android:id="@+id/imgCaratula"
        android:layout_width="120dp"
        android:layout_height="120dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:srcCompat="@tools:sample/avatars" />
```

Fragment_second.xml

Quitamos el textview y añadimos el recyclerView, no olvidar de ponerle un id.




```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://sche
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecondFragment">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerId"
        android:layout_width="389dp"
        android:layout_height="562dp"
        app:layout_constraintBottom_toTopOf="@id/button_second"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button_second"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Previous"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/recyclerId" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

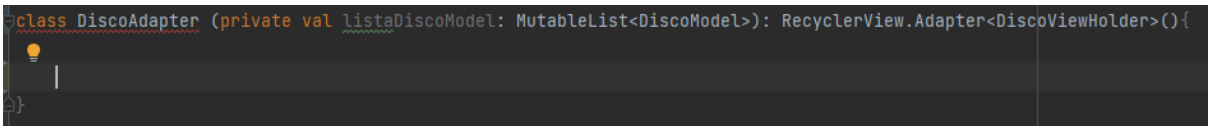
DiscoViewHolder

Creamos la vista (nueva clase)

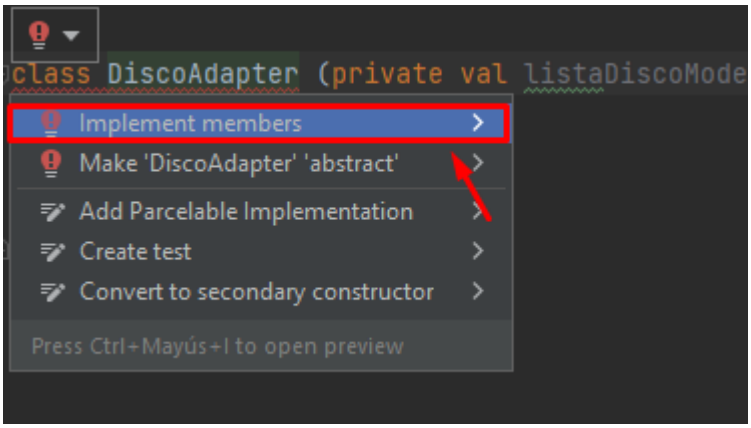
```kotlin
package com.example.moreira_natalia_examen3t

import android.view.View
import androidx.recyclerview.widget.RecyclerView
import com.example.moreira_natalia_examen3t.databinding.DiscoBinding

/* Vista */
class DiscoViewHolder (view: View): RecyclerView.ViewHolder(view) {
    val binding = DiscoBinding.bind(view)

    fun render(modelo: DiscoModel){
        // Seleccionamos cada información del disco del modelo (DiscoModel)
        binding.imgCaratula.setImageResource(modelo.foto ?: R.drawable.ic_launcher_background)
        binding.textAlbum.text = modelo.titulo
        binding.textArtista.text = modelo.artista
    }
}
```

