**CURSO KOTLIIN DESDE 0**

**Guía de estilo:**

**Pascal case** es un estilo de mayúsculas y minúsculas en el que las palabras de un nombre de clase o tipo se unen y la primera letra de cada una se escribe con mayúscula, como PascalCase.

**Camel case** es un estilo de mayúsculas y minúsculas en el que los nombres de variables o funciones se combinan y solo se escribe con mayúscula la segunda letra, como camelCase

**Package:** todo en minúsculas => package com.ejemplo.miproyecto

**Clases**(**P**ascal**C**ase): class Characte, InmutableList, List

**Funciones** (camel**C**ase)**:** fun sendMessage, stop

**@Composable fun**(**P**ascal**C**ased)**:** fun NameTag(name: String)

**Constantes(**UPPER\_SNAKE\_CASE**):** const val NUMBER\_EJEMPLO = 5

**Valores**(camel**C**ase)**:** val nomMiValor = “texto”

Debe haber un espacio antes de la llave de apertura.

El cuerpo de la función debe tener una sangría de 4 espacios. No uses caracteres de tabulación

**Comentarios:**

// Comentario simple

*/\*\* Comentario en una sola línea \*/*

*/\*\*  
 \* Comentario en  
 \* varias líneas  
 \*/*

**Variables**

**val** = valores que no se pueden modificar

**var** = variables a los que sí que se les puede modificar el valor

package com.user.cursokotlindesde0.androidmaster  
  
val *variableDeClase*: Int = 55 // Una variable de clase puede acceder toda la clase  
  
fun main() {  
 val variableDeFuncion: String = "Texto" //Una variable de función sólo puede acceder su función  
  
 */\*\* Como declara variables o valores: (Se puede especificar el tipo, pero no es necesario)\*/  
 /\*\* Variables numéricas \*/  
 /\*\* Int \*/* val intValEdad: Int = 30 // == val intEdad = 30  
 var intVarEdad: Int = 30 // == var intEdad2 = 30  
  
 */\*\* Long ( como Int pero usa más memoria del pc que los Int) \*/* val longValEjemplo: Long = 30000 // == val longEjemplo = 30000  
 var longVarEjemplo: Long = 30000 // == val longEjemplo = 30000  
  
 */\*\* Float, esta soporta números decimales, hasta 6 decimales precedido de la "f" \*/* val floatValEjemplo: Float = 30.0f  
  
 // Double, soporta 14 decimales y consume más memoria que lo Float  
 val doubleValEjemplo: Double = 35.2654  
 var doubleVarEjemplo: Double = 35.2654  
  
 */\*\* Variables alfanuméricas \*/  
 /\*\* Char, sólo soporta un carácter (entre comillas simples). \*/* val charValEjemplo1: Char = 'e'  
 val charValEjemplo2: Char = '2' // No es numérico  
 val charValEjemplo3: Char = '@'  
  
 */\*\* String vale para todo \*/* val stringValEjemplo: String = "Texto lo "  
 val stringVarEjemplo: String = " que sea "  
  
 */\*\* String Multilíneas \*/* val strMulLin: String = """   
 |Esto es un texto de pruebas Multi líneas  
 |con un parámetro final que se puede cambiar   
 |según lo que busques  
 """.*trimMargin*()  
  
 */\*\* Variables booleanas \*/  
 /\*\* Boolean, verdadero o falso \*/* var booleanEjemplo1: Boolean = true  
 var booleanEjemplo2: Boolean = false  
  
 */\*\* Para convertir un tipo de valor a otro \*/* val newInt: Int = floatValEjemplo.toInt() // newInt = 30

/\*\* CONCATENACIONES \*/  
  
 */\*\* Concatenar Strins \*/* val strConca01: String = stringValEjemplo + stringVarEjemplo // Texto lo que sea  
  
 */\*\* Concatenar String con una variable se usará el símbolo $ \*/* val strConca02: String = " Mi edad es de $intValEdad " //Mi edad es de 30  
  
 */\*\* Concatenar con .plus() \*/* val strConca03: String = strConca01.plus(strConca02) // Texto lo que sea Mi edad es de 30  
  
 */\*\* Concatenar con StringBuilders "Recomendado Kotlin"\*/* val strConca04: String =  
 StringBuilder(strConca01).append(strConca02)  
 .toString() // Texto lo que sea Mi edad es de 30  
  
 */\*\* Concatenar usando .joinToString("") \*/* val list = *listOf*("Hola", "Mundo", "Tierra", "Nuevo")  
 val strConca05 = list.*joinToString*("", "[", "]", 3) // [HolaMundoTierra...]  
  
 */\*\* Concatenar con ${...} \*/* val strConca06 = "Sumando variables ${intValEdad + intVarEdad}" // Sumando variables 60  
  
 *println*(strConca06)  
  
}

**Constantes:**

*Para declarar una constante, hay que poner el const delante y va en letras mayúsculas:*

const val IMC\_KEY = "NOMBRE\_DE\_LA\_CONSTANTE"

Para Usar una constante en todo el proyecto:

*/\*\* Para declarar una constante para que se pueda acceder desde cualquier parte del proyecto \*/*companion object {  
 */\*\* Lo que esté aquí dentro, t.o.d.o. el mundo puede acceder a ello \*/  
 /\*\* Para declarar una constante, hay que poner el const delante y va en letras mayúsculas \*/* const val IMC\_KEY = "NOMBRE\_DE\_LA\_CONSTANTE"  
 */\*\* En la Activity dónde se vaya a usar, hay que importar la \*/*}

En la activity donde se vaya a usar hay importa la

import com.user.nombreApp.nombreClass.Companion.*IMC\_KEY*

**Funciones**

package com.user.cursokotlindesde0.androidmaster  
  
*/\*\* Las funciones van con el camelCase \*/*fun main() {  
 *holaMundo*() // Llamar a una función básica  
 *miEdad*(45) // Llamar a una función pasándole un parámetro  
 *miNumeroFavorito*() // Llamar a una función con parámetro por defecto  
 *sumarValores*(16, 54) // Llamar una fun con retorno de valores  
}  
  
*/\*\* función básica \*/*fun holaMundo() {  
 *println*("Hola Mundo")  
}  
  
*/\*\* función con parámetro de entrada \*/*fun miEdad(edad: Int) {  
 *println*("Mi edad es de $edad años")  
}  
  
*/\*\* Función con parámetro por defecto \*/*fun miNumeroFavorito(numFavorito: Int = 13) {  
 *println*("Mi número favorito es el $numFavorito")  
}  
  
*/\*\* función con retorno de datos \*/  
/\*\* cuando hay un "return", hay que especificar el tipo de dato que devuelve \*/*fun sumarValores(valor1: Int, valor2: Int): Int {  
 return (valor1 + valor2)  
}  
  
*/\*\* Se puede simplificar una función cuando sólo hay una sentencia \*/*fun subtract(valor1: Int, valor2: Int) = valor1 - valor2

**if else**

package com.nanduky.cursokotlindesde0.androidmaster  
  
const val *DIA*: String = "Viernes"  
const val *FESTIVO*: Boolean = false  
  
fun main() {  
 *ifBasico*(*DIA*)  
 *ifAnidado*(*DIA*)  
 *ifBoolean*(*FESTIVO*)  
}  
  
*/\*\* Un if básico \*/*fun ifBasico(dia: String) {  
 if (dia == "Viernes") {  
 */\*\* si la comparación es correcta ejecuta este código \*/  
 println*("La comparación es igual")  
 } else {  
 */\*\* Si la comparación no es correcta se ejecuta este código \*/  
 println*("La comparación no es igual")  
 }  
 */\*\* Esto se puede simplificar siempre que haya una sola sentencia \*/* if (dia == "Lunes") *println*("Si es correcto")  
 else *println*("No es correcto")  
  
}  
  
*/\*\* Un else if anidado pero esto no es óptimo, lo ideal son los when \*/*fun ifAnidado(dia: String) {  
 if (dia == "Lunes") *println*("Hoy es Lunes")  
 else if (dia == "Viernes") *println*("Hoy es Viernes")  
 else *println*("No es ningún día seleccionado")  
}  
  
*/\*\* Un if con un Boolean \*/*fun ifBoolean(festivo: Boolean) {  
 if (festivo) *println*("Es festivo") //si es true  
 if (!festivo) *println*("No es festivo") //si es false  
}

**When**

package com.nanduky.cursokotlindesde0.androidmaster  
  
const val *MES*: Int = 7  
fun main() {  
 *selectMes*(*MES*)  
 *selectTrimestre*(*MES*)  
 *selectSemestre*(*MES*)  
 *println*("Retorno de dato: ${*retornoDatos*(*MES*)}")  
 *println*("Retorno de dato mejorado: ${*retornoDatosMejorado*(*MES*)}")  
 *println*("Retorno de dato Re\_mejorado: ${*retornoDatosReMejorado*(*MES*)}")  
 *comprobarObjeto*(*MES*)  
  
  
}

*/\*\*  
 \* El when recorre todas los opciones y selecciona la de la variable  
 \* en el caso de más de una sentencia, deben ponerse las llave {...}  
 \* \*/  
/\*\* when con un solo resultado \*/*fun selectMes(mes: Int) {  
 when (mes) {  
 1 -> *println*("Enero")  
 2 -> *println*("Febrero")  
 3 -> *println*("Marzo")  
 4 -> *println*("Abril")  
 5 -> *println*("Mayo")  
 6 -> *println*("Junio")  
 7 -> *println*("Julio")  
 8 -> {  
 *println*("Agosto")  
 *println*(" mes de vacaciones")  
 }  
  
 9 -> *println*("Septiembre")  
 10 -> *println*("Octubre")  
 11 -> *println*("Noviembre")  
 12 -> *println*("Diciembre")  
 else -> *println*("No es un mes valido")  
 }  
}  
  
*/\*\* when con varios valores con el mismo resultados \*/*fun selectTrimestre(mes: Int) {  
 when (mes) {  
 1, 2, 3 -> *println*("Primer trimestre")  
 4, 5, 6 -> *println*("Segundo trimestre")  
 7, 8, 9 -> *println*("Tercer trimestre")  
 10, 11, 12 -> *println*("Cuarto trimestre")  
 else -> *println*("No es un trimestre válido")  
 }  
  
}  
  
*/\*\* when con un rango de valores con el mismo resultado \*/*fun selectSemestre(mes: Int) {  
 when (mes) {  
 in 1..6 -> *println*("Primer semestre")  
 in 7..12 -> *println*("Segundo semestre")  
 !in 1..12 -> *println*("No es un semestre válido")  
 }  
  
}  
  
*/\*\* when con devolución de datos \*/*fun retornoDatos(mes: Int): String {  
 val dato: String = when (mes) {  
 in 1..6 -> "Primer semestre"  
 in 7..12 -> "Segundo semestre"  
 !in 1..12 -> "No es un semestre válido"  
 else -> "No es un dato correcto"  
 }  
 return dato  
}

*/\*\* when con devolución de datos mejorado \*/*fun retornoDatosMejorado(mes: Int): String {  
 return when (mes) {  
 in 1..6 -> "Primer semestre"  
 in 7..12 -> "Segundo semestre"  
 !in 1..12 -> "No es un semestre válido"  
 else -> "No es un dato correcto"  
 }  
  
}  
  
fun retornoDatosReMejorado(mes: Int) = when (mes) {  
 in 1..6 -> "Primer semestre"  
 in 7..12 -> "Segundo semestre"  
 !in 1..12 -> "No es un semestre válido"  
 else -> "No es un dato correcto"  
}  
  
*/\*\* Comprobar que tipo de objeto es con when \*/*fun comprobarObjeto(mes: Any) {  
 when (mes) {  
 is Int -> *println*(mes + mes)  
 is String -> *println*(mes)  
 is Boolean -> if (mes) *println*("Es true") else println("Es false")  
 }  
}

**Nulabilidad:**

package com.nanduky.cursokotlindesde0.androidmaster  
  
  
*/\*\* Las variables pueden ser nulas asignándoles él ? al final \*/*fun main() {  
 var name: String? = null  
 *println*(name?.get(0)) // con él ? se le dice que si no es null, haga algo  
 *println*(name?.get(8) ?: "Es nulo") // en caso de que sea null haz otra cosa  
  
}

**Arrays**

package com.nanduky.cursokotlindesde0.sintaxis  
  
fun main() {  
 */\*\* Declarar una variable con valores\*/* val numerosLetras = *arrayOf*("Uno", "Dos", "Tres", "Cuatro", "Cinco")  
 numerosLetras[0] // Selecciona un valor por su índice  
 numerosLetras.size // Muestra el tamaño del array  
 numerosLetras[0] = "Primer número" // Cambiar un dato en un array  
  
 */\*\* BUCLES PARA ARRAYS \*/* for (posicion in numerosLetras.*indices*) { // Recorrer el array  
 *println*(numerosLetras[posicion])  
 }  
 for ((posicion, valor) in numerosLetras.*withIndex*()) {  
 *println*("índex: $posicion = $valor") // obtener el índice y/o el valor del array  
 }  
 for (valorNumerosLetras in numerosLetras) {  
 *println*("Valor: $valorNumerosLetras") //obtener el valor del array  
 }  
  
 numerosLetras.*forEach* **{** *println*(**it**) **}** // Recorre *todo el array* numerosLetras.*forEach* **{** valor **->** *println*(valor) **}** // Lo mismo de arriba sustituyendo el "it"  
}

**Listas**

package com.nanduky.cursokotlindesde0.androidmaster  
  
fun main() {  
 *listaInmutable*()  
 *listaMutable*()  
}  
  
fun listaInmutable() {  
 */\*\* Declarar una lista INMUTABLE (sólo lectura)\*/* val listaInmutableNum: List<String> = *listOf*("Uno", "Dos", "Tres", "Cuatro", "Cinco")  
 listaInmutableNum.size // Obtienes el tamaño de la lista  
 listaInmutableNum[0] // Obtienes el valor del índice indicado  
 listaInmutableNum.*first*() // Obtienes el primer valor de la lista  
 listaInmutableNum.*last*() // Obtienes el último valor de la lista  
 */\*\* "it" es el iterador, es cada uno de los valores de la lista en cada vuelta del recorrido \*/* val filtrar = listaInmutableNum.*filter* **{ it**.*contains*("a") **}** // Filtrar lista  
  
 */\*\* BUCLES PARA LISTAS \*/  
 /\*\* "it" es el iterador, es cada uno de los valores de la lista en cada vuelta del recorrido \*/* listaInmutableNum.*forEach* **{** *println*("a): $**it**") **}** // Obtienes todos los valores de la lista  
 */\*\* sustituye el "it" por una variable con nombre elegido por el usuario \*/* listaInmutableNum.*forEach* **{** valor **->** *println*("b) $valor") **}** // Obtienes todos los valores de la lista  
}  
  
fun listaMutable() {  
 */\*\* Declarar una lista MUTABLE (se puede modificar) \*/* val listaMutableNum: MutableList<String> = *mutableListOf*() // Se puede declarar sin valores  
 listaMutableNum.addAll(*listOf*("Uno", "Dos", "Tres", "Cuatro", "Cinco")) // Añadir valores a la lista  
 listaMutableNum.add("Seis") //Añadir un valor al final de la lista  
 listaMutableNum.add(0, "Cero") //Añadir un valor en el índice especificado  
 *println*(listaMutableNum)  
  
 */\*\* Para comprobar si la lista está vacía o no\*/* if (listaMutableNum.isEmpty()) *println*("no hay datos")  
 else listaMutableNum.*forEach* **{** *println*("c) $**it**") **}** */\*\* Para comprobar si la lista NO está vacía o no\*/* if (listaMutableNum.*isNotEmpty*()) listaMutableNum.*forEach* **{** *println*("d) $**it**") **}** else *println*("no hay datos")  
}

**setOnClickListener** (Con binding)

Este código ejecuta una acción cuando se pulsa un botón:

binding.btn2.setOnClickListener **{** Toast.makeText(this,"Botón pulsado", Toast.*LENGTH\_LONG*).show()  
**}**

**Intents**

Para navegar de una activity a otra se usan los “intent”

// Crear un intent y pasarle la dirección donde hay que ir   
 val intentResultadoActivity = Intent(this,ResultadoActivity::class.*java*)  
 // Iniciar la activity a la que se llama   
 startActivity(intentResultadoActivity)

Para navegar a otra activity y pasarle un dato se usa los “putExtra”

val nombre: String = "Juan"

// Crear un intent y pasarle la dirección donde hay que ir  
 val intentResultadoActivity = Intent(this, ResultadoActivity::class.*java*)  
 // Enviar un dato al activity que vamos a inicializar y ponerle un nombre al extra  
 intentResultadoActivity.putExtra("NOMBRE",nombre)  
 // Iniciar la activity a la que se llama  
 startActivity(intentResultadoActivity)

Para recoger el valor en la Activity que hemos llamado, en este caso un String,

// Si existe un valor con esta "key" dame lo, si no, dame lo vacío  
val nombre: String = *intent*.*extras*?.getString("NOMBRE").*orEmpty*()

Es signo “?” es por si es un valor nulo. El “orEmpty()” es para en el caso de que no haya ningún valor en la variable, nos devuelva una String vacía. Si fiera un “Int” se cambiaría el (getString) por (getInt) o por cualquier otro tipo dato que venga.

**Also**

Otra forma de pasar un dato aun textView por ejemplo sería así:

binding.tvResultado.*text* = "Hola $nombre"  
"Hola $nombre".*also* **{** binding.tvResultado.*text* = **it }**"Hola $nombre".*also* **{** valor **->** binding.tvResultado.*text* = valor **}**

**Color**

*/\*\* Para cambiar el color de un texto en un textView \*/*binding.tvNombre.setTextColor(ContextCompat.getColor(this,R.color.*nombre\_color*))

Para acceder a un color cualquier otro valor cuando no nos vale el “this”, podemos coger el contexto de cualquier vista que tengamos creada:

binding.tvNombre.setBackgroundColor(  
 ContextCompat.getColor( binding.tvNombre.*context*, R.color.*nombre\_color*))

**enableEdgeToEdge()**

Este método hace que la barra donde va la hora, batería, etc., sea como transparente, no tiene color propio y coje el del fondo del contenedor, si eliminas el método, puedes cambiar el color de la barra

A través del themes.xml, en style con el item:

<style name="Theme.mi\_estilo" parent="Theme.MaterialComponents.DayNight.NoActionBar">  
 <item name="android:statusBarColor">#000000</item>  
</style>

**Sealed Class**

Primer paso

Se crea una nueva clase: (GameModel)

class GameModel {  
}

Se cambia por una data class:

data class GameModel()

Se supone que va a ser de juegos de Nintendo y cada juego tendrán unos valores:

data class GameModel(val titulo: String, val numeroSerie: String, val error: String)

Segundo paso

Vamos a la Activity y (antes del onCreate) creamos un valor de un listado de juegos con los parámetros título ,nº serie y el posible error:

val listaJuegos = *listOf*<GameModel>(  
 GameModel("Mario", "0123456789", "error con internet"),  
 GameModel("Mario 2", "9876543210", "rayado"),  
 GameModel("Mario 3", "1478523690", ""),  
 )

Supongamos que tenemos que hacer una acción según el error.

Ahora recorreremos el listado y haremos algo por cada uno de los errores. (Dentro del onCreate)

*/\*\* Recorremos el listado de juegos y por cada error encontrado haremos algo \*/*

listaJuegos.*forEach* **{** modeloJuego **->** when (modeloJuego.error) {  
 "error con internet" -> llamarAlSoporteTecnico() // Se llamaría a la función  
 "rayado" -> eliminarJuego() // según el error  
 "" -> vender()  
 else -> niIdea() // El problema es el else, que no se sabe que hacer  
 }  
**}**

En el caso de que no sea ninguna de los tres posibles casos de error es cuando viene el problema.

Es el problema de tener el error como una String. Aquí es cuando sale la sealed Class.

Tercer paso

Volvemos a la data class GameModel y cambiamos el topo de error de String a gameError que obviamente no existe ahora mismo.

data class GameModel(val titulo: String, val numeroSerie: String, val error: GameError)

Creamos la sealed class GameError en GameModel

data class GameModel(val titulo: String, val numeroSerie: String, val error: GameError)

sealed class GameError(){

}

Y creamos los objetos del tipo “error internet, rayado y sin errores” que serán del tipo GameError

*/\*\* data Class para la lista de juegos \*/*data class GameModel(val titulo: String, val numeroSerie: String, val error: GameError)  
  
*/\*\* Creamos la sealed class \*/*sealed class GameError() {  
 */\*\* creamos los objetos de cada error que son del tipo GameError \*/  
 /\*\* Los objetos de las sealed class van en mayúscula la primera letra \*/* object Rayado : GameError()  
 object ErrorInternet : GameError()  
 object NoError : GameError()  
}

Cuarto paso

Volvemos a la activity y vemos que ahora falla el listado, cambiamos el error que es de tipo String por los objetos de la sealed class

val listaJuegos = *listOf*<GameModel>(  
 GameModel("Mario", "0123456789", GameError.ErrorInternet),  
 GameModel("Mario 2", "9876543210", GameError.Rayado),  
 GameModel("Mario 3", "1478523690", GameError.NoError),  
)

Podemos importar el “GameError” con la bombillita o Alt + Intro

val listaJuegos = *listOf*<GameModel>(  
 GameModel("Mario", "0123456789", ErrorInternet),  
 GameModel("Mario 2", "9876543210", Rayado),  
 GameModel("Mario 3", "1478523690", NoError),  
)

Una vez cambiado esto, el when falla, por lo que borramos todos los errores del juego:

*/\*\* Recorremos el listado de juegos y por cada error encontrado haremos algo \*/*listaJuegos.*forEach* **{** modeloJuego **->** when (modeloJuego.error) {  
 }  
**}**

Nos saldrá el when con un error que si le damos a la bombillita o Alt + Intro, nos saldrá una opción que dice “Add remainig braches” y nos agregará todos los tipos de errores que teníamos en la lista:

*/\*\* Recorremos el listado de juegos y por cada error encontrado haremos algo \*/*listaJuegos.*forEach* **{** modeloJuego **->** when (modeloJuego.error) {  
 InternetError -> *TODO*()  
 NoError -> *TODO*()  
 RayadoError -> *TODO*()  
 }  
**}**

Ahora se le pasan las funciones según el tipo de error

*/\*\* Recorremos el listado de juegos y por cada error encontrado haremos algo \*/*listaJuegos.*forEach* **{** modeloJuego **->** when (modeloJuego.error) {  
 InternetError -> llamarAlSoporteTecnico()  
 NoError -> vender()  
 RayadoError -> eliminarJuego()  
 }  
**}**

En el caso de que saliera un nuevo error, solo lo tendríamos que añadir en la sealed class:

*/\*\* data Class para la lista de juegos \*/*data class GameModel(val titulo: String, val numeroSerie: String, val error: GameError)  
  
*/\*\* Creamos la sealed class \*/*sealed class GameError() {  
 */\*\* creamos los objetos de cada error que son del tipo GameError \*/  
 /\*\* Los objetos de las sealed class van en mayúscula la primera letra \*/* object RayadoError : GameError()  
 object InternetError : GameError()  
 object NoError : GameError()  
 object CongeladoError : GameError() // 🡨 Nuevo error  
}

Hay que volver a la Activity y añadir el error al when:

*/\*\* Recorremos el listado de juegos y por cada error encontrado haremos algo \*/*listaJuegos.*forEach* **{** modeloJuego **->** when (modeloJuego.error) {  
 InternetError -> llamarAlSoporteTecnico()  
 NoError -> vender()  
 RayadoError -> eliminarJuego()  
 CongeladoError -> *TODO*() // 🡨 Se puede cambiar por else  
 }  
**}**

El último error se puede cambiar por un else, para que recoja todos los errores no especificados, pero no es recomendable, así te aseguras de que tienes siempre cada uno de los tipos necesarios.

Para un nuevo error con parámetros, en vez de object se crea un data class en el GameModel.kt:

*/\*\* data Class para la lista de juegos \*/*data class GameModel(val titulo: String, val numeroSerie: String, val error: GameError)  
  
*/\*\* Creamos la sealed class \*/*sealed class GameError() {  
 */\*\* creamos los objetos de cada error que son del tipo GameError \*/  
 /\*\* Los objetos de las sealed class van en mayúscula la primera letra \*/* object RayadoError : GameError()  
 object InternetError : GameError()  
 object NoError : GameError()  
 object CongeladoError : GameError()  
 */\*\* Otro error pero con parámetros \*/* data class VersionError(val version: String) : GameError() // 🡨 Error con parámetros

En el activity, si queremos crear un objeto error del tipo “VersionError” tendremos que pasarle el parámetro String que nos pide, que en este caso en la versión del juego:

*/\*\* Creamos una variable con el listado de juegos, con el titulo, nº de serie y el posible error \*/*val listaJuegos = *listOf*<GameModel>(  
 GameModel("Mario", "0123456789", InternetError),  
 GameModel("Mario 2", "9876543210", RayadoError),  
 GameModel("Mario 3", "1478523690", NoError),  
 */\*\* Si creamos un objeto error del tipo VersionError, le tendremos que pasar el parámetro que pide \*/* GameModel("Mario 4", "3658452257", VersionError("1.2.0")), 🡨 nuevo juego  
)

Añadimos en el when como antes el nuevo objeto, que en este caso no es un “object” sino un “data class” , hay que poner un ”is” delante:

listaJuegos.*forEach* **{** modeloJuego **->** when (modeloJuego.error) {  
 InternetError -> llamarAlSoporteTecnico()  
 NoError -> vender()  
 RayadoError -> eliminarJuego()  
 CongeladoError -> revisar()  
 is VersionError -> *TODO*()  
 }  
**}**

Para recuperar la versión del juego, cómo estamos iterando sobre cada uno de los objetos “juego” que es a través del “modeloJuego” del forEach de la gameList, “error” (del parámetro de la data class GameModel) y version (del data class VersionError dentro de la sealed class)

*/\*\* Recorremos el listado de juegos y por cada error encontrado haremos algo \*/*listaJuegos.*forEach* **{** modeloJuego **->** when (modeloJuego.error) {  
 InternetError -> llamarAlSoporteTecnico()  
 NoError -> vender()  
 RayadoError -> eliminarJuego()  
 CongeladoError -> revisar()  
 is VersionError -> consultarVersion(modeloJuego.error.version) // Al ser un data class hay que poner el "is" delante  
 }  
**}**

En la función consultarVersion, hay ponerle que recibe un parámetro: (Alt + Enter)

fun consultarVersion(version: String) {

// El código aquí …   
}

**RecyclerView (**añadir ítems con código**)**

Lo primero es crear una sealed class (Clase sellada). Para esto, se crea una nueva clase normal y luego se le añade delante el sealed

class TodoTareaCategorias {} // Primero se crea la clase normalmente

sealed class TodoTareaCategorias {} // Luego se le añade el sealed

Se crean objetos y se le pasa el tipo de objeto

object Personal : TodoTareaCategorias()

Quedaría así si son 3 categorías:

sealed class TodoTareaCategorias {  
 object Personal : TodoTareaCategorias()  
 object Negocio : TodoTareaCategorias()  
 object Otro : TodoTareaCategorias()  
}

En el RecyclerView, cuando lo vayamos a pintar, le vamos a mandar un listado de tareaCategoria: List<TodoTareaCategoria> que podrá contener cualquiera de los objetos de tipo creados en la sealed class, pero ningún otro que nos sean esos.

Para que funcione un RecyclerView, consta de dos partes, hay que crearle un “adapter” y un “viewHolder”

El “adapter” es la clase que va a conectar toda la información con el RecyclerView y el “viewHolder” va a ser la clase que lo pinta.

Creamos una nueva “Kotlin class” (TodoCategoriasAdapter):

class TodoCategoriasAdapter {  
}

Y esta clase deberá recibir un parámetro, como una función:

class TodoCategoriasAdapter() {  
}

Recibirá un parámetro de una lista privada del tipo “TodoTareaCategorias”

class TodoCategoriasAdapter(private val categoria:List<TodoTareaCategorias>) {  
}

(TodoTareaCategorias es lo de la sealed class). Cuando creamos esta clase, hay que pasarle el listado que queremos pintar.

Creamos una nueva “Kotlin class” (TodoCategoriasViewHolder):

class TodoCategoriasViewHolder {  
}

El ViewHolde recibirá una “View”: (los imports, se van importando, normalmente lo hace sólo)

import android.view.View  
  
class TodoCategoriasViewHolder(view: View) {  
}

Este “viewHolder” va a implementar el RecycleView.ViewHolder y va a recibir un parámetro que es “view”:

import android.view.View  
import androidx.recyclerview.widget.RecyclerView  
  
class TodoCategoriasViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
}

Volvemos al “adapter” y lo extendemos a “RecyclerView.Adapter” y al “viewHolder”:

import androidx.recyclerview.widget.RecyclerView  
  
class TodoCategoriasAdapter(private val categoria: List<TodoTareaCategorias>) :  
 RecyclerView.Adapter<TodoCategoriasViewHolder>() {  
}

Esto empezará a llorar diciendo que es una clase abstracta que necesita importar unos métodos, bombillita y “Implement members” y seleccionamos los 3:

import android.view.ViewGroup  
import androidx.recyclerview.widget.RecyclerView  
  
class TodoCategoriasAdapter(private val categoria: List<TodoTareaCategorias>) :  
 RecyclerView.Adapter<TodoCategoriasViewHolder>() {

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TodoCategoriasViewHolder {  
 *TODO*("Not yet implemented")  
 }  
  
 override fun getItemCount(): Int {  
 *TODO*("Not yet implemented")  
 }  
  
 override fun onBindViewHolder(holder: TodoCategoriasViewHolder, position: Int) {  
 *TODO*("Not yet implemented")  
 }  
}

El adapter es el que nos permite pintar las listas y mostrarlas, y necesita cierta información.

Si empezamos por el más fácil, (getItemCount) es el tamaño de la lista que va a mostrar:

override fun getItemCount(): Int {  
 // Retorna el tamaño de la lista categoría  
 return categoria.size  
}

Esto se puede optimizar de la siguiente forma:

override fun getItemCount = categoria.size

El (onCreateViewHolder), lo que hace es crear la vista, el contenido, cardView o lo que sea para que el (onBindViewHolder) pueda pasarle la información que tiene que pintar.

Al onCreateViewHolder no le podemos pasar una vista porque aún no está creada, así que hay que crearla.

En layauts creamos un nuevo layaut. New->Layout Resouce File. Normalmente los diseños de los recyclerView, las vistas son items, por lo que le pondremos un nombre que lo refleje item\_todo\_tarea\_categorias.

<?xml version="1.0" encoding="utf-8"?>  
<androidx.cardview.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
</androidx.cardview.widget.CardView>

Hay que importar las app:

xmlns:app="http://schemas.android.com/apk/res-auto"

El diseño básico quedaría así:

<?xml version="1.0" encoding="utf-8"?>  
<androidx.cardview.widget.CardView 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:layout\_width="160dp"  
 android:layout\_height="90dp"  
 android:layout\_marginHorizontal="8dp"  
 android:layout\_marginVertical="16dp"  
 app:cardBackgroundColor="@color/todo\_background\_card"  
 app:cardCornerRadius="16dp">  
  
 <LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:layout\_margin="8dp"  
 android:gravity="center\_vertical"  
 android:orientation="vertical">  
  
 <TextView  
 android:id="@+id/tv\_nombre\_categoria"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:textAllCaps="true"  
 android:textColor="@color/white"  
 android:textSize="19sp"  
 android:textStyle="bold"  
 tools:text="Prueba" />  
  
 <View  
 android:id="@+id/linea\_divisoria"  
 android:layout\_marginTop="4dp"  
 android:layout\_width="match\_parent"  
 android:layout\_height="2dp"  
 tools:background="@color/white" />  
 </LinearLayout>  
</androidx.cardview.widget.CardView>

Volvemos al adapter.