DiscoAdapter

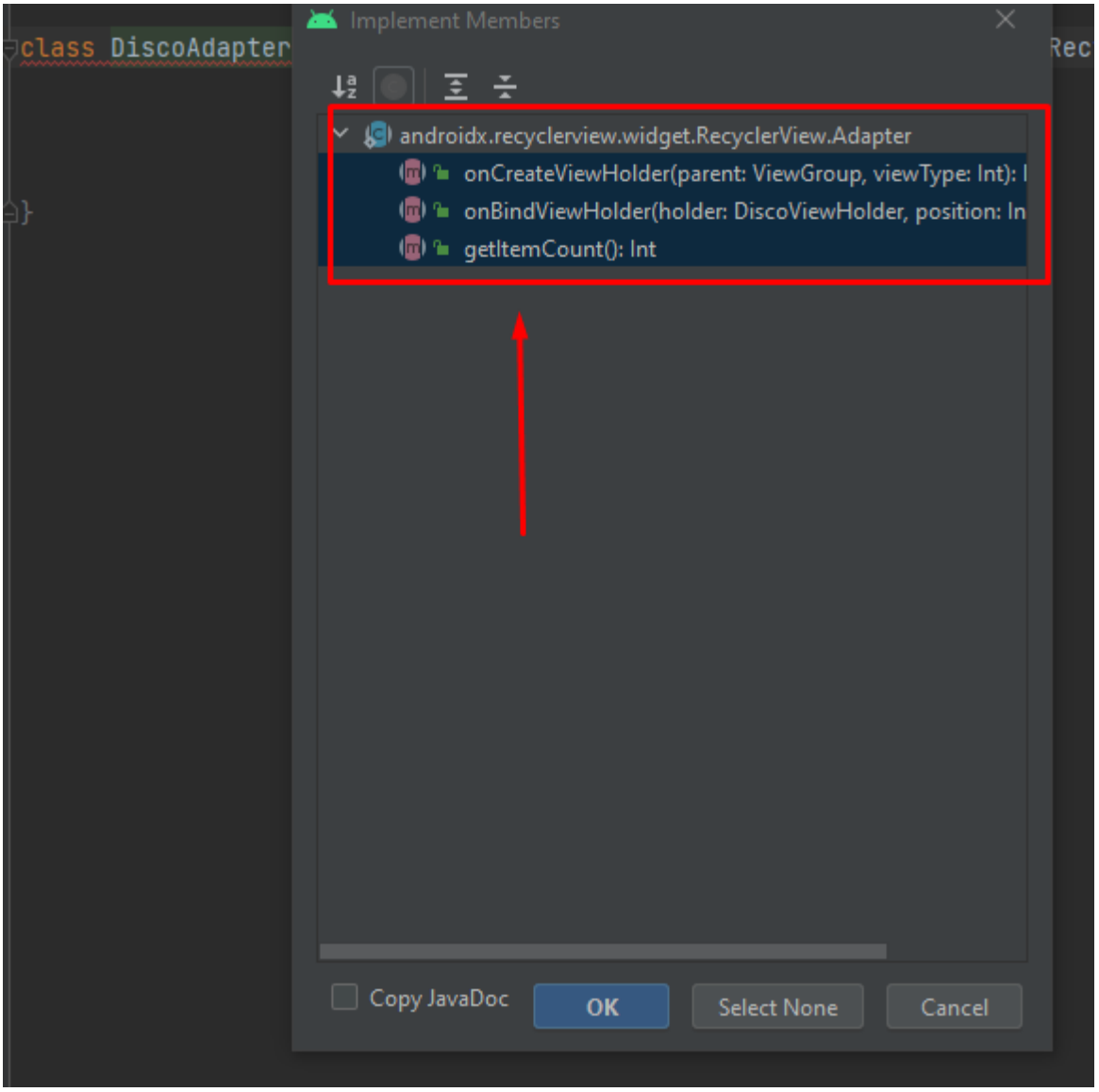
Aquí debemos poner lo siguiente:



Nos ponemos encima de DiscoAdapter y nos saldrá la bombilla roja que nos indica que debemos crear 3 métodos.



Seleccionamos los tres e implementamos.



Ahora agregamos los siguiente en cada método.

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): DiscoViewHolder {
    val layoutInflater = LayoutInflater.from(parent.context)
    return DiscoViewHolder(layoutInflater.inflate(R.layout.disco, parent, attachToRoot: false))
}
```

```
override fun onBindViewHolder(holder: DiscoViewHolder, position: Int) {
    val item = listaDiscoModel[position]
    holder.render(item)

    // esto es para cambiar el color del fondo de cada caja de información del disco
    with(holder){ this: DiscoViewHolder
        if(position % 2 == 0){
            binding.fondo.setBackgroundColor(Color.CYAN)
        }else{
            binding.fondo.setBackgroundColor(Color.BLUE)
        }
    }

    // botón eliminar
    holder.binding.btnEliminar.setOnClickListener { it: View!
        listaDiscoModel.removeAt(position)
        // notificamos la eliminación del elemento
        notifyItemRemoved(position)
        notifyItemRangeChanged(position, listaDiscoModel.size)
    }
}
```

```
    // Reducimos este método
    override fun getItemCount(): Int = listaDiscoModel.size
}
```

Nos quedará así la clase entera

```kotlin
package com.example.moreira_natalia_examen3t.ViewModel

import ...