Creamos un inflador de vistas en el onCreateViewHolder:

val view = LayoutInflater.from(parent.*context*)

Necesitamos un contexto, como no tenemos contexto aquí (el this) lo podemos sacar de cualquier componente, como el ViewGroup es un componente lo extraemos de ahí:

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TodoCategoriasViewHolder {  
 */\*\* Creamos un inflador de vistas. El context lo extraemos del ViewGroup \*/* val nuevaView = LayoutInflater.from(parent.*context*)  
 .inflate(R.layout.*item\_todo\_tarea\_categorias*, parent, false)  
   
}

Inflamos la vista layaut (hay que importar la “R” y le pasamos el viewGroup que es donde se va a meter y un false.

Ahora hay que retornar un ViewHolder que pedía una view “TodoCategoriasViewHolder(view: View)“ y le pasamos la view que acabamos de crear:

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TodoCategoriasViewHolder {  
 */\*\* Creamos un inflador de vistas. El context lo extraemos del ViewGroup \*/* val nuevaView = LayoutInflater.from(parent.*context*)  
 .inflate(R.layout.*item\_todo\_tarea\_categorias*, parent, false)  
 return TodoCategoriasViewHolder(nuevaView)  
}

El el método que queda, que ya recibe es el viewHolde y la posición. Entonces cogemos “holder” que sería como llamar a cada uno de los itms y la pasamos un render, que es una función que aún no existe, que hay que crearla en el viewHolder:

override fun onBindViewHolder(holder: TodoCategoriasViewHolder, position: Int) {  
 holder.render  
}

Volvemos al “TodoCategoriasViewHolder” y creamos la función render

import android.view.View  
import androidx.recyclerview.widget.RecyclerView  
  
class TodoCategoriasViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
  
 */\*\* Esta función recibe una tareaCategoría\*/* fun render(tareaCategorias: TodoTareaCategorias) {  
   
 }  
}

Ahora que está creada la función en el “TodoCategoriasAdapter” podemos llamar la función:

override fun onBindViewHolder(holder: TodoCategoriasViewHolder, position: Int) {  
 holder.render()  
}

Ahora la función me pide una tareaCategorias y tenemos un listado de categories , y para saber cuál tenemos que pasarle, tenemos la “posotion”

override fun onBindViewHolder(holder: TodoCategoriasViewHolder, position: Int) {  
holder.render(categoria[position])  
}

Y ya tendríamos el adapter terminado y quedaría así:

import android.view.LayoutInflater  
import android.view.ViewGroup  
import androidx.recyclerview.widget.RecyclerView  
import com.nanduky.cursokotlindesde0.R  
  
class TodoCategoriasAdapter(private val categoria: List<TodoTareaCategorias>) :  
 RecyclerView.Adapter<TodoCategoriasViewHolder>() {  
  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TodoCategoriasViewHolder {  
 */\*\* Creamos un inflador de vistas. El context lo extraemos del ViewGroup \*/* val nuevaView = LayoutInflater.from(parent.*context*)  
 .inflate(R.layout.*item\_todo\_tarea\_categorias*, parent, false)  
 return TodoCategoriasViewHolder(nuevaView)  
 }  
 override fun onBindViewHolder(holder: TodoCategoriasViewHolder, position: Int) {holder.render(categoria[position])  
 }  
 override fun getItemCount() = categoria.size  
}

Volvemos al “TodoCategoriasViewHolder.ky”.

El ViewHolder es básicamente donde pintamos

Creo el “binding” para acceder a las vistas:

import android.view.View  
import androidx.recyclerview.widget.RecyclerView  
import com.nanduky.cursokotlindesde0.databinding.ItemTodoTareaCategoriasBinding  
  
class TodoCategoriasViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
  
 */\*\* creamos el binding \*/* private lateinit var binding: ItemTodoTareaCategoriasBinding  
  
 */\*\* Esta función recibe una tareaCategoría \*/* fun render(tareaCategorias: TodoTareaCategorias) {  
 }  
}

El siguiente paso es en TodoActivite.kt y creamos un acceso para el TodoCAtegoriasAdapter.kt

class TodoActivity : AppCompatActivity() {  
  
 */\*\* Crear el binding \*/* private lateinit var binding: ActivityTodoBinding  
  
 */\*\* Crear un acceso al clase TodoCategoriasAdapter.kt \*/* private lateinit var todoCategoriasAdapter: TodoCategoriasAdapter

...

Inicializamos el todoCategoriasAdapter

class TodoActivity : AppCompatActivity() {  
  
 */\*\* Crear el binding \*/* private lateinit var binding: ActivityTodoBinding  
  
 */\*\* Crear un acceso al clase TodoCategoriasAdapter.kt \*/* private lateinit var todoCategoriasAdapter: TodoCategoriasAdapter  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *enableEdgeToEdge*()  
 */\*\* Inflar el binding con la vista \*/* binding = ActivityTodoBinding.inflate(*layoutInflater*)  
 setContentView(binding.*root*)

*/\*\* \*/* initUI()  
 }  
  
 private fun initUI() {  
 */\*\* Inicializar el todoCategoriasAdapter \*/* todoCategoriasAdapter = TodoCategoriasAdapter() // 🡨 inicializar  
 }  
}

TodoCategoriasAdapter() necesita un listado de categorías que ahora mismo no tenemos y lo tenemos que crear:

import android.os.Bundle  
import androidx.activity.enableEdgeToEdge  
import androidx.appcompat.app.AppCompatActivity  
import androidx.core.view.ViewCompat  
import androidx.core.view.WindowInsetsCompat  
import com.nanduky.cursokotlindesde0.R  
import com.nanduky.cursokotlindesde0.databinding.ActivityTodoBinding  
  
class TodoActivity : AppCompatActivity() {  
  
 */\*\* Crear el listado de categorías para el TodoCategoriasAdpter() \*/* private val categorias = *listOf*(  
 TodoTareaCategorias.Negocio,  
 TodoTareaCategorias.Personal, // 🡨 Nuevo listado  
 TodoTareaCategorias.Otro  
 )  
  
 */\*\* Crear el binding \*/* private lateinit var binding: ActivityTodoBinding  
  
 */\*\* Crear un acceso al clase TodoCategoriasAdapter.kt \*/* private lateinit var todoCategoriasAdapter: TodoCategoriasAdapter  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *enableEdgeToEdge*()  
 */\*\* Inflar el binding con la vista \*/* binding = ActivityTodoBinding.inflate(*layoutInflater*)  
 setContentView(binding.*root*)  
 ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.*main\_todo*)) **{** v, insets **->** val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())  
 v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)  
 insets  
 **}** */\*\* \*/* initUI()  
 }  
  
 private fun initUI() {  
 */\*\* Inicializar el todoCategoriasAdapter \*/* todoCategoriasAdapter = TodoCategoriasAdapter()  
 }  
}

Aquí emos creado un listado con las categorías que necesitamos (de la sealed class) y ya se la podemos pasar a TodoCategoriasAdapter() .

...  
private fun initUI() {  
 */\*\* Inicializar el todoCategoriasAdapter \*/* todoCategoriasAdapter = TodoCategoriasAdapter(categorias)  
}

Al rvTodoCategoria hay que ponerle un layoutManager que es el que se encarga de si lo queremos horizontal o vertical:

private fun initUI() {  
 */\*\* Inicializar el todoCategoriasAdapter \*/* todoCategoriasAdapter = TodoCategoriasAdapter(categorias)  
 binding.rvTodoCategoria.*layoutManager* =  
 LinearLayoutManager(this, LinearLayoutManager.*HORIZONTAL*, false)   
 */\*\* También le pasamos el adapte \*/* binding.rvTodoCategoria.*adapter* = todoCategoriasAdapter

}

El TodoActivity quedaría así de momento:

import android.os.Bundle  
import androidx.activity.enableEdgeToEdge  
import androidx.appcompat.app.AppCompatActivity  
import androidx.core.view.ViewCompat  
import androidx.core.view.WindowInsetsCompat  
import androidx.recyclerview.widget.LinearLayoutManager  
import com.nanduky.cursokotlindesde0.R  
import com.nanduky.cursokotlindesde0.databinding.ActivityTodoBinding  
  
class TodoActivity : AppCompatActivity() {  
  
 */\*\* Crear el listado de categorías para el TodoCategoriasAdpter() \*/* private val categorias = *listOf*(  
 TodoTareaCategorias.Negocio,  
 TodoTareaCategorias.Personal,  
 TodoTareaCategorias.Otro  
  
 )  
  
 */\*\* Crear el binding \*/* private lateinit var binding: ActivityTodoBinding  
  
 */\*\* Crear un acceso al clase TodoCategoriasAdapter.kt \*/* private lateinit var todoCategoriasAdapter: TodoCategoriasAdapter  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *enableEdgeToEdge*()  
 */\*\* Inflar el binding con la vista \*/* binding = ActivityTodoBinding.inflate(*layoutInflater*)  
 setContentView(binding.*root*)*/\*\* \*/* initUI()  
 }  
   
 private fun initUI() {  
 */\*\* Inicializar el todoCategoriasAdapter \*/* todoCategoriasAdapter = TodoCategoriasAdapter(categorias)  
 binding.rvTodoCategoria.*layoutManager* =  
 LinearLayoutManager(this, LinearLayoutManager.*HORIZONTAL*, false)  
 */\*\* También le pasamos el adapte \*/* binding.rvTodoCategoria.*adapter* = todoCategoriasAdapter  
 }  
}

Volvemos a TodoCategoriasViewHolder.kt y creamos un when que recorrerá toda la sealed class TodoTareaCategorias, si se le da al Alt + Intro, te implementa el when con los componentes necesarios y los implementamos con los valores que hacen falta para que los pinte en el RecyclerView:

package com.nanduky.cursokotlindesde0.todoapp  
  
import android.view.View  
import androidx.core.content.ContextCompat  
import androidx.recyclerview.widget.RecyclerView  
import com.nanduky.cursokotlindesde0.R  
import com.nanduky.cursokotlindesde0.databinding.ItemTodoTareaCategoriasBinding  
  
class TodoCategoriasViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
  
 */\*\* creamos el binding \*/* private val binding = ItemTodoTareaCategoriasBinding.bind(view)  
  
 */\*\* Esta función recibe una tareaCategoría y una strimg\*/* fun render(tareaCategorias: TodoTareaCategorias) {  
 */\*\* Se crea un when que recorra las categorías de la sealed class\*/* when (tareaCategorias) {  
 TodoTareaCategorias.Negocio -> {  
 // El texto de la categoría  
 binding.tvNombreCategoria.*text* = "Negocios"  
 // El color de la línea divisoria  
 binding.lineaDivisoria.setBackgroundColor(  
 ContextCompat.getColor(  
 binding.lineaDivisoria.*context*,  
 R.color.*todo\_trabajo\_categoria* ) ) }  
  
 TodoTareaCategorias.Otro -> {  
 binding.tvNombreCategoria.*text* = "Otros"  
 binding.lineaDivisoria.setBackgroundColor(  
 ContextCompat.getColor(  
 binding.lineaDivisoria.*context*,  
 R.color.*todo\_personal\_categoria* ) ) }  
  
 TodoTareaCategorias.Personal -> {  
 binding.tvNombreCategoria.*text* = "Personal"  
 binding.lineaDivisoria.setBackgroundColor(  
 ContextCompat.getColor(  
 binding.lineaDivisoria.*context*,  
 R.color.*todo\_otros\_categoia* ) ) }  
 }  
 }  
}

Para quitar la advertencia del texto “Negocios” hay que pasarle la dirección del string:

binding.tvNombreCategoria.*text* =  
 binding.tvNombreCategoria.*context*.getString(R.string.*todo\_negocio*)

**RecyclerView (**añadir ítems en ejecución app**)**

Hay que crear otro adapter

class TodoTareasAdapter {  
}

y un viewHolder:

class TodoTareasViewHolder {  
}

El ViewHolder se le pasa un view y se le dice que es del tipo recyclerView.

También hay que crear una función que en este caso no recibirá una TareaCategorias, sino que recibirá un objeto que aún no se ha creado:

import android.view.View  
import androidx.recyclerview.widget.RecyclerView  
  
class TodoTareasViewHolder(view: View ):RecyclerView.ViewHolder(view) {  
 fun render(){ 🡨función  
 //...  
 }  
}

Creamos una “data class” que es crear una clase normal y ponerle “data” delante y cambiar los corchetes por paréntesis:

data class TodoTareas ()

Lleva varios atributos:

nombreTarea 🡪 El nombre de la tarea.

categorías:TodoTareaCategoria 🡪Para saber de qué categoría es cada tarea para poder asociarla

estaSelect: Boolean = false 🡪 Cuando se crea una tarea, este parámetro no hay que pasárselo, porque cuando creas una tarea siempre va a esta en siempre en false porque lo acabas de crear.

data class TodoTareas(val nombreTarea: String, val categorias: TodoTareaCategorias, var estaSelect:Boolean = false)

Volvemos a viewHolden (TodoTareasViewHolder) y le pasamos el objeto de tipo TodoTarea que acabamos de crear:

class TodoTareasViewHolder(view: View ):RecyclerView.ViewHolder(view) {  
 fun render(tarea : TodoTareas){ 🡨 pasamos objeto creado  
 //...  
 }  
}

Ahora en el adapter (TodoTareasAdapter) en vez de recibir un listado de categorías, va a recibir un listado de tareas:

class TodoTareasAdapter(private val tareas:List<TodoTareas>) {  
}

Y hay que pasarle el tipo RecyclerView con su ViewHolder:

class TodoTareasAdapter(private val tareas:List<TodoTareas>) : RecyclerView.Adapter<TodoTareasViewHolder>(){  
}

Llora el class, pues implementamos los métodos que nos pide con la bombillita o Alt + Intro:

import android.view.ViewGroup  
import androidx.recyclerview.widget.RecyclerView  
  
class TodoTareasAdapter(private val tareas: List<TodoTareas>) :  
 RecyclerView.Adapter<TodoTareasViewHolder>() {  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TodoTareasViewHolder {  
 *TODO*("Not yet implemented")  
 }  
  
 override fun getItemCount(): Int {  
 *TODO*("Not yet implemented")  
 }  
  
 override fun onBindViewHolder(holder: TodoTareasViewHolder, position: Int) {  
 *TODO*("Not yet implemented")  
 }  
}

Le pasamos el tamaño:

import android.view.ViewGroup  
import androidx.recyclerview.widget.RecyclerView  
  
class TodoTareasAdapter(private val tareas: List<TodoTareas>) :  
 RecyclerView.Adapter<TodoTareasViewHolder>() {  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TodoTareasViewHolder {  
 *TODO*("Not yet implemented")  
 }   
 override fun getItemCount() = tareas.size 🡨 El tamaño de la lista  
  
 override fun onBindViewHolder(holder: TodoTareasViewHolder, position: Int) {  
 *TODO*("Not yet implemented")  
 }  
}

Para el onCreateViewHolder hay que crear la vista xml:

<?xml version="1.0" encoding="utf-8"?>  
<androidx.cardview.widget.CardView 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:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginHorizontal="8dp"  
 android:layout\_marginVertical="4dp"  
 app:cardBackgroundColor="@color/todo\_background\_card"  
 app:cardCornerRadius="16dp">  
  
 <androidx.appcompat.widget.LinearLayoutCompat  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginStart="8dp"  
 android:paddingVertical="8dp"  
 android:gravity="center\_vertical"  
 android:orientation="horizontal">  
  
 <CheckBox  
 android:id="@+id/cb\_todo\_tarea"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content" />  
  
 <TextView  
 android:id="@+id/tv\_todo\_tarea"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:textAllCaps="false"  
 android:textColor="@color/white"  
 android:textSize="18sp"  
 tools:text="Prueba" />  
  
  
 </androidx.appcompat.widget.LinearLayoutCompat>  
  
</androidx.cardview.widget.CardView>

Y ya podemos ir al adapter e implementamos el método onCreateViewHolder:

import android.view.LayoutInflater  
import android.view.ViewGroup  
import androidx.recyclerview.widget.RecyclerView  
import com.nanduky.cursokotlindesde0.R  
  
class TodoTareasAdapter(private val tareas: List<TodoTareas>) :  
 RecyclerView.Adapter<TodoTareasViewHolder>() {  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TodoTareasViewHolder {  
 */\*\* Creamos un inflador de vistas. El context lo extraemos del ViewGroup \*/* val nuevaVista =  
 LayoutInflater.from(parent.*context*).inflate(R.layout.*item\_todo\_tarea*, parent, false)  
 return TodoTareasViewHolder(nuevaVista) 🡨 método relleno  
 }  
  
 override fun getItemCount() = tareas.size  
  
 override fun onBindViewHolder(holder: TodoTareasViewHolder, position: Int) {  
 *TODO*("Not yet implemented")  
 }  
}

y el tercer método que nos queda:

import android.view.LayoutInflater  
import android.view.ViewGroup  
import androidx.recyclerview.widget.RecyclerView  
import com.nanduky.cursokotlindesde0.R  
  
class TodoTareasAdapter(private val tareas: List<TodoTareas>) :  
 RecyclerView.Adapter<TodoTareasViewHolder>() {  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TodoTareasViewHolder {  
 */\*\* Creamos un inflador de vistas. El context lo extraemos del ViewGroup \*/* val nuevaVista =  
 LayoutInflater.from(parent.*context*).inflate(R.layout.*item\_todo\_tarea*, parent, false)  
 return TodoTareasViewHolder(nuevaVista)  
 }  
  
 override fun getItemCount() = tareas.size  
  
 override fun onBindViewHolder(holder: TodoTareasViewHolder, position: Int) {  
 holder.render(tareas[position]) 🡨 Le pasamos la posición  
 }  
}

Volvemos al TodoTareaViewHolder a rellenar la función.

Primero creamos el binding para acceder a las vistas:

import android.view.View  
import androidx.recyclerview.widget.RecyclerView  
import com.nanduky.cursokotlindesde0.databinding.ItemTodoTareaBinding  
  
class TodoTareasViewHolder(view: View ):RecyclerView.ViewHolder(view) {  
  
 */\*\* Creamos el binding \*/* private val binding = ItemTodoTareaBinding.bind(view) 🡨 binding creado  
  
 fun render(tarea : TodoTareas){  
 //...  
 }  
}

Y le pasamos el nombre al textView para probar si lo pinta.

import android.view.View  
import androidx.recyclerview.widget.RecyclerView  
import com.nanduky.cursokotlindesde0.databinding.ItemTodoTareaBinding  
  
class TodoTareasViewHolder(view: View ):RecyclerView.ViewHolder(view) {  
  
 */\*\* Creamos el binding \*/* private val binding = ItemTodoTareaBinding.bind(view)   
  
 fun render(tarea : TodoTareas){  
 binding.tvTodoTarea.*text* = tarea.nombreTarea 🡨 nombre de la data class  
 }  
}

Vamos al TodoActivity y creamos el acceso al adapter:

*...*

*/\*\* Crear el binding \*/*private lateinit var binding: ActivityTodoBinding  
  
*/\*\* Crear un acceso al clase TodoCategoriasAdapter.kt \*/*private lateinit var todoCategoriasAdapter: TodoCategoriasAdapter  
*/\*\* Crear un acceso al clase TodoTareasAdapter.kt \*/*private lateinit var todoTareasAdapter: TodoTareasAdapter 🡨 acceso al adapter

...

Inicializamos el TodoTareasAdapter:

import android.os.Bundle  
import androidx.activity.enableEdgeToEdge  
import androidx.appcompat.app.AppCompatActivity  
import androidx.core.view.ViewCompat  
import androidx.core.view.WindowInsetsCompat  
import androidx.recyclerview.widget.LinearLayoutManager  
import com.nanduky.cursokotlindesde0.R  
import com.nanduky.cursokotlindesde0.databinding.ActivityTodoBinding  
  
class TodoActivity : AppCompatActivity() {  
  
 */\*\* Crear el listado de categorías para el TodoCategoriasAdapter() \*/* private val categorias = *listOf*(  
 TodoTareaCategorias.Negocio,  
 TodoTareaCategorias.Personal,  
 TodoTareaCategorias.Otro,  
  
 /\* Esta es la lista que se pintará en el RecyclerView. Pinta 3 categorías, no porque sean las del TodoTareaCategorias, sino porque son las que se le pasan aquí, si hubiesen 10, pintaría 10 \*/  
  
 )  
  
 */\*\* Crear el binding \*/* private lateinit var binding: ActivityTodoBinding  
  
 */\*\* Crear un acceso al clase TodoCategoriasAdapter.kt \*/* private lateinit var todoCategoriasAdapter: TodoCategoriasAdapter  
 */\*\* Crear un acceso al clase TodoTareasAdapter.kt \*/* private lateinit var todoTareasAdapter: TodoTareasAdapter  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *enableEdgeToEdge*()  
 */\*\* Inflar el binding con la vista \*/* binding = ActivityTodoBinding.inflate(*layoutInflater*)  
 setContentView(binding.*root*)  
 ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.*main\_todo*)) **{** v, insets **->** val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())  
 v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)  
 insets  
 **}** */\*\* \*/* initUI()  
 }  
  
 private fun initUI() {  
 */\*\* Inicializar el todoCategoriasAdapter \*/* todoCategoriasAdapter = TodoCategoriasAdapter(categorias)  
 binding.rvTodoCategoria.*layoutManager* =  
 LinearLayoutManager(this, LinearLayoutManager.*HORIZONTAL*, false)  
 */\*\* También le pasamos el adapte \*/* binding.rvTodoCategoria.*adapter* = todoCategoriasAdapter  
  
 */\*\* Inicializar el todoTareasAdapter \*/* todoTareasAdapter = TodoTareasAdapter() 🡨 Necesita un listado  
 }  
}

para el TodoTareasAdapter necesitamos un listado que aún no está creado, pues lo creamos:

package com.nanduky.cursokotlindesde0.todoapp  
  
import ...  
  
class TodoActivity : AppCompatActivity() {  
  
 */\*\* Crear el listado de categorías para el TodoCategoriasAdapter() \*/* private val categorias = *listOf*(  
 TodoTareaCategorias.Negocio,  
 TodoTareaCategorias.Personal,  
 TodoTareaCategorias.Otro  
  
 )  
  
 */\*\* Crear el listado de categorías para el TodoTareasAdapter() \*/* private val tareas =  
 *mutableListOf*( 🡨 Esta vez es una lista mutable porque iremos añadiendo datos  
 TodoTareas("Negocios", TodoTareaCategorias.Negocio)  
 nombreTarea, categoria:TodoCategorias, estaSeleccionado no hace falta ponerlo   
 )  
  
 */\*\* Crear el binding \*/* private lateinit var binding: ActivityTodoBinding

*/\*\* Crear un acceso al clase TodoCategoriasAdapter.kt \*/* private lateinit  
 var todoCategoriasAdapter: TodoCategoriasAdapter  
  
 */\*\* Crear un acceso al clase TodoTareasAdapter.kt \*/* private lateinit var todoTareasAdapter: TodoTareasAdapter  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *enableEdgeToEdge*()  
 */\*\* Inflar el binding con la vista \*/* binding = ActivityTodoBinding.inflate(*layoutInflater*)  
 setContentView(binding.*root*)  
 ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.*main\_todo*)) **{** v, insets **->** val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())  
 v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)  
 insets  
 **}** */\*\* \*/* initUI()  
 }  
  
 private fun initUI() {  
 */\*\* Inicializar el todoCategoriasAdapter \*/* todoCategoriasAdapter = TodoCategoriasAdapter(categorias)  
 binding.rvTodoCategoria.*layoutManager* =  
 LinearLayoutManager(this, LinearLayoutManager.*HORIZONTAL*, false)  
 */\*\* También le pasamos el adapte \*/* binding.rvTodoCategoria.*adapter* = todoCategoriasAdapter  
  
 */\*\* Inicializar el todoTareasAdapter \*/* todoTareasAdapter = TodoTareasAdapter()  
 }  
}

Y añadimos a la lista las dos categorias que faltan:

package com.nanduky.cursokotlindesde0.todoapp  
  
import android.os.Bundle  
import androidx.activity.enableEdgeToEdge  
import androidx.appcompat.app.AppCompatActivity  
import androidx.core.view.ViewCompat  
import androidx.core.view.WindowInsetsCompat  
import androidx.recyclerview.widget.LinearLayoutManager  
import com.nanduky.cursokotlindesde0.R  
import com.nanduky.cursokotlindesde0.databinding.ActivityTodoBinding  
  
class TodoActivity : AppCompatActivity() {  
  
 */\*\* Crear el listado de categorías para el TodoCategoriasAdapter() \*/* private val categorias = *listOf*(  
 TodoTareaCategorias.Negocio,  
 TodoTareaCategorias.Personal, 🡨 Nueva categoría  
 TodoTareaCategorias.Otro 🡨 Nueva categoría  
 )

Podemos importar el TodoTareasCategorias para no tener que estar refiriéndote a él continuamente. Sólo hay que poner el cursor encima y en la bombillita marcar el importar

*/\*\* Crear el listado de categorías para el TodoTareasAdapter(). \*/* private val tareas =  
 *mutableListOf*(  
 TodoTareas("Negocios", TodoTareaCategorias.Negocio),  
 TodoTareas("Personal", TodoTareaCategorias.Personal),  
 TodoTareas("Otros", TodoTareaCategorias.Otro)  
 )  
  
  
 */\*\* Crear el binding \*/* private lateinit var binding: ActivityTodoBinding  
  
 */\*\* Crear un acceso al clase TodoCategoriasAdapter.kt \*/* private lateinit  
 var todoCategoriasAdapter: TodoCategoriasAdapter  
  
 */\*\* Crear un acceso al clase TodoTareasAdapter.kt \*/* private lateinit var todoTareasAdapter: TodoTareasAdapter  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *enableEdgeToEdge*()  
 */\*\* Inflar el binding con la vista \*/* binding = ActivityTodoBinding.inflate(*layoutInflater*)  
 setContentView(binding.*root*)  
 ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.*main\_todo*)) **{** v, insets **->** val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())  
 v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)  
 insets  
 **}** */\*\* \*/* initUI()  
 }  
  
 private fun initUI() {  
 */\*\* Inicializar el todoCategoriasAdapter \*/* todoCategoriasAdapter = TodoCategoriasAdapter(categorias)  
 binding.rvTodoCategoria.*layoutManager* =  
 LinearLayoutManager(this, LinearLayoutManager.*HORIZONTAL*, false)  
 */\*\* También le pasamos el adapte \*/* binding.rvTodoCategoria.*adapter* = todoCategoriasAdapter  
  
 */\*\* Inicializar el todoTareasAdapter \*/* todoTareasAdapter = TodoTareasAdapter()  
 }  
}

Podemos importar el TodoTareasCategorias para no tener que estar repitiendo en el código la llamada para cada objeto o valor:

package com.nanduky.cursokotlindesde0.todoapp  
  
import android.os.Bundle  
import androidx.activity.enableEdgeToEdge  
import androidx.appcompat.app.AppCompatActivity  
import androidx.core.view.ViewCompat  
import androidx.core.view.WindowInsetsCompat  
import androidx.recyclerview.widget.LinearLayoutManager  
import com.nanduky.cursokotlindesde0.R  
import com.nanduky.cursokotlindesde0.databinding.ActivityTodoBinding  
import com.nanduky.cursokotlindesde0.todoapp.TodoTareaCategorias.\* 🡨 importado  
  
class TodoActivity : AppCompatActivity() {  
  
 */\*\* Crear el listado de categorías para el TodoCategoriasAdapter() \*/* private val categorias = *listOf*(  
 Negocio, 🡨 Ya no hay que llamara al TodoTareaCategorias  
 Personal, 🡨 Ya no hay que llamara al TodoTareaCategorias   
 Otro 🡨 Ya no hay que llamara al TodoTareaCategorias  
 /\* Esta es la lista que se pintará en el RecyclerView. Pinta 3 categorías, no porque sean las del TodoTareaCategorias, sino porque son las que se le pasan aquí, si hubiesen 10, pintaría 10 \*/  
  
 )  
  
 */\*\* Crear el listado de categorías para el TodoTareasAdapter(). \*/  
 /\*\* Esta vez es una lista mutable porque iremos añadiendo datos \*/* private val tareas =  
 *mutableListOf*(  
 TodoTareas("Negocios", Negocio), 🡨 TodoTareaCategoria.Negocio

TodoTareas("Personal", Personal), 🡨 TodoTareaCategoria.Personal  
 TodoTareas("Otros", Otro) 🡨 TodoTareaCategoria.Otro  
 nombreTarea, categoria:TodoCategorias, estaSeleccionado no hace falta ponerlo  
 )  
  
  
 */\*\* Crear el binding \*/* private lateinit var binding: ActivityTodoBinding  
  
 */\*\* Crear un acceso al clase TodoCategoriasAdapter.kt \*/* private lateinit  
 var todoCategoriasAdapter: TodoCategoriasAdapter  
  
 */\*\* Crear un acceso al clase TodoTareasAdapter.kt \*/* private lateinit var todoTareasAdapter: TodoTareasAdapter  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *enableEdgeToEdge*()  
 */\*\* Inflar el binding con la vista \*/* binding = ActivityTodoBinding.inflate(*layoutInflater*)  
 setContentView(binding.*root*)  
 ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.*main\_todo*)) **{** v, insets **->** val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())  
 v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)  
 insets  
 **}** */\*\* \*/* initUI()  
 }  
  
 private fun initUI() {  
 */\*\* Inicializar el todoCategoriasAdapter \*/* todoCategoriasAdapter = TodoCategoriasAdapter(categorias)  
 binding.rvTodoCategoria.*layoutManager* =  
 LinearLayoutManager(this, LinearLayoutManager.*HORIZONTAL*, false)  
 */\*\* También le pasamos el adapte \*/* binding.rvTodoCategoria.*adapter* = todoCategoriasAdapter  
  
 */\*\* Inicializar el todoTareasAdapter \*/* todoTareasAdapter = TodoTareasAdapter()  
 }  
}

Ya le podemos pasar la lista al TodoTareasAdapter() :

todoTareasAdapter = TodoTareasAdapter(tareas)

En este caso en el LayoutManager, no hay que pasarle los tres parámetros, con el contexto ya es suficiente, porque por defecto ya es orientación vertical y el boolean tampoco lo pide:

binding.rvTodoTareas.*layoutManager* =  
 LinearLayoutManager(this, LinearLayoutManager.*VERTICAL*, false) 🡨 no es necesario

binding.rvTodoTareas.*layoutManager* = LinearLayoutManager(this) 🡨 Quedaría así

Y lo mandamos al adapter:

binding.rvTodoTareas.*adapter* = todoTareasAdapter

El código completo quedaría así:

package com.nanduky.cursokotlindesde0.todoapp  
  
import android.os.Bundle  
import androidx.activity.enableEdgeToEdge  
import androidx.appcompat.app.AppCompatActivity  
import androidx.core.view.ViewCompat  
import androidx.core.view.WindowInsetsCompat  
import androidx.recyclerview.widget.LinearLayoutManager  
import com.nanduky.cursokotlindesde0.R  
import com.nanduky.cursokotlindesde0.databinding.ActivityTodoBinding  
import com.nanduky.cursokotlindesde0.todoapp.TodoTareaCategorias.Negocio  
import com.nanduky.cursokotlindesde0.todoapp.TodoTareaCategorias.Otro  
import com.nanduky.cursokotlindesde0.todoapp.TodoTareaCategorias.Personal  
  
class TodoActivity : AppCompatActivity() {  
  
 */\*\* Crear el listado de categorías para el TodoCategoriasAdapter() \*/* private val categorias = *listOf*(  
 Negocio,  
 Personal,  
 Otro  
 /\* Esta es la lista que se pintará en el RecyclerView. Pinta 3 categorías, no porque sean las del  
 TodoTareaCategorias, sino por que son las que se le pasan aquí, si hubiesen 10, pintaría 10 \*/  
  
 )  
  
 */\*\* Crear el listado de categorías para el TodoTareasAdapter(). \*/  
 /\*\* Esta vez es una lista mutable porque iremos añadiendo datos \*/* private val tareas =  
 *mutableListOf*(  
 TodoTareas("Negocios", Negocio),  
 TodoTareas("Personal", Personal),  
 TodoTareas("Otros", Otro)  
 /\* nombreTarea, categoria:TodoCategorias, estaSeleccionado no hace falta ponerlo \*/  
 )  
  
  
 */\*\* Crear el binding \*/* private lateinit var binding: ActivityTodoBinding  
  
 */\*\* Crear un acceso al clase TodoCategoriasAdapter.kt \*/* private lateinit  
 var todoCategoriasAdapter: TodoCategoriasAdapter  
  
 */\*\* Crear un acceso al clase TodoTareasAdapter.kt \*/* private lateinit var todoTareasAdapter: TodoTareasAdapter  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *enableEdgeToEdge*()  
 */\*\* Inflar el binding con la vista \*/* binding = ActivityTodoBinding.inflate(*layoutInflater*)  
 setContentView(binding.*root*)  
 ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.*main\_todo*)) **{** v, insets **->** val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())  
 v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)  
 insets  
 **}** */\*\* \*/* initUI()  
 }  
  
 private fun initUI() {  
 */\*\* Inicializar el todoCategoriasAdapter \*/* todoCategoriasAdapter = TodoCategoriasAdapter(categorias)  
 */\*\* Lo que le mandaremos al adapter \*/* binding.rvTodoCategoria.*layoutManager* =  
 LinearLayoutManager(this, LinearLayoutManager.*HORIZONTAL*, false)  
 */\*\* También le pasamos el adapte \*/* binding.rvTodoCategoria.*adapter* = todoCategoriasAdapter  
  
 */\*\* Inicializar el todoTareasAdapter \*/* todoTareasAdapter = TodoTareasAdapter(tareas)  
 */\*\* Lo que le mandaremos al adapter \*/* binding.rvTodoTareas.*layoutManager* = LinearLayoutManager(this)  
 */\*\* Lo que el adaptado le va a poner \*/* binding.rvTodoTareas.*adapter* = todoTareasAdapter  
  
 }  
}

Una buena practica es que para iniciar los listeners, se cree una función y meterla dentro del onCreate que las inicie todas.