/* Aquí implementamos los 3 métodos que nos pide */
class DiscoAdapter (private val listaDiscoModel: MutableList<DiscoModel>): RecyclerView.Adapter<DiscoViewHolder>(){

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): DiscoViewHolder {
        val layoutInflater = LayoutInflater.from(parent.context)
        return DiscoViewHolder(layoutInflater.inflate(R.layout.disco, parent, attachToRoot: false))
    }

    override fun onBindViewHolder(holder: DiscoViewHolder, position: Int) {
        val item = listaDiscoModel[position]
        holder.render(item)

        // esto es para cambiar el color del fondo de cada caja de información del disco
        with(holder){ this: DiscoViewHolder
            if(position % 2 == 0){
                binding.fondo.setBackgroundColor(Color.CYAN)
            }else{
                binding.fondo.setBackgroundColor(Color.BLUE)
            }
        }

        // botón eliminar
        holder.binding.btnEliminar.setOnClickListener { it: View!
            listaDiscoModel.removeAt(position)
            // notificamos la eliminación del elemento
            notifyItemRemoved(position)
            notifyItemRangeChanged(position, listaDiscoModel.size)
        }
    }

    // Reducimos este método
    override fun getItemCount(): Int = listaDiscoModel.size
}
```

SecondFragment.kt

En este fragment agregamos las siguientes líneas dentro del método **onViewCreated**

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

    binding.buttonSecond.setOnClickListener { it: View!
        findNavController().navigate(R.id.action_SecondFragment_to_FirstFragment)
    }
    // Añadimos lo siguiente para llamar nuestro layout y adapter
    val recyclerView = view?.findViewById<RecyclerView>(R.id.recyclerId)
    recyclerView?.layoutManager = LinearLayoutManager(context)
    recyclerView?.adapter = DiscoAdapter(DiscoProvider.listaDisco)
}
```

Spinner

En el primer fragment hay un spinner, para ello crearemos una nueva clase:

- **SpinnerProvider**: Clase que contiene un array de strings.

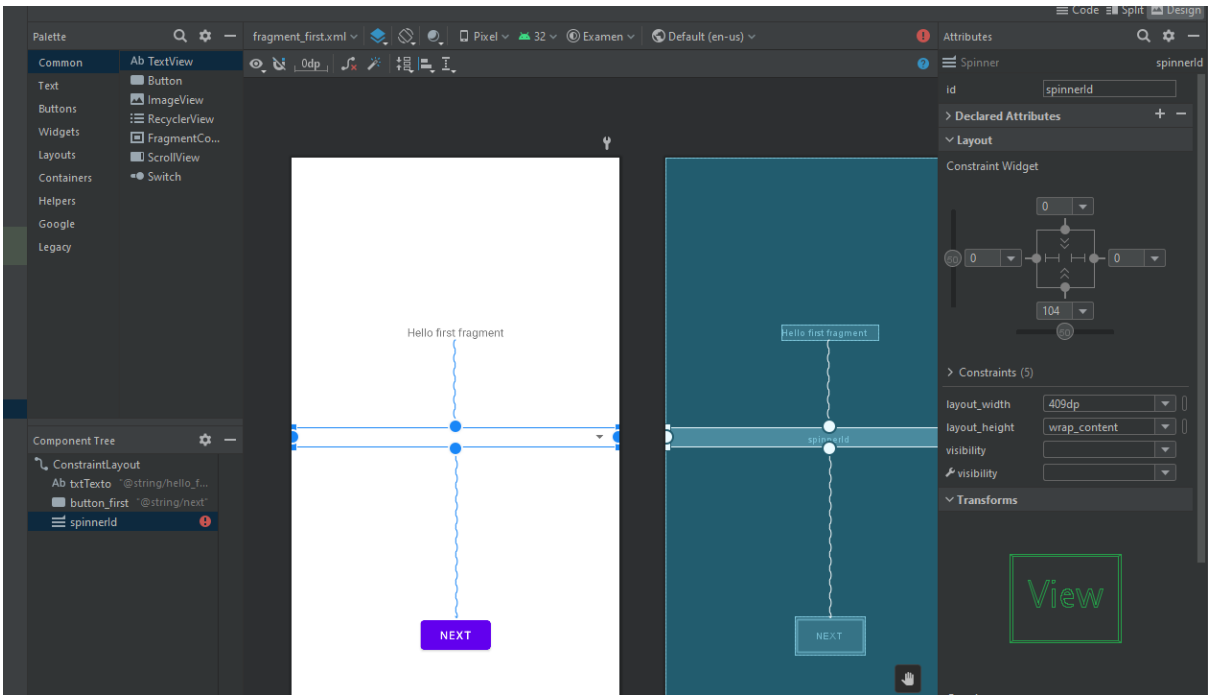
y modificaremos **FirstFragment.kt** y **fragment_first.xml**