*...*

*/\*\* \*/*initUI()  
*/\*\* Función para iniciar todos los listeners \*/*iniListeners() // Nueva llamada para iniciar los listeners

...

…

...  
// Nueva función para iniciar los listeners   
private fun iniListeners() {  
 */\*\* Para el botón flotante "fab"\*/* // nuevo listener del fab   
 binding.fabAddTarea.setOnClickListener **{** mostrarDialogo() **}**}

...

…

...  
*/\*\* Método que se lanzará al pulsar el fab \*/*// Nuevo metodo que se lanzará al pulsar el fab  
private fun mostrarDialogo(){

//...  
}

...

**Dialog:**

Para crear diálogos lo primero es crear una vista a la que asociar el dialogo.

En “res/layout” se crea un nuevo “Layout Resource File” nombrándolo dialog\_nombre.

Una vez está hecho el diseño:

<?xml version="1.0" encoding="utf-8"?>  
<androidx.appcompat.widget.LinearLayoutCompat xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="250dp"  
 android:layout\_height="wrap\_content"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:background="@color/todo\_background\_todo\_app"  
 android:orientation="vertical">  
  
 <EditText  
 android:id="@+id/et\_tarea"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:backgroundTint="@color/white"  
 android:hint="@string/todo\_add\_tarea"  
 android:inputType="text"  
 android:textColor="@color/white"  
 android:textColorHint="#A8A4A4" />  
  
 <RadioGroup  
 android:id="@+id/rg\_categorias"  
 android:checkedButton="@id/rb\_Negocios"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content">  
  
 <RadioButton  
 android:id="@+id/rb\_Negocios"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 app:buttonTint="@color/todo\_trabajo\_categoria"  
 android:text="Negocios"  
 android:textColor="@color/white" />  
  
 <RadioButton  
 android:id="@+id/rg\_personal"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 app:buttonTint="@color/todo\_personal\_categoria"  
 android:text="Personal"  
 android:textColor="@color/white" />  
  
 <RadioButton  
 android:id="@+id/rb\_otras"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 app:buttonTint="@color/todo\_otros\_categoria"  
 android:text="Otros"  
 android:textColor="@color/white" />  
  
  
 </RadioGroup>  
</androidx.appcompat.widget.LinearLayoutCompat>

Volvemos al código en “TodoActivity” en el método mostrarDialogo()

Creamos el dialogo para que se muestre, de momento sólo se muestra, no hace nada:

*/\*\* Método que se lanzará al pulsar el fab \*/*   
private fun mostrarDialogo(){  
 */\*\* Creamos una variable\*/* val dialogo = Dialog(this)  
 */\*\* Ponemos la vista al dialogo \*/* dialogo.setContentView(R.layout.*dialogo\_tarea*)  
 */\*\* Llamar al dialogo \*/* dialogo.show()  
}

Añadimos los componentes que vamos a usar y creamos el setOnclickListener:

*/\*\* Método que se lanzará al pulsar el fab \*/*   
private fun mostrarDialogo() {  
 */\*\* Creamos una variable\*/* val dialogo = Dialog(this)*/\*\* Ponemos la vista al dialogo \*/* dialogo.setContentView(R.layout.*dialogo\_tarea*)

*/\*\* hacemos el cast a los componentes, ya que con el binding queda desformado \*/* val btnAddTarea: Button = dialogo.findViewById(R.id.*btn\_add\_tarea*) 🡨 Boton  
 val etTarea:EditText = dialogo.findViewById(R.id.*et\_tarea*) 🡨 EditText  
 val rgCategorias:RadioGroup = dialogo.findViewById(R.id.*rg\_categorias*) 🡨 RB

*/\*\* Captura el click del radioBroup \*/* btnAddTarea.setOnClickListener **{** 🡨 captura el click  
  
 **}**  
 */\*\* Llamar al dialogo \*/* dialogo.show()  
}

…

*/\*\* Método que se lanzará al pulsar el fab \*/*  
private fun mostrarDialogo() {  
 */\*\* Creamos una variable\*/* val dialogo = Dialog(this)

*/\*\* Ponemos la vista al dialogo \*/* dialogo.setContentView(R.layout.*dialogo\_tarea*)

*/\*\* hacemos el cast a los componentes, ya que con el binding queda desformado \*/* val btnAddTarea: Button = dialogo.findViewById(R.id.*btn\_add\_tarea*)   
 val etTarea:EditText = dialogo.findViewById(R.id.*et\_tarea*)   
 val rgCategorias:RadioGroup = dialogo.findViewById(R.id.*rg\_categorias*)

*/\*\* Captura el click del radioBroup \*/* btnAddTarea.setOnClickListener **{** //  
 /\* Guardar en una variable el id del RB (RadioButton) seleccionado \*/  
 val selectId = rgCategorias.*checkedRadioButtonId*

**}**   
 */\*\* Llamar al dialogo \*/* dialogo.show()  
}

…

*/\*\* Método que se lanzará al pulsar el fab \*/*  
private fun mostrarDialogo() {  
 */\*\* Creamos una variable\*/* val dialogo = Dialog(this)

*/\*\* Ponemos la vista al dialogo \*/* dialogo.setContentView(R.layout.*dialogo\_tarea*)

*/\*\* hacemos el cast a los componentes, ya que con el binding queda desformado \*/* val btnAddTarea: Button = dialogo.findViewById(R.id.*btn\_add\_tarea*)   
 val etTarea:EditText = dialogo.findViewById(R.id.*et\_tarea*)   
 val rgCategorias:RadioGroup = dialogo.findViewById(R.id.*rg\_categorias*)

*/\*\* Captura el click del radioBroup \*/* dialogBinding.btnAddTarea.setOnClickListener **{** //  
 val selectId = rgCategorias.*checkedRadioButtonId* /\* Acceder a un RB dentro del grupo y guardar el la variable el RB seleccionado\*/  
 val selectRadioButtom:RadioButton = dialogBinding.rgCategorias.

findViewById(selectId)

**}**  
  
 */\*\* Llamar al dialogo \*/* dialogo.show()  
}

…

*/\*\* Método que se lanzará al pulsar el fab \*/*  
private fun mostrarDialogo() {  
 dialogBinding = DialogoTareaBinding.inflate(*layoutInflater*)  
 */\*\* Creamos una variable\*/* val dialogo = Dialog(this)

*/\*\* Ponemos la vista al dialogo \*/* dialogo.setContentView(R.layout.*dialogo\_tarea*)

*/\*\* hacemos el cast a los componentes, ya que con el binding queda desformado \*/* val btnAddTarea: Button = dialogo.findViewById(R.id.*btn\_add\_tarea*)   
 val etTarea:EditText = dialogo.findViewById(R.id.*et\_tarea*)   
 val rgCategorias:RadioGroup = dialogo.findViewById(R.id.*rg\_categorias*)

*/\*\* Captura el click del radioBroup \*/* dialogBinding.btnAddTarea.setOnClickListener **{** //  
 val selectId = dialogBinding.rgCategorias.*checkedRadioButtonId*  
 val selectRadioButtom:RadioButton = dialogBinding.rgCategorias.findViewById(selectId)  
 /\* Comprobar que RB ha sido seleccionado a través del texto y guardar en la variable la categoría \*/  
 val categoriaActual: TodoTareaCategorias = when (selectRadioButtom.*text*) {  
 "Negocio" -> Negocio  
 "Personal" -> Personal  
 else -> Otro  
 }  
  
 **}**  
 */\*\* Llamar al dialogo \*/* dialogo.show()  
}

Ahora en la mutableListOf de tareas podemos añadirle la nueva tarea con la categoría que se seleccionó en el RadioButton y con el texto que hay en el EditText y cerramos el dialogo

*/\*\* Método que se lanzará al pulsar el fab \*/*// Nuevo metodo que se lanzará al pulsar el fab (BORRAR)  
private fun mostrarDialogo() {  
 */\*\* Creamos una variable\*/* val dialogo = Dialog(this)  
 */\*\* Ponemos la vista al dialogo \*/* dialogo.setContentView(R.layout.*dialogo\_tarea*)  
  
 */\*\* hacemos el cast a los componentes, ya que con el binding queda desformado \*/* val btnAddTarea: Button = dialogo.findViewById(R.id.*btn\_add\_tarea*) //  
 val etTarea:EditText = dialogo.findViewById(R.id.*et\_tarea*) //  
 val rgCategorias:RadioGroup = dialogo.findViewById(R.id.*rg\_categorias*) //  
  
 */\*\* Captura el click del radioBroup \*/* btnAddTarea.setOnClickListener **{** //  
 */\*\* Guardar un una variable el id del RB (RadioButton) seleccionado \*/* val selectId = rgCategorias.*checkedRadioButtonId  
  
 /\*\* Acceder a un RB dentro del grupo y guardar el la variable el RB seleccionado\*/* val selectRadioButtom: RadioButton = rgCategorias.findViewById(selectId)  
  
 */\*\* Comprobar que RB ha sido seleccionado a través del texto y guardar en la variable  
 \* la categoría \*/* val categoriaActual: TodoTareaCategorias = when (selectRadioButtom.*text*) {  
 "Negocio" -> Negocio  
 "Personal" -> Personal  
 else -> Otro  
 }  
 /\* Añadir la tarea a la mutableListOf de tareas\*/  
 tareas.add(TodoTareas(etTarea.*text*.toString(), categoriaActual))   
 /\* Tras pulsar el botón de añadir tarea,cerramos el dialogo \*/  
 dialogo.hide()

**}** */\*\* Llamar al dialogo \*/* dialogo.show()  
}

Con esta añadimos al listado la tarea nueva creada.

En este punto, aun no se ha actualizado la tarea, para esto hay que avisar al adapter de que la lista ha sido actualizada, así que creamos una función que avisará al adapter y le la llamamos desde el dialogo:

*/\*\* Método que se lanzará al pulsar el fab \*/*  
private fun mostrarDialogo() {val dialogo = Dialog(this)dialogo.setContentView(R.layout.*dialogo\_tarea*)  
val btnAddTarea: Button = dialogo.findViewById(R.id.*btn\_add\_tarea*)   
 val etTarea: EditText = dialogo.findViewById(R.id.*et\_tarea*)   
 val rgCategorias: RadioGroup = dialogo.findViewById(R.id.*rg\_categorias*)   
btnAddTarea.setOnClickListener **{** val selectId = rgCategorias.*checkedRadioButtonId* val selectRadioButtom: RadioButton = rgCategorias.findViewById(selectId)val categoriaActual: TodoTareaCategorias = when (selectRadioButtom.*text*) {  
 "Negocio" -> Negocio  
 "Personal" -> Personal  
 else -> Otro  
 }tareas.add(TodoTareas(etTarea.*text*.toString(), categoriaActual))  
 /\* Llamamos a la función que actualiza la lista del adapter\*/  
 updateTarea()dialogo.hide()  
 **}**dialogo.show()  
}  
  
/\* Función para actualizar la lista que tiene el adapter\*/  
private fun updateTarea(){ todoTareasAdapter.notifyDataSetChanged()}

Para optimizar el a través de los valores strings when:

val categoriaActual: TodoTareaCategorias = when (selectRadioButtom.*text*) {  
 getString(R.string.*todo\_negocio*) -> Negocio  
 getString(R.string.*todo\_personal*) -> Personal  
 else -> Otro  
}

Para evitar que se añadan tareas sin que se haya rellenado el editText

El siguiente paso es poder capturar los click’s que se hacen el los CheckBox y cada item tiene que tener su propio click, para esto están las funciones “**lambda**”.

Para modificar el item, hay que crear una función que recibirá una posición, que será un Int:

/\* Para modificar los items de las tareas, se crea una función que recibirá una posición (Int) \*/  
private fun onItemSeleccionado(posicion: Int) {  
 /\* cambiar de seleccionado a no seleccionado a través del Boolean del TodoTareas\*/  
 tareas[posicion].estaSelect = !tareas[posicion].estaSelect   
 /\* Notifecar al adapter que hanhabido cambios \*/  
 updateTarea()

}

onItemSeleccionado será llamado cuando se pulse un Item, y tendrá que poner lo contrarío de si está seleccionado o no del item pulsado. Luego se lo notificamos al adapter (todoTareasAdapter)

Volvemos al TodoTareasAdapter y le pasamos un segundo parámetro para poder controlar si está checado o no:

import android.view.LayoutInflater  
import android.view.ViewGroup  
import androidx.recyclerview.widget.RecyclerView  
import com.nanduky.cursokotlindesde0.R  
  
/\* Le pasamos a la clase un segundo parámetro para el check del item \*/

class TodoTareasAdapter(private val tareas: List<TodoTareas>, private val onTareaSeleccionado:(Int) -> Unit) :  
 RecyclerView.Adapter<TodoTareasViewHolder>() {  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TodoTareasViewHolder {  
 */\*\* Creamos un inflador de vistas. El context lo extraemos del ViewGroup \*/* val nuevaVista =  
 LayoutInflater.from(parent.*context*).inflate(R.layout.*item\_todo\_tarea*, parent, false)  
 return TodoTareasViewHolder(nuevaVista)  
 }  
  
 override fun getItemCount() = tareas.size  
  
 override fun onBindViewHolder(holder: TodoTareasViewHolder, position: Int) {  
 holder.render(tareas[position])  
 }  
}

Si reformateamos el código con (Ctrl + Alt + L) quedaría así:

class TodoTareasAdapter(  
 private val tareas: List<TodoTareas>,  
 private val onTareaSeleccionado: (Int) -> Unit 🡨 variable nueva tipo lambda  
) :  
 RecyclerView.Adapter<TodoTareasViewHolder>() {  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): TodoTareasViewHolder {  
 */\*\* Creamos un inflador de vistas. El context lo extraemos del ViewGroup \*/* val nuevaVista =  
 LayoutInflater.from(parent.*context*).inflate(R.layout.*item\_todo\_tarea*, parent, false)  
 return TodoTareasViewHolder(nuevaVista)  
 }  
  
 override fun getItemCount() = tareas.size  
  
 override fun onBindViewHolder(holder: TodoTareasViewHolder, position: Int) {  
 holder.render(tareas[position])  
 }

Ahora en el onBindViewHolder llamamos a la función creada lambda:

override fun onBindViewHolder(holder: TodoTareasViewHolder, position: Int) {  
 holder.render(tareas[position])  
 /\* Llamar a la función lambda y le pasamos la posición \*/  
 onTareaSeleccionada(position)  
}

onTareaSeleccionada(position) significa que la función lambda a sido ejecutada

private val onTareaSeleccionada: (Int) -> Unit Esto pregunta que función lada se le pasó.