SpinnerProvider

```kotlin
package com.example.moreira_natalia_examen3t

/* Clase que contiene un array de strings */
class SpinnerProvider {
    companion object {
        val listaEstilos = listOf<String>(
            "Género",
            "Rock",
            "Blues",
            "Jazz",
            "Varios"
        )
    }
}
```

fragment_first

Aquí creamos nuestro diseño, agregando un spinner (no olvidar colocarle un id)



First_Fragment

Creamos funciones para cargar spinner y funcionalidades del spinner

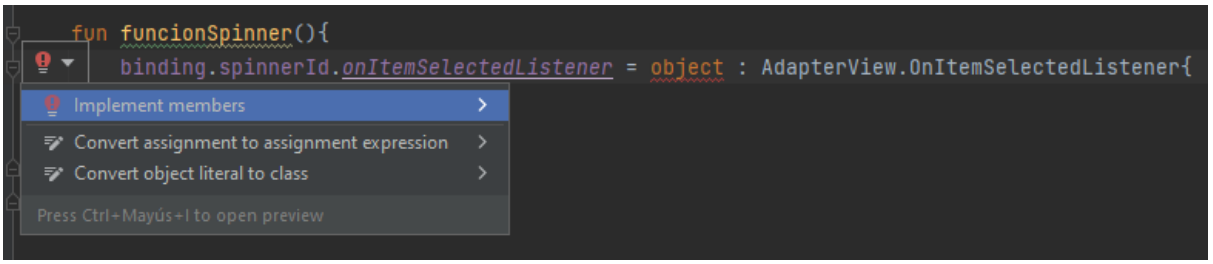
Primero creamos función para cargar el spinner debajo del **onViewCreated**

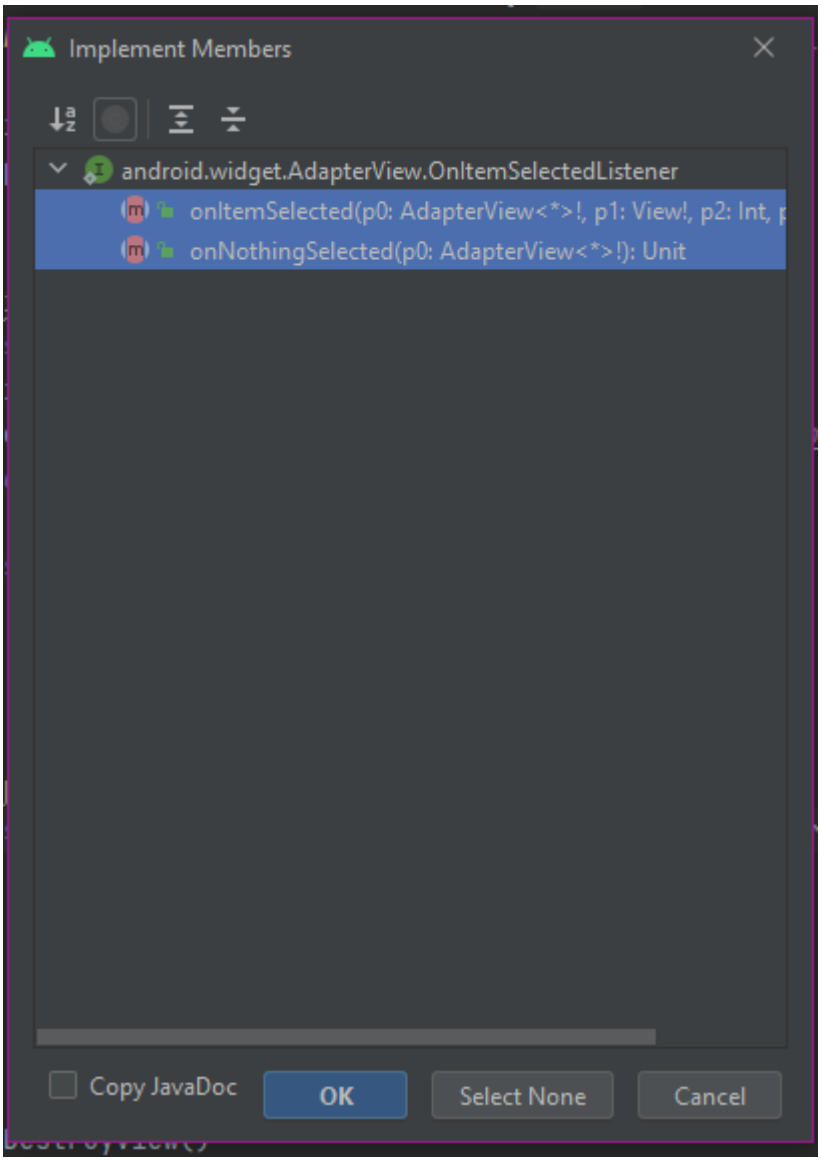
```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    binding.buttonFirst.setOnClickListener {  it: View!
        findNavController().navigate(R.id.action_FirstFragment_to_SecondFragment)
    }
    //llamamos las funciones de abajo
    cargarSpinner()
    funcionSpinner()
}

// función para cargar el spinner con la lista de géneros que creamos en SpinnerProvider
fun cargarSpinner(){
    binding.spinnerId.adapter = ArrayAdapter<String>(
        requireContext(),
        androidx.appcompat.R.layout.support_simple_spinner_dropdown_item,
        listaEstilos
    )
    binding.spinnerId.setSelection( position: 0,  animate: false)
}
```

Luego creamos función para agregar funcionalidades al spinner. Primero escribimos la siguiente línea:

```
fun funcionSpinner(){
    binding.spinnerId.onItemSelectedListener = object : AdapterView.OnItemSelectedListener{


    }
}
```

Nos ponemos encima de object y nos pedirá implementar dos métodos, los seleccionamos e implementamos





Agregamos lo siguiente en el método **onItemSelected**

```
override fun onItemSelected(p0: AdapterView<*>?, p1: View?, p2: Int, p3: Long) {
    // esto muestra el nombre
    binding.txtTexto.setText(binding.spinnerId.selectedItem.toString())
    // esto muestra la posición en el array
    //binding.txtTexto.setText(binding.spinnerId.selectedItemPosition.toString())
    // esto muestra el id
    //binding.txtTexto.setText(binding.spinnerId.selectedItemId.toString())

    // mostramos un toast al seleccionar un género
    Toast.makeText(
        binding.root.context,
        binding.spinnerId.selectedItem.toString(),
        Toast.LENGTH_SHORT).show()

    // mostramos un snackbar
    //Snackbar.make(binding.root, "Mensaje", Snackbar.LENGTH_LONG).show()

    // mostramos una alerta
    //AlertDialog.Builder(binding.root.context).setMessage("Mensaje").setTitle("Título").create().show()

}
```

El otro método lo dejamos vacío.

Nos quedaría así

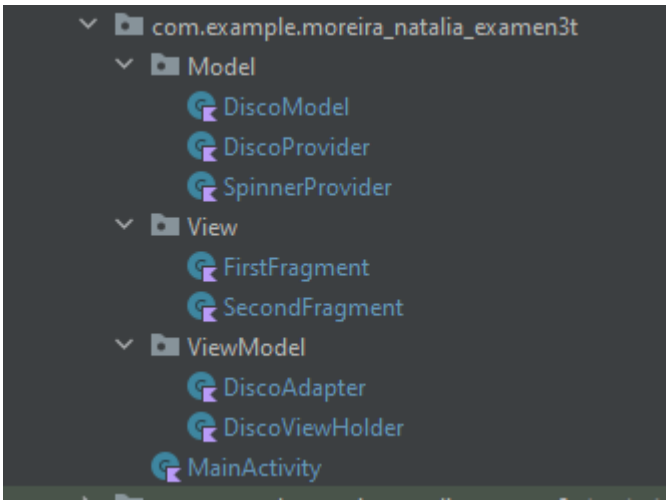
```
// función para agregar funcionalidades al spinner
fun funcionSpinner(){
    binding.spinnerId.onItemSelectedListener = object : AdapterView.OnItemSelectedListener{
        override fun onItemSelected(p0: AdapterView<*>?, p1: View?, p2: Int, p3: Long) {
            // esto muestra el nombre
            binding.txtTexto.setText(binding.spinnerId.selectedItem.toString())
            // esto muestra la posición en el array
            //binding.txtTexto.setText(binding.spinnerId.selectedItemPosition.toString())
            // esto muestra el id
            //binding.txtTexto.setText(binding.spinnerId.selectedItemId.toString())

            // mostramos un toast al seleccionar un género
            Toast.makeText(
                binding.root.context,
                binding.spinnerId.selectedItem.toString(),
                Toast.LENGTH_SHORT).show()

            // mostramos un snackbar
            //Snackbar.make(binding.root, "Mensaje", Snackbar.LENGTH_LONG).show()

            // mostramos una alerta
            //AlertDialog.Builder(binding.root.context).setMessage("Mensaje").setTitle("Título").create().show()

        }
        // esto no sabemos que hace pero se deja vacío. Alejandro tampoco sabe
        override fun onNothingSelected(p0: AdapterView<*>?) {

        }
    }
}
```