Y en el TodoActivity, donde se inicializa el adapter, se le pasa la función que mandamos la TodoTareasAdspter:

*/\*\* Inicializar el todoTareasAdapter \*/*todoTareasAdapter = TodoTareasAdapter(tareas , onItemSeleccionado ())

Pero esto así no funciona, porque así lo que se hace es llamarlo directamente y se necesita que se ejecute en el lambda

*/\*\* Inicializar el todoTareasAdapter \*/*todoTareasAdapter = TodoTareasAdapter(tareas, **{** onItemSeleccionado(**it**) **}**)

Para que sea más fácil de entender, al (it) le pondremos un nombre:

*/\*\* Inicializar el todoTareasAdapter \*/*todoTareasAdapter = TodoTareasAdapter(tareas, **{** posicionIt **->** onItemSeleccionado(posicionIt) **}**)

Se aconseja sacar la funcion fuera de los paréntesis

*/\*\* Inicializar el todoTareasAdapter \*/*todoTareasAdapter = TodoTareasAdapter(tareas) **{** posicionIt **->** onItemSeleccionado(posicionIt) **}**

Aquí le estamos pasando una función que va a ser llamada cuando alguien llame a:

private val onTareaSeleccionada: (Int) -> Unit

del adapter que en este caso se le llama desde:

override fun onBindViewHolder(holder: TodoTareasViewHolder, position: Int) {  
 holder.render(tareas[position])  
 */\*\* Llamar a la función lambda pasandole la posición \*/* onTareaSeleccionada(position) 🡨 llamada desde aquí  
}

Pero llamando desde ahí, siempre se va a ejecutar al crear un item y siempre se va a marcar, así que le decimos la holder que cuando se pulse en el item, en cualquier parte del item (itemView) que lance la función:

override fun onBindViewHolder(holder: TodoTareasViewHolder, position: Int) {  
 holder.render(tareas[position])  
 /\* Llamar a la función lambda pasandole la posición cuando se haga click en el item\*/  
 holder.itemView.setOnClickListener**{**onTareaSeleccionada(position) **}**}

Esto de momento no funciona, porque en el “TodoTareasViewHolder” no estamos haciendo nada solo pintando el nombre:

import android.view.View  
import androidx.recyclerview.widget.RecyclerView  
import com.nanduky.cursokotlindesde0.databinding.ItemTodoTareaBinding  
  
class TodoTareasViewHolder(view: View ):RecyclerView.ViewHolder(view) {  
  
 */\*\* Creamos el binding \*/* private val binding = ItemTodoTareaBinding.bind(view)   
  
 fun render(tarea : TodoTareas){  
 binding.tvTodoTarea.*text* = tarea.nombreTarea 🡨 sólo pinta el nombre  
 }  
}

Para que se tache el texto según se pulsa:

import android.graphics.Paint  
import android.view.View  
import androidx.recyclerview.widget.RecyclerView  
import com.nanduky.cursokotlindesde0.databinding.ItemTodoTareaBinding  
  
class TodoTareasViewHolder(view: View ):RecyclerView.ViewHolder(view) {  
 */\*\* Creamos el binding \*/* private val binding = ItemTodoTareaBinding.bind(view)  
  
 fun render(tarea : TodoTareas){  
 /\* Variable para recoger el binding y no tener que ponerlo toto el rato \*/  
 val tvTodoTarea = binding.tvTodoTarea  
  
 /\* Hacer que se tache el nombre al hacer click \*/  
 if(tarea.estaSelect){  
 tvTodoTarea.*paintFlags* = tvTodoTarea.*paintFlags* or Paint.*STRIKE\_THRU\_TEXT\_FLAG* } else{ // Para que se destache  
 tvTodoTarea.*paintFlags* = tvTodoTarea.*paintFlags* and Paint.*STRIKE\_THRU\_TEXT\_FLAG*.inv()

}  
 tvTodoTarea.*text* = tarea.nombreTarea  
 }  
}

Para que se parque el checkBox cuando se pulsa en alguna parte el item:

/\* Para marcar el checkBox aunque pulse fuera, en otra parte del item \*/  
cbTodoTarea.*isChecked* = tarea.estaSelect

Para poner un color al checkBox:

/\* Para pintar un color en el checkBox \*/  
cbTodoTarea.*buttonTintList* = ColorStateList.valueOf(  
 ContextCompat.getColor(cbTodoTarea.*context*, R.color.*todo\_trabajo\_categoria*)  
)

Pero esto pintaría todos los checkBox del mismo color, para solucionarlo hacemos un when para comprobar que categoría es, como las categorias están en la “sealed class”, el when tiene que ser exhaustivo, en la bombillita marcamos añadir branches

when(tarea.categorias){  
 TodoTareaCategorias.Negocio -> *TODO*()  
 TodoTareaCategorias.Otro -> *TODO*()  
 TodoTareaCategorias.Personal -> *TODO*()  
}

Importamos “TodoTareaCategorias:

when(tarea.categorias){  
 Negocio -> *TODO*()  
 Otro -> *TODO*()  
 Personal -> *TODO*()  
}

Y le ponemos un color a cada categoría:

/\* Para pintar un color en el checkBox \*/  
when (tarea.categorias) {  
 Negocio -> {  
 cbTodoTarea.*buttonTintList* = ColorStateList.valueOf(  
 ContextCompat.getColor(cbTodoTarea.*context*, R.color.*todo\_trabajo\_categoria*) )  
 }  
 Otro -> {  
 cbTodoTarea.*buttonTintList* = ColorStateList.valueOf(  
 ContextCompat.getColor(cbTodoTarea.*context*, R.color.*todo\_otros\_categoria*)  
 )  
 }  
 Personal -> {  
 cbTodoTarea.*buttonTintList* = ColorStateList.valueOf(  
 ContextCompat.getColor(cbTodoTarea.*context*, R.color.*todo\_personal\_categoria*)  
 )  
 }  
}

Pero esto es mala practica, ya que hay mucho código repetido y esto se puede mejorar:

/\* Para pintar un color en el checkBox \*/  
 val colorTarea = when (tarea.categorias) {  
 Negocio -> R.color.*todo\_trabajo\_categoria* Otro -> R.color.*todo\_otros\_categoria* Personal -> R.color.*todo\_personal\_categoria* }  
 cbTodoTarea.*buttonTintList* =  
 ColorStateList.valueOf(ContextCompat.getColor(cbTodoTarea.*context*, colorTarea))  
}

Para seleccionar una categoría y que solo aparezcan las tareas de esa categoría, hay que hacer algo parecido a lo anterior en el “TodoCategoriasViewHolder” que en este momento esta así:

package com.nanduky.cursokotlindesde0.todoapp  
  
import android.view.View  
import androidx.core.content.ContextCompat  
import androidx.recyclerview.widget.RecyclerView  
import com.nanduky.cursokotlindesde0.R  
import com.nanduky.cursokotlindesde0.databinding.ItemTodoTareaCategoriasBinding  
  
class TodoCategoriasViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
  
 */\*\* creamos el binding \*/* private val binding = ItemTodoTareaCategoriasBinding.bind(view)  
  
 */\*\* Esta función recibe una tareaCategoría y una string\*/* fun render(tareaCategorias: TodoTareaCategorias) {  
 */\*\* Se crea un when que recorra las categorías de la sealed class\*/* when (tareaCategorias) {  
 TodoTareaCategorias.Negocio -> {  
 // El texto de la categoría  
 binding.tvNombreCategoria.*text* =  
 binding.tvNombreCategoria.*context*.getString(R.string.*todo\_negocio*)  
 // El color de la línea divisoría  
 binding.lineaDivisoria.setBackgroundColor(  
 ContextCompat.getColor(  
 binding.lineaDivisoria.*context*,  
 R.color.*todo\_trabajo\_categoria* )  
 )  
 }  
  
 TodoTareaCategorias.Otro -> {  
 binding.tvNombreCategoria.*text* =  
 binding.tvNombreCategoria.*context*.getString(R.string.*todo\_otros*)  
 binding.lineaDivisoria.setBackgroundColor(  
 ContextCompat.getColor(  
 binding.lineaDivisoria.*context*,  
 R.color.*todo\_personal\_categoria* )  
 )  
 }  
  
 TodoTareaCategorias.Personal -> {  
 binding.tvNombreCategoria.*text* =  
 binding.tvNombreCategoria.*context*.getString(R.string.*todo\_personal*)  
 binding.lineaDivisoria.setBackgroundColor(  
 ContextCompat.getColor(  
 binding.lineaDivisoria.*context*,  
 R.color.*todo\_otros\_categoria* )  
 )  
 }  
 }  
 //  
 }  
}

que esto, siguiendo el ejemplo del “TodoTareasViewHolder” se puede optimizar mucho.

Empezando por crear unas variables de los componentes del binding para no tener que esta repitiendo todo el rato el binding.loquesea….

/\* Crear variables para los componentes del binding \*/  
val tvNombreCategoria = binding.tvNombreCategoria  
val lineaDivisoria = binding.lineaDivisoria

Las creo con el mismo nombre del componente para que sea más cómodo llamarlas sin que haya errores de que es cada cosa.

Se crea una variable que guardará el nombre de la categoría, a través de un when que accederá a la “sealed class”, y también será exhaustiva (bombillita y añadir branches:

val nombreCategorias = when (tareaCategorias){  
 TodoTareaCategorias.Negocio -> *TODO*()  
 TodoTareaCategorias.Otro -> *TODO*()  
 TodoTareaCategorias.Personal -> *TODO*()  
}

Añadimos los nombres de las categorias que están en “values/string”:

val nombreCategorias = when (tareaCategorias){  
 TodoTareaCategorias.Negocio -> R.string.*todo\_negocio* TodoTareaCategorias.Otro -> R.string.*todo\_otros* TodoTareaCategorias.Personal -> R.string.*todo\_personal*}

Le ponemos el nombre al item según su categoría:

/\* Variable que guardará el nombre de la categoría\*/  
val nombreCategorias = when (tareaCategorias){  
 TodoTareaCategorias.Negocio -> R.string.*todo\_negocio* TodoTareaCategorias.Otro -> R.string.*todo\_otros* TodoTareaCategorias.Personal -> R.string.*todo\_personal*}  
/\* Pasar al item el nombre de la categoría guardado en la variable \*/  
tvNombreCategoria.*text* = tvNombreCategoria.*context*.getString(nombreCategorias)

Creamos una segunda variable para guardar el color de la línea divisoria con el mismo proceso que para el nombre de la categoría:

/\* Variable que guardará el color de la linea divisoría \*/  
val colorDivisor = when (tareaCategorias){  
 TodoTareaCategorias.Negocio -> R.color.*todo\_trabajo\_categoria* TodoTareaCategorias.Otro -> R.color.*todo\_otros\_categoria* TodoTareaCategorias.Personal -> R.color.*todo\_personal\_categoria*}  
/\* Pasar el color al item guardado en la variable \*/  
lineaDivisoria.setBackgroundColor(ContextCompat.getColor(lineaDivisoria.*context*,colorDivisor))

Si importamos “TodoTareaCategorias”, no hay que ponerlo cada vez

Y quedaría así:

package com.nanduky.cursokotlindesde0.todoapp  
  
import android.view.View  
import androidx.core.content.ContextCompat  
import androidx.recyclerview.widget.RecyclerView  
import com.nanduky.cursokotlindesde0.R  
import com.nanduky.cursokotlindesde0.databinding.ItemTodoTareaCategoriasBinding  
import com.nanduky.cursokotlindesde0.todoapp.TodoTareaCategorias.\*  
  
class TodoCategoriasViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
  
 */\*\* creamos el binding \*/* private val binding = ItemTodoTareaCategoriasBinding.bind(view)  
  
 */\*\* Esta función recibe una tareaCategoría y una string\*/* fun render(tareaCategorias: TodoTareaCategorias) {  
 /\* Crear variables para los componentes del binding \*/  
 val tvNombreCategoria = binding.tvNombreCategoria  
 val lineaDivisoria = binding.lineaDivisoria  
  
 */\*\* Se crea dos when que recorran las categorías de la sealed class\*/* /\* Variable que guardará el nombre de la categoría\*/  
 val nombreCategorias = when (tareaCategorias) {  
 Negocio -> R.string.*todo\_negocio* Otro -> R.string.*todo\_otros* Personal -> R.string.*todo\_personal* }  
 /\* Pasar al item el nombre de la categoría guardado en la variable \*/  
 tvNombreCategoria.*text* = tvNombreCategoria.*context*.getString(nombreCategorias)  
  
 /\* Variable que guardará el color de la linea divisoría \*/  
 val colorDivisor = when (tareaCategorias) {  
 Negocio -> R.color.*todo\_trabajo\_categoria* Otro -> R.color.*todo\_otros\_categoria* Personal -> R.color.*todo\_personal\_categoria* }  
 /\* Pasar el color al item guardado en la variable \*/  
 lineaDivisoria.setBackgroundColor(  
 ContextCompat.getColor(  
 lineaDivisoria.*context*,  
 colorDivisor  
 )  
 )  
 }  
}

Primero creamos la función “lambda” en el “TotoCategoriasAdapter” y de esto:

class TodoCategoriasAdapter(private val categoria: List<TodoTareaCategorias>) :  
 RecyclerView.Adapter<TodoCategoriasViewHolder>() {

pasamos a esto:

class TodoCategoriasAdapter(  
 private val categoria: List<TodoTareaCategorias>,  
 private val onCategoriaSeleccionada: (Int) -> Unit  
) :  
 RecyclerView.Adapter<TodoCategoriasViewHolder>() {

En vez de hacer un setOnClockListener en el “onBindViewHolder” como se hizo en el “TodoTareasAdapter”, se puede mandar la función “lambda” más lejos.

En el onBiendViewHolder, la función “render”, la que recibe la “tareaCategoria” del “TodoCategoriasViewHolder” se le puede pasar el “onCategoriaSeleccionada” si los paréntesis “()”, porque con los paréntesis, se le está llamando, y sin los paréntesis, se le está enviando.

override fun onBindViewHolder(holder: TodoCategoriasViewHolder, position: Int) {  
holder.render(categoria[position], onCategoriaSeleccionada)  
}

Esto dará un error ya que la función render sólo pedía un parámetro y con esto le estamos mandando un segundo parámetro. Esto se arregla con la bombillita y en add parameter. Ahora en la función “render” de “TodoCategoriaViewHolder” se le ha añadido un parámetro nuevo de la función “lambda”

fun render(tareaCategorias: TodoTareaCategorias, onCategoriaSeleccionada: (Int) -> Unit) {

Siguiendo en el “TodoCategoriaViewHolder” en la función render, añadimos al item seleccionado un setOnClickListener:

itemView.setOnClickListener**{ }**

Y le que al pulsar el item lance la función Lambda:

itemView.setOnClickListener**{** onCategoriaSeleccionada() **}**

Y para saber la posición del item le pasamos el “latoutPosition” que nos da la posición:

itemView.setOnClickListener**{** onCategoriaSeleccionada(*layoutPosition*) **}**

Para poder comprobar si se seleccionó una categoría, hay que comprobarlo con un boleano, el problema es que son objetos de la “sealed class” y no pueden tener parámetros al menos que se conviertan en “data class” o si todos los objetos tienen los mismos atributos, se le puede poner un paramentro a la misma “sealed class”:

sealed class TodoTareaCategorias(var estaSeleccionado: Boolean = true) {  
 object Personal : TodoTareaCategorias()  
 object Negocio : TodoTareaCategorias()  
 object Otro : TodoTareaCategorias()  
}

En el “TodoCategoriasViewHolder” se crea una variable que almacenará el color después de comprueba que categoría a sido selecionada y le pasamos un color:

/\* Comprobamos que categoría a sido seleccionada \*/  
val color = if(tareaCategorias.estaSeleccionado){  
 R.color.*todo\_background\_card*}  
else {  
 R.color.*todo\_background\_disabled*}

Se necesita el nombre de la cardView para modificar el color, pero al diseñarla no se le puso, así que hay que ir al “item\_todo\_tares\_categorias.xml” ponerle ina id:

<?xml version="1.0" encoding="utf-8"?>  
<androidx.cardview.widget.CardView 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/contenedor\_Vista" 🡨 id de la cardView  
 android:layout\_width="160dp"

...

Se le pasa el color al CradView:

/\* Añadimos el color a la CardView seleccionada\*/  
contenedoVista.setBackgroundColor(

ContextCompat.getColor(contenedoVista.*context*,colorSeleccion))

El “TodoCategoriasViewHolder” queda así ahora mismo:

package com.nanduky.cursokotlindesde0.todoapp  
  
import android.view.View  
import androidx.core.content.ContextCompat  
import androidx.recyclerview.widget.RecyclerView  
import com.nanduky.cursokotlindesde0.R  
import com.nanduky.cursokotlindesde0.databinding.ItemTodoTareaCategoriasBinding  
import com.nanduky.cursokotlindesde0.todoapp.TodoTareaCategorias.Negocio  
import com.nanduky.cursokotlindesde0.todoapp.TodoTareaCategorias.Otro  
import com.nanduky.cursokotlindesde0.todoapp.TodoTareaCategorias.Personal  
  
class TodoCategoriasViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
  
 */\*\* creamos el binding \*/* private val binding = ItemTodoTareaCategoriasBinding.bind(view)  
  
 */\*\* Esta función recibe una tareaCategoría y una función lambda\*/* fun render(tareaCategorias: TodoTareaCategorias, onCategoriaSeleccionada: (Int) -> Unit) {  
 */\*\* Crear variables para los componentes del binding \*/* val tvNombreCategoria = binding.tvNombreCategoria  
 val lineaDivisoria = binding.lineaDivisoria  
 val contenedoVista = binding.contenedorVista  
  
 /\* 02 Comprobamos que categoría a sido seleccionada \*/  
 val colorSeleccion = if (tareaCategorias.estaSeleccionado) {  
 R.color.*todo\_background\_card* } else {  
 R.color.*todo\_background\_disabled* }  
  
 /\* 03 Añadimos el color a la CardView seleccionada\*/  
 contenedoVista.setBackgroundColor(  
 ContextCompat.getColor(  
 contenedoVista.*context*,  
 colorSeleccion  
 )  
 )  
  
 /\* 01 Capturar el click del item seleccionado\*/  
 itemView.setOnClickListener **{** onCategoriaSeleccionada(*layoutPosition*) **}** */\*\* Se crea dos when que recorran las categorías de la sealed class\*/  
  
 /\*\* Variable que guardará el nombre de la categoría\*/* val nombreCategorias = when (tareaCategorias) {  
 Negocio -> R.string.*todo\_negocio* Otro -> R.string.*todo\_otros* Personal -> R.string.*todo\_personal* }  
 */\*\* Pasar al item el nombre de la categoría guardado en la variable \*/* tvNombreCategoria.*text* = tvNombreCategoria.*context*.getString(nombreCategorias)  
  
 */\*\* Variable que guardará el color de la linea divisoría \*/* val colorDivisor = when (tareaCategorias) {  
 Negocio -> R.color.*todo\_trabajo\_categoria* Otro -> R.color.*todo\_otros\_categoria* Personal -> R.color.*todo\_personal\_categoria* }  
 */\*\* Pasar el color al item guardado en la variable \*/* lineaDivisoria.setBackgroundColor(  
 ContextCompat.getColor(  
 lineaDivisoria.*context*,  
 colorDivisor  
 )  
 )  
 }  
}

En el “TodoActivity”, “todoCategoriasAdapte” nos pide el segundo parámetro, que es la función “lambda” que aún no está hecha, y hay que crear una nueva función:

private fun initUI() {  
 */\*\* Inicializar el todoCategoriasAdapter \*/*

todoCategoriasAdapter =TodoCategoriasAdapter(categorias) 🡨 Pide 2º parámetro

Creamos la función:

private fun actualizaCategoria(posicion : Int){}

Ahora hay que acceder al listado de las categorías “private val categorías= listOf(...)

Y modificar el valor de “estaSeleccionado” (De la sealed class) según se pulsa:

/\* Función para cambiar la selección del item \*/  
private fun actualizaCategoria(posicion : Int){  
 /\* pon lo opuesto de lo que había \*/  
 categorias[posicion].estaSeleccionado = !categorias[posicion].estaSeleccionado

Aquí cómo sólo se va a modificar un item se puede obtimizar para que no actualize todo la lista (notifyDataSetChanged) y sólo actualice el item seleccionado notifyItemChanged():

*/\*\* Función para cambiar la selección del item \*/*private fun actualizaCategoria(posicion : Int){  
 */\*\* pon lo opuesto de lo que había \*/* categorias[posicion].estaSeleccionado = !categorias[posicion].estaSeleccionado  
 /\* Actualizar el item seleccionado \*/  
 todoCategoriasAdapter.notifyItemChanged(posicion)  
}

Le pasamos la función al “todoCategoriaAdapter”:

private fun initUI() {  
 */\*\* Inicializar el todoCategoriasAdapter \*/*//todoCategoriasAdapter =TodoCategoriasAdapter(categorias) {actualizaCategoria(it)}  
 todoCategoriasAdapter =  
 TodoCategoriasAdapter(categorias) **{** posicionIt **->** actualizaCategoria(posicionIt) **}**

**...**

Al ejecutar la app, se ve que lo CardView del recycler del las categorías, se ha quedado sin las esquinas redondas. El problema reside en la forma de añadir el color:

package com.nanduky.cursokotlindesde0.todoapp  
  
import ...  
  
class TodoCategoriasViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
  
 */\*\* creamos el binding \*/* private val binding = ItemTodoTareaCategoriasBinding.bind(view)  
  
 */\*\* Esta función recibe una tareaCategoría y una función lambda\*/* fun render(tareaCategorias: TodoTareaCategorias, onCategoriaSeleccionada: (Int) -> Unit) {  
 */\*\* Crear variables para los componentes del binding \*/* val tvNombreCategoria = binding.tvNombreCategoria  
 val lineaDivisoria = binding.lineaDivisoria  
 val contenedoVista = binding.contenedorVista  
  
 */\*\* Comprobamos que categoría a sido seleccionada \*/* val colorSeleccion = if (tareaCategorias.estaSeleccionado) {  
 R.color.*todo\_background\_card* } else {  
 R.color.*todo\_background\_disabled* }

*/\*\* Añadimos el color a la CardView seleccionada\*/* contenedoVista.setBackgroundColor( 🡨 el error es este (setCardBackgroudColor)  
 ContextCompat.getColor(  
 contenedoVista.*context*,  
 colorSeleccion  
 )  
 )  
  
 */\*\* Capturar el click del item seleccionado\*/* itemView.setOnClickListener **{** onCategoriaSeleccionada(*layoutPosition*) **}** */\*\* Se crea dos when que recorran las categorías de la sealed class\*/  
  
 /\*\* Variable que guardará el nombre de la categoría\*/* val nombreCategorias = when (tareaCategorias) {  
 Negocio -> R.string.*todo\_negocio* Otro -> R.string.*todo\_otros* Personal -> R.string.*todo\_personal* }  
 */\*\* Pasar al item el nombre de la categoría guardado en la variable \*/* tvNombreCategoria.*text* = tvNombreCategoria.*context*.getString(nombreCategorias)  
  
 */\*\* Variable que guardará el color de la linea divisoría \*/* val colorDivisor = when (tareaCategorias) {  
 Negocio -> R.color.*todo\_trabajo\_categoria* Otro -> R.color.*todo\_otros\_categoria* Personal -> R.color.*todo\_personal\_categoria* }  
 */\*\* Pasar el color al item guardado en la variable \*/* lineaDivisoria.setBackgroundColor(  
 ContextCompat.getColor(  
 lineaDivisoria.*context*,  
 colorDivisor  
 )  
 )  
 }  
}

Al ser un CardView, no se usa el “setBackgrounColor” sino el “ setCardBackgroundColor”:

Ahora hay que filtra las tareas según estén las categorías habilitadas o deshabilitadas:

En “TodoActivity” en la función “actualizarCategoria” llamamos al al función “updateTarea()” que será la que se va a modificar para quinte los resultados según las categorías habilitadas o des habilitadas:

import android.app.Dialog  
import android.os.Bundle  
import android.widget.Button  
import android.widget.EditText  
import android.widget.RadioButton  
import android.widget.RadioGroup  
import android.widget.Toast  
import androidx.activity.enableEdgeToEdge  
import androidx.appcompat.app.AppCompatActivity  
import androidx.core.view.ViewCompat  
import androidx.core.view.WindowInsetsCompat  
import androidx.recyclerview.widget.LinearLayoutManager  
import com.nanduky.cursokotlindesde0.R  
import com.nanduky.cursokotlindesde0.databinding.ActivityTodoBinding  
import com.nanduky.cursokotlindesde0.todoapp.TodoTareaCategorias.Negocio  
import com.nanduky.cursokotlindesde0.todoapp.TodoTareaCategorias.Otro  
import com.nanduky.cursokotlindesde0.todoapp.TodoTareaCategorias.Personal  
  
class TodoActivity : AppCompatActivity() {  
  
 */\*\* Crear el listado de categorías para el TodoCategoriasAdapter() \*/* private val categorias = *listOf*(  
 Negocio,  
 Personal,  
 Otro  
 */\*\* Esta es la lista que se pintará en el RecyclerView. Pinta 3 categorías, no porque sean las del  
 TodoTareaCategorias, sino por que son las que se le pasan aquí, si hubiesen 10, pintaría 10 \*/* )  
  
 */\*\* Crear el listado de categorías para el TodoTareasAdapter(). \*/  
 /\*\* Esta vez es una lista mutable porque iremos añadiendo datos \*/* private val tareas =  
 *mutableListOf*(  
 TodoTareas("Negocios", Negocio),  
 TodoTareas("Personal", Personal),  
 TodoTareas("Otros", Otro)  
 */\*\* nombreTarea, categoria:TodoCategorias, estaSeleccionado no hace falta ponerlo \*/* )  
  
  
 */\*\* Crear el binding \*/* private lateinit var binding: ActivityTodoBinding  
  
 */\*\* Crear un acceso al clase TodoCategoriasAdapter.kt \*/* private lateinit var todoCategoriasAdapter: TodoCategoriasAdapter  
  
 */\*\* Crear un acceso al clase TodoTareasAdapter.kt \*/* private lateinit var todoTareasAdapter: TodoTareasAdapter  
  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *enableEdgeToEdge*()  
 */\*\* Inflar el binding con la vista \*/* binding = ActivityTodoBinding.inflate(*layoutInflater*)  
 setContentView(binding.*root*)  
 ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.*main\_todo*)) **{** v, insets **->** val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())  
 v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)  
 insets  
 **}** val dialog = Dialog(this)  
  
  
 */\*\* \*/* initUI()  
 */\*\* Función para iniciar todos los listeners \*/* iniListeners()  
 }  
  
 private fun initUI() {  
 */\*\* Inicializar el todoCategoriasAdapter \*/*//todoCategoriasAdapter =TodoCategoriasAdapter(categorias) {actualizaCategoria(it)}  
 todoCategoriasAdapter =  
 TodoCategoriasAdapter(categorias) **{** posicionIt **->** actualizaCategoria(posicionIt) **}** */\*\* Lo que le mandaremos al adapter \*/* binding.rvTodoCategoria.*layoutManager* =  
 LinearLayoutManager(this, LinearLayoutManager.*HORIZONTAL*, false)  
 */\*\* También le pasamos el adapte \*/* binding.rvTodoCategoria.*adapter* = todoCategoriasAdapter  
  
 */\*\* Inicializar el todoTareasAdapter \*/* todoTareasAdapter =  
 TodoTareasAdapter(tareas) **{** posicionIt **->** onItemSeleccionado(posicionIt) **}** */\*\* Lo que le mandaremos al adapter \*/* binding.rvTodoTareas.*layoutManager* = LinearLayoutManager(this)  
 */\*\* Lo que el adaptado le va a poner \*/* binding.rvTodoTareas.*adapter* = todoTareasAdapter  
  
 }  
  
 */\*\* Función para iniciar los listeners \*/* private fun iniListeners() {  
 */\*\* Para el botón flotante "fab"\*/* binding.fabAddTarea.setOnClickListener **{** mostrarDialogo() **}** }  
  
 */\*\* Método que se lanzará al pulsar el fab \*/* private fun mostrarDialogo() {  
 */\*\* Creamos una variable\*/* val dialogo = Dialog(this)  
 */\*\* Ponemos la vista al dialogo \*/* dialogo.setContentView(R.layout.*dialogo\_tarea*)  
  
 */\*\* hacemos el cast a los componentes, ya que con el binding queda desformado \*/* val btnAddTarea: Button = dialogo.findViewById(R.id.*btn\_add\_tarea*) //  
 val etTarea: EditText = dialogo.findViewById(R.id.*et\_tarea*) //  
 val rgCategorias: RadioGroup = dialogo.findViewById(R.id.*rg\_categorias*) //  
  
 */\*\* Captura el click del radioBroup \*/* btnAddTarea.setOnClickListener **{** //  
 /\* Variable para almacenar el texto el editText\*/  
 val textoTareaSelect = etTarea.*text*.toString()  
 if (textoTareaSelect.*isNotEmpty*()) {  
 */\*\* Guardar un una variable el id del RB (RadioButton) seleccionado \*/* val selectId = rgCategorias.*checkedRadioButtonId  
  
 /\*\* Acceder a un RB dentro del grupo y guardar el la variable el RB seleccionado\*/* val selectRadioButton: RadioButton = rgCategorias.findViewById(selectId)  
  
 */\*\* Comprobar que RB ha sido seleccionado a través del texto y guardar en la variable  
 \* la categoría \*/* val categoriaActual: TodoTareaCategorias = when (selectRadioButton.*text*) {  
 getString(R.string.*todo\_negocio*) -> Negocio  
 getString(R.string.*todo\_personal*) -> Personal  
 else -> Otro  
 }  
 */\*\* Añadir la tarea a la mutableListOf de tareas\*/* tareas.add(TodoTareas(textoTareaSelect, categoriaActual))  
 */\*\* Llamamos a la función que actualiza la lista del adapter\*/* updateTarea()  
 */\*\* Tras pulsar el botón de añadir tarea,cerramos el dialogo \*/* dialogo.hide()  
 } else {  
 Toast.makeText(this, "Ingresa la tarea primero", Toast.*LENGTH\_SHORT*).show()  
 }  
 **}** */\*\* Llamar al dialogo \*/* dialogo.show()  
 }  
  
 */\*\* Para modificar los items de las tareas, se crea una función que recibirá una posición (Int) \*/* private fun onItemSeleccionado(posicion: Int) {  
 */\*\* cambiar de seleccionado a no seleccionado a través del Boolean del TodoTareas\*/* tareas[posicion].estaSelect = !tareas[posicion].estaSelect  
 */\*\* Notifecar al adapter que hanhabido cambios \*/* updateTarea()  
 }  
  
 */\*\* Función para cambiar la selección del item \*/* private fun actualizaCategoria(posicion: Int) {  
 */\*\* pon lo opuesto de lo que había \*/* categorias[posicion].estaSeleccionado = !categorias[posicion].estaSeleccionado  
 */\*\* Actualizar el item seleccionado \*/* todoCategoriasAdapter.notifyItemChanged(posicion)  
 /\* Llamar al updateTarea\*/  
 updateTarea()  
 }  
  
 */\*\* Función para actualizar la lista que tiene el adapter\*/* private fun updateTarea() {  
 // Aquí pondremos el código para que pinte la selección según se habilite o no  
 todoTareasAdapter.notifyDataSetChanged()  
 }  
}

Creamos la variable:

/\* Variable para las categorías seleccionadas\*/  
val categoriasSelected : List<TodoTareaCategorias> = categorias.*filter* **{ it**.estaSeleccionado **}**

La variable va a ser un listado de “TodoTareaCategorias, que es la “sealed class” que va a ser igual a, de todas las categorías de la listOf de “categorías”, filtrame (filter) cada una de la lista (it) que esté seleccionada (El boloeano de la sealed class)

Creamos una nueva variable que almacenará las tareas delas categorías seleccionadas:

/\* Variable guardará las tareas que tengan las categorías seleccionadas \*/  
val nuevasTareas = tareas.*filter* **{** categoriasSelected.contains(**it**.categorias) **}**

Esta variable va a ser igual a: de todas las tareas de la mutablaListOf de “tareas”, filtrame y sólo quiero las tareas de las categorías seleccionadas “categoriasSelected” que contengan la tarea de la categoría seleccionada.