No olvidar llamar a las funciones que creamos en **onViewCreated**

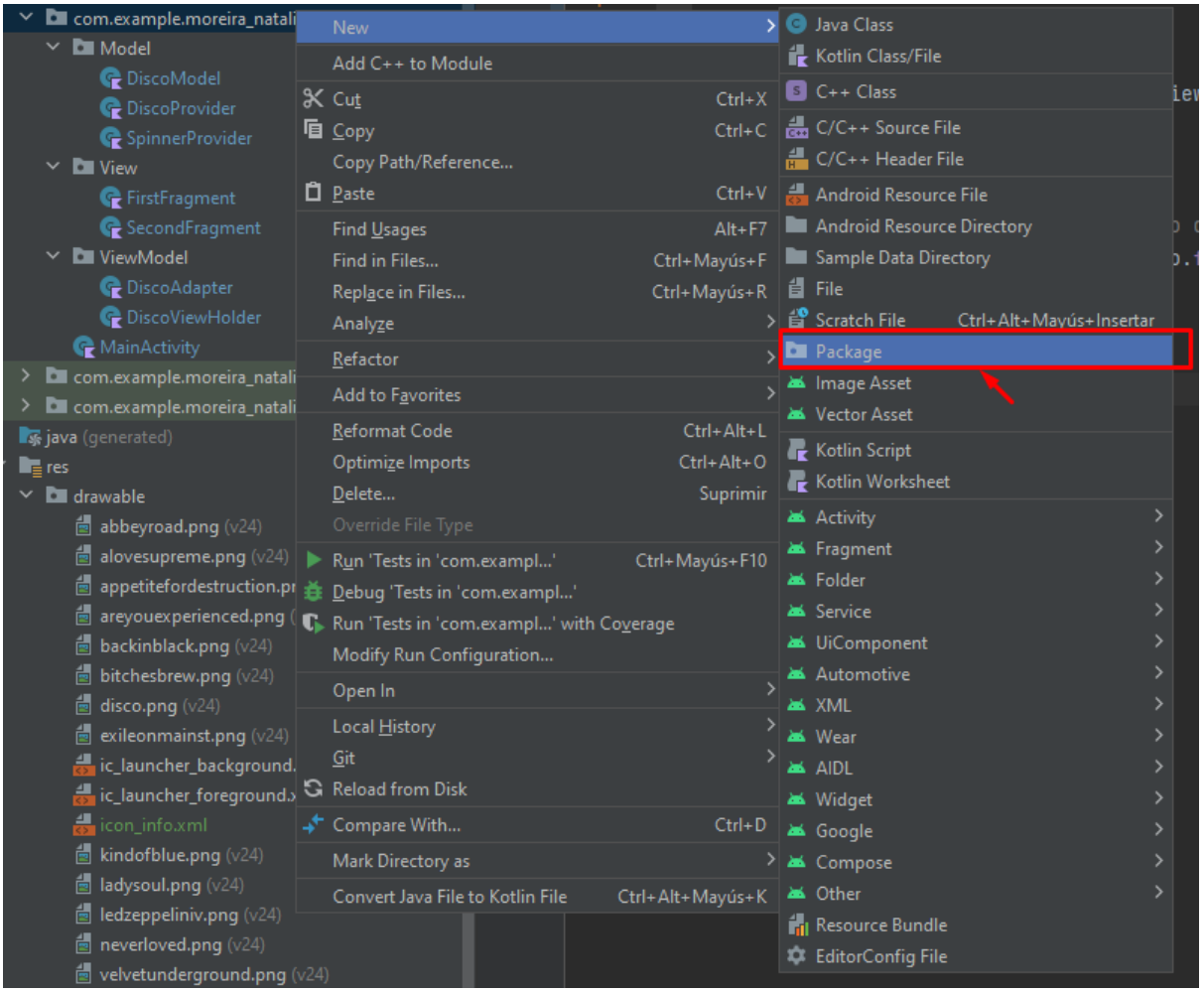
```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    binding.buttonFirst.setOnClickListener {  it: View!
        findNavController().navigate(R.id.action_FirstFragment_to_SecondFragment)
    }
    //llamamos las funciones de abajo
    cargarSpinner()
    funcionSpinner()
}
```

Organización carpetas

Organizamos nuestro código así:



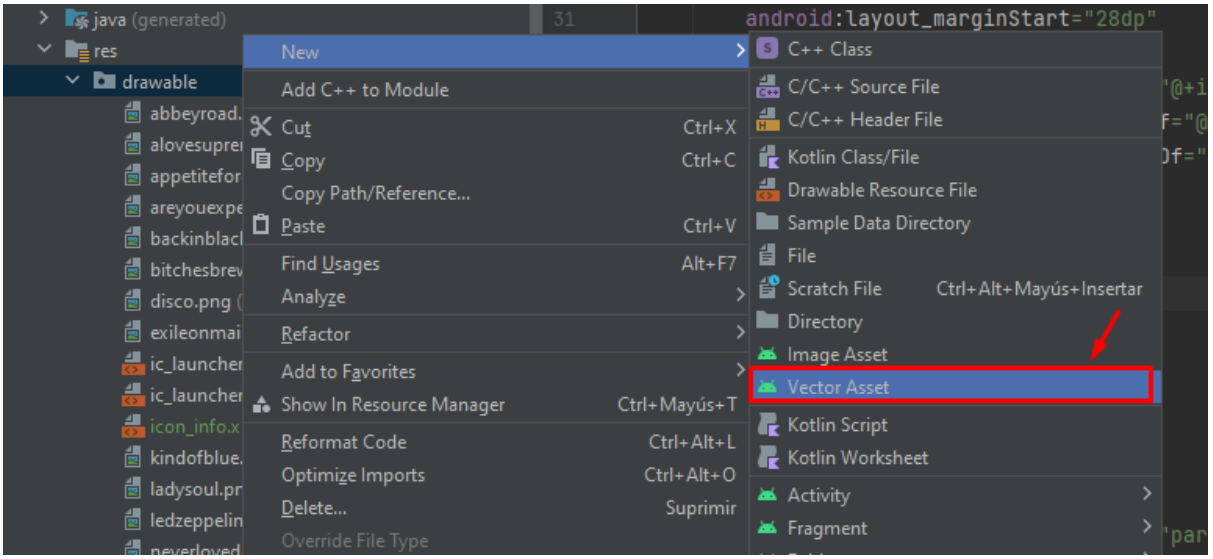
Esto se hace creando nuevas carpetas: clic derecho sobre nuestra app, **New/ Package.**



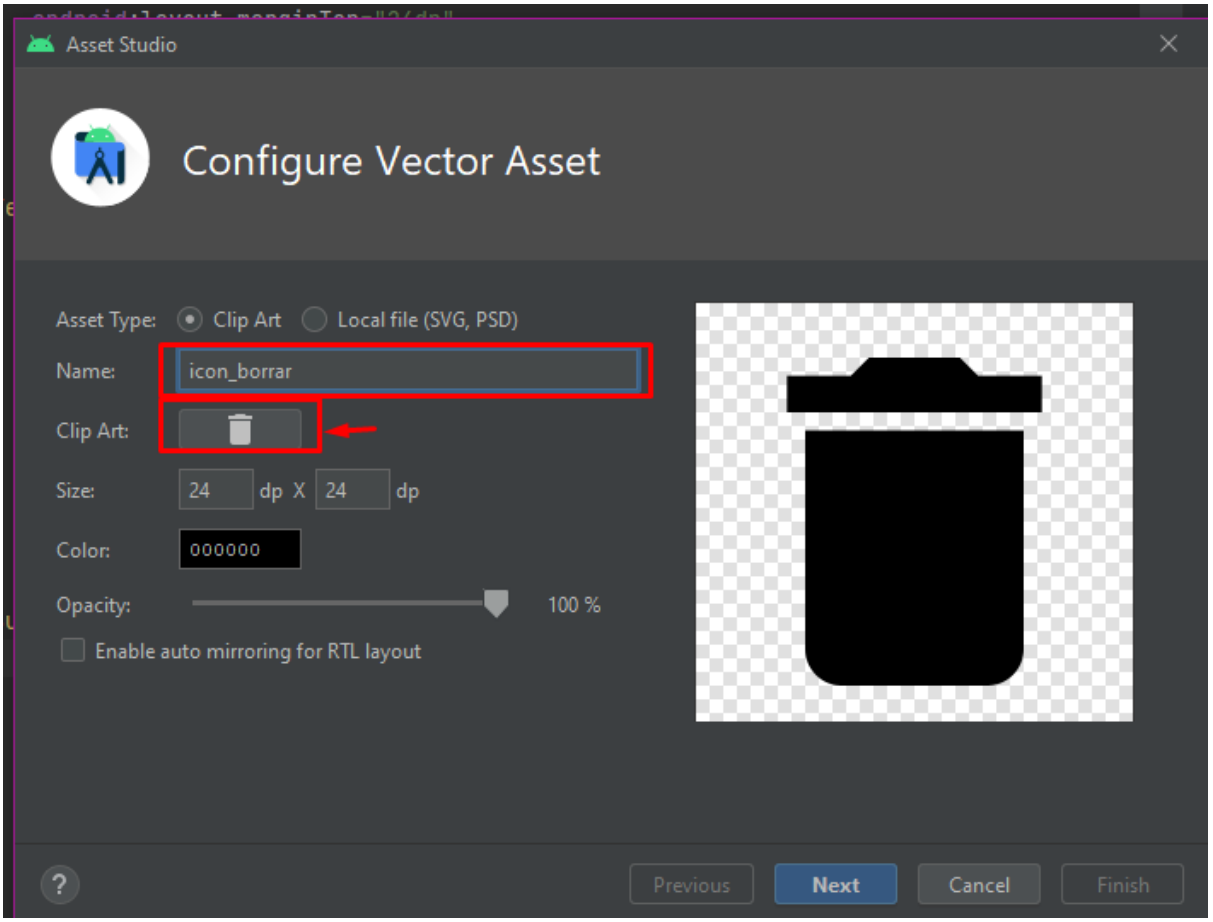
Movemos las clases a la carpeta que corresponda y nos pide refactorizar.