Ahora en el todoTareasAdapter le tenemos que pasar las tareas filtradas que va a pintar en vez de que pinte la lista completa:

/\* Le pasamos al adapte las tareas que tiene que pintar y no toda la lista \*/  
todoTareasAdapter.tareas

No nos coje “tareas” porque “tareas” de la clase “TodoTareasAdapte” está declarada con privada:

class TodoTareasAdapter(  
 private val tareas: List<TodoTareas>,  
 private val onTareaSeleccionada: (Int) -> Unit  
) :  
 RecyclerView.Adapter<TodoTareasViewHolder>() {

...

hay que quitarle el private y ya nos coje el atributo “tareas”. También saldrá el error de que en un “val” y no se puede modificar, por lo que se lo cambiamos a “var:

class TodoTareasAdapter(  
 var tareas: List<TodoTareas>,  
 private val onTareaSeleccionada: (Int) -> Unit  
) :  
 RecyclerView.Adapter<TodoTareasViewHolder>() {

Y con estos cambios ya podemos pasarle al “todoTareasAdapter” la nueva lista de tareas

*/\*\* Función para actualizar la lista que tiene el adapter\*/*private fun updateTarea() {  
 */\*\* Variable para guardar las categorías seleccionadas\*/* val categoriasSelected : List<TodoTareaCategorias> = categorias.*filter* **{ it**.estaSeleccionado **}** */\*\* Variable guardará las tareas que tengan las categorías seleccionadas \*/* val nuevasTareas = tareas.*filter* **{** categoriasSelected.contains(**it**.categorias) **}** /\* Le pasamos al adapte las tareas que tiene que pintar y no toda la lista \*/  
 todoTareasAdapter.tareas = nuevasTareas  
 todoTareasAdapter.notifyDataSetChanged()  
}

.

**NUEVA APP A PARTIR DE AQUÍ:**

El nuevo componente es el **SearchView,** este lleva un listener que no es le setOnClickListener, sino que es el **setOnQueryTextListener** y no va con corchetes “{}”, sino que va con paréntesis “()”que se le pasa un objeto del tipo **SearchView**  pero **ojo!!!** El que hay que importar es este:

import androidx.appcompat.widget.SearchView

Que es el mismo que seleccionemos en el xml

<androidx.appcompat.widget.SearchView

Y la clase principal de esta app quedaría así:

package com.nanduky.cursokotlindesde0.superheroapp  
  
import android.os.Bundle  
import androidx.activity.enableEdgeToEdge  
import androidx.appcompat.app.AppCompatActivity  
import androidx.appcompat.widget.SearchView  
import androidx.core.view.ViewCompat  
import androidx.core.view.WindowInsetsCompat  
import com.nanduky.cursokotlindesde0.R  
import com.nanduky.cursokotlindesde0.databinding.ActivitySuperHeroListBinding  
  
class SuperHeroListActivity : AppCompatActivity() {  
  
 */\*\* Crear el binding \*/* private lateinit var binding: ActivitySuperHeroListBinding  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *enableEdgeToEdge*()  
 */\*\* Inflar el binding \*/* binding = ActivitySuperHeroListBinding.inflate(*layoutInflater*)  
 setContentView(binding.*root*)  
 ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.*main\_super*)) **{** v, insets **->** val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())  
 v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)  
 insets  
 **}** */\*\* Función para iniciar los UI que aún no se que significa \*/* initUI()  
 }  
  
 private fun initUI() {  
 binding.svSuper.setOnQueryTextListener(object : SearchView.OnQueryTextListener{}) 🡨 Nuevo SearchView  
 }  
}

Esto da un error con el “**object”** y con la bombilla **implements members**

private fun initUI() {  
 binding.svSuper.setOnQueryTextListener(object : SearchView.OnQueryTextListener{  
 override fun onQueryTextSubmit(query: String?): Boolean {  
 *TODO*("Not yet implemented")  
 }  
  
 override fun onQueryTextChange(newText: String?): Boolean {  
 *TODO*("Not yet implemented")  
 }  
 })  
}

La primera es la que nos interesa, es la que se llamerá al pulsar el botón de buscar.

private fun initUI() {  
 binding.svSuper.setOnQueryTextListener(object : SearchView.OnQueryTextListener{  
 override fun onQueryTextSubmit(query: String?): Boolean {  
 //Esta es la función que se llamará al pulsar en el botón de buscar  
 }  
  
 override fun onQueryTextChange(newText: String?): Boolean {  
 // Esta función se llamará cada vez que se vaya escribiendo, pero no interesa  
 }  
 })  
}

La segunda es la que se llama mientras se escribe, pero ahora no nos interesa y quedará sin uso, para eso hay que pasarle un **return:**

private fun initUI() {  
 binding.svSuper.setOnQueryTextListener(object : SearchView.OnQueryTextListener{  
 */\*\* Función que se llamará al pulsar el botón de buscar \*/* override fun onQueryTextSubmit(query: String?): Boolean {  
 //  
 }  
  
 */\*\* Función que se llamará mientras se escribe, aquí quedará sin uso \*/* override fun onQueryTextChange(newText: String?): Boolean {  
 return false 🡨 se le pasa el return false  
 }  
 })  
}

y esto se puede optimizar así:

private fun initUI() {  
 binding.svSuper.setOnQueryTextListener(object : SearchView.OnQueryTextListener {  
 */\*\* Función que se llamará al pulsar el botón de buscar \*/* override fun onQueryTextSubmit(query: String?): Boolean {  
 *TODO*("Not yet implemented")  
 }  
  
  
 */\*\* Función que se llamará mientras se escribe, aquí quedará sin uso \*/* override fun onQueryTextChange(newText: String?) = false 🡨 optimezado  
   
 })  
}

Nos quedamos con la primera función y le pasamos un **return false**  que siempre tienen que llevar:

*/\*\* Función que se llamará al pulsar el botón de buscar \*/*override fun onQueryTextSubmit(query: String?): Boolean {  
  
 return false 🡨 Se le pone siempre

}

Creamos una función nueva “**buscarPorNombre(query)”** y se le pasa el query que es un String

override fun onQueryTextSubmit(query: String?): Boolean {  
 buscarPorNombre(query)  
 return false  
}

Si le damos al **Alt + Intro**  para crear la función dentro de la **class SuperHeroListActivity** nos crea una función con un parámetro nullable

private fun initUI() {  
 binding.svSuper.setOnQueryTextListener(object : SearchView.OnQueryTextListener {  
 */\*\* Función que se llamará al pulsar el botón de buscar \*/* override fun onQueryTextSubmit(query: String?): Boolean {  
 buscarPorNombre(query)  
 return false  
 }  
  
  
 */\*\* Función que se llamará mientras se escribe, aquí quedará sin uso \*/* override fun onQueryTextChange(newText: String?) = false  
  
 })  
}  
  
private fun buscarPorNombre(query: String?) { 🡨 nueva función   
}

No nos interesa que sea una String nullable, le quitamos la nullabilidad “?”

private fun buscarPorNombre(query: String) { 🡨 “?” borrado  
}

y el la función **onQueryTextSubmit** le decimos que nos pase el query o vacio:

*/\*\* Función que se llamará al pulsar el botón de buscar \*/*override fun onQueryTextSubmit(query: String?): Boolean {  
 /\* Nos pasa un query o una String vacía, pero no null \*/  
 buscarPorNombre(query.*orEmpty*())  
 return false  
}

Tenemos que trabajar con **Retrofit**, y para que funcione **Retrofit,** primero hay que crearlo. Así que creamos la función **getRetrofitSHLA** que devolverá una instancia de **Retrofit:**

/\* Creación de Retrofit \*/  
private fun getRetrofitSHLA(): Retrofit{  
  
}

**Retrofit** dará un error porque hay que importarla:

import retrofit2.Retrofit 🡨 Nuevo import

La API con la que estoy trabajando aquí es de:

<https://superheroapi.com/index.html>

Access-token: 3609252f23e4fc59e44367838366f23b

Para crear un objeto **Retrofit:**

*/\*\* Creación de Retrofit \*/*private fun getRetrofitSHLA(): Retrofit{  
 return Retrofit  
 .Builder()  
 .baseUrl("https://superheroapi.com/api/") 🡨 Siempre con eñ "/" final   
 .addConverterFactory(GsonConverterFactory.create()) 🡨 para convertir   
 .build() 🡨 construye te  
}

**.baseUrl()** SIEMPRE tiene que llevar el “/” final, sino da error

**.addConverterFactory(GsonConverterFactory.create())** es el convertidor y hay que importarlo:

import retrofit2.converter.gson.GsonConverterFactory 🡨 Nuevo import

Lo ideal es no llamar a getRetrofit en cada llamda, Para hacer el apaño por ahora, ya que no estamos usando la “arquitectura” todavía, creamos una variable **lateinit:**

private lateinit var retrofitSHLA: Retrofit

...Crearemos un objeto en **Retrofit,** peri aún no...

Se creará cuando se creé la pantalla en el **onCreate**:

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

*...*/\* Iniciar el retrofit\*/  
 retrofitSHLA = getRetrofitSHLA()  
 */\*\* Función para iniciar los UI que aún no se que significa \*/* initUI()  
}

Ahora hay que crear una cosa nueva que es una **interface,**  que se crea igual que si fuese una **class**

interface ApiServiceSuperInterface {  
}

Empezamos identificando el tipo de llamada que vamos a hacer, que en este caso, la API sólo tiene llamdas **@GET** y importamos el get:

import retrofit2.http.GET 🡨 Nuevo import  
  
interface ApiServiceSuperInterface {  
 @GET 🡨 Tipo de llamada que vamos a hacer  
}

Al **@GET**  hay que pasarle el final de la url:

@GET("") // final de la ruta url que nos faltaba en baseUrl de getRetrofitSHLA()

Y quedaría asi:

@GET("/3609252f23e4fc59e44367838366f23b/search/{nombre}")

Ahora hay que hacer una función, pero como vamos a trabajar con courutinas, (que las courutinas sirven para cuando se hacen llamadas a bases de datos, internet o sitios que pueden tardar en responder, no se trabe el móvil esperando la respuesta),ya no se le pone **private,**  sino que se le pone **suspend fun:**

import retrofit2.http.GET  
import retrofit2.http.Path 🡨 Nuevo import  
  
interface ApiServiceSuperInterface {  
 */\*\* Identificamos el tipo de llamada que vamos a hacer. Esta API sólo tiene GET \*/* @GET("/3609252f23e4fc59e44367838366f23b/search/{nombre}")  
 */\*\* Aquí no es private, sino que son suspend \*/* suspend fun getSuperHeroInterface(@Path("nombre")) 🡨 Ruta url + {nombre}  
}

A **“gerSuperHeroInterface”** se le dice que a la ruta de la **baseUrl** (el principio está en: SuperHeroListActivity.kt/ fun getRetroFitSHLA) del **@GET** me busque en toda lo que se le pase con la variable “**nombre”** que se la pasaremos a través de una String:

suspend fun getSuperHeroInterface(@Path("nombre") superHeroNombreInterface : String)

Esta función recibirá por parámetros un valor que será el de la función:

private fun buscarPorNombreSHLA(query: String) {  
 *TODO*("Not yet implemented")  
}

de la activity principal **(SuperHeroListActivity)** que es la que le pasará el valor String que será el **nombre.** Esta función nos va a devolver un **response<> (**de retrofit):

import retrofit2.Response 🡨 Nuevo import  
import retrofit2.http.GET  
import retrofit2.http.Path  
  
interface ApiServiceSuperInterface {  
 */\*\* Identificamos el tipo de llamada que vamos a hacer. Esta API sólo tiene GET \*/* @GET("/3609252f23e4fc59e44367838366f23b/search/{nombre}")  
 */\*\* Aquí no son private, sino que son suspend \*/* suspend fun getSuperHeroInterface(@Path("nombre") superHeroNombreInterface : String):Response<> 🡨 Nos devuelve una respuesta  
}

El **Response<>**  nos devuelve un “JSON” convertido, Para eso hay que crear las data class:

package com.nanduky.cursokotlindesde0.superheroapp  
  
data class SuperHeroDataClassResponse ()

**\*\*\* ACLARACIÓN \*\*\***

Los datos que estamos usando son del la página de la API:

<https://superheroapi.com/index.html>

Te logeas con tu GitHub y te da un Access-token, en mi caso es :

3609252f23e4fc59e44367838366f23b

Para ver una lista (Json) de super hay que ingresar en:

<https://www.superheroapi.com/api/3609252f23e4fc59e44367838366f23b/search/super>

copias esa lista y en la página:

https://jsoneditoronline.org/

Pegas tu Json con Ctrl + V, le das a **tree**  y te sale un árbol con 2 valores y un array:

"response": "success", 🡨 String

"results-for": "super", 🡨 String, pero no nos interesa

"results": [ ] 🡨 Array

Con esto es con lo que hay que trabajar en la data class SuperHeroDataClassResponse (

**\*\*\* FIN ACLARACIÓN \*\*\***

Se le pasan los parámetros necesarios para el Json:

package com.nanduky.cursokotlindesde0.superheroapp  
  
data class SuperHeroDataClassResponse (val response:String)

El nombre de la variable se puede cambiar usando @SerializedName:

package com.nanduky.cursokotlindesde0.superheroapp  
  
import com.google.gson.annotations.SerializedName 🡨 nuevo import de gson  
  
data class SuperHeroDataClassResponse (

@SerializedName("response") val respuestaSHDTR :String

)

Si le mandáramos esto ahora a nuestro **getSuperHeroInterface** nos devolvería un **Json**  gigante:

interface ApiServiceSuperInterface {  
 */\*\* Identificamos el tipo de llamada que vamos a hacer. Esta API sólo tiene GET \*/* @GET("/3609252f23e4fc59e44367838366f23b/search/{nombre}")  
 */\*\* Aquí no son private, sino que son suspend \*/* suspend fun getSuperHeroInterface(@Path("nombre") superHeroNombreInterface: String):  
 Response<SuperHeroDataClassResponse> 🡨 Aún sin filtrar  
}

**COROUTINE**

Para trabajar con coroutines, hay que indicarle si será en el hilo principla (**MAIN)** o en un hilo secundarñiao (**IO):**

private fun buscarPorNombreSHLA(query: String) {  
 /\* Corrutina en un hilo secundarío\*/  
 *CoroutineScope*(Dispatchers.IO).*launch* **{**  *Todo que que se haga aquí es en el hilo secundarío* **}**}

Aquí abajo se le dice que con retrofitSHLA nos cree la instancia que hemos creado antes, **(ApiServiceSuperInterface )**

private fun buscarPorNombreSHLA(query: String) {  
 */\*\* Corrutina en un hilo secundarío\*/  
 CoroutineScope*(Dispatchers.IO).*launch* **{** */\*\* lo que se haga aquí es en el hilo secundarío \*/* val miRespuesta = retrofitSHLA.create(ApiServiceSuperInterface ::class.*java*)  
 **}**}

y se le pasa la interface que creemos también antes y nos pide una **String** que será el **query** que es para el String **“nombre”** de la interface:

**PERMISOS:**

En el manifest hay que concederle permisos:

<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools">  
  
 <uses-permission android:name="android.permission.INTERNET"/> 🡨 permiso   
  
 <application

...

Para probar si funciona, hacemos un if con **isSuccessful,**  que esto nos dirá si funciona o no, pasando le un Log.i Para comprobar si funcionó:

private fun buscarPorNombreSHLA(query: String) {  
 */\*\* Corrutina en un hilo secundarío\*/  
 CoroutineScope*(Dispatchers.IO).*launch* **{** */\*\* lo que se haga aquí es en el hilo secundarío \*/* val miRespuesta: Response<SuperHeroDataClassResponse> = retrofitSHLA.create(ApiServiceSuperInterface ::class.*java*).getSuperHeroInterface(query)  
 if(miRespuesta