Iconos

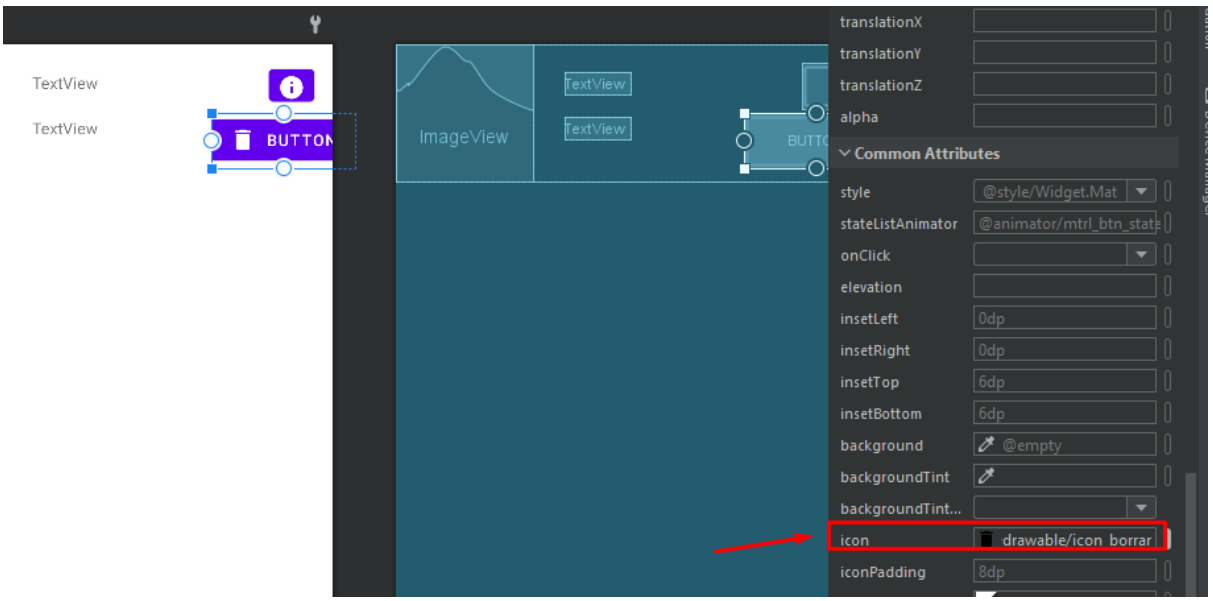
Para crear los iconos de info y de eliminar. Clic derecho encima de **drawable New/ Vector Asset**



Escogemos el icono y ponemos un nombre



Después en el diseño se agrega en un button



Recursos examen

ViewBindind
buildFeatures {
viewBinding true
}

RecyclerView
https://cursokotlin.com/capitulo-15-recyclerview-kotlin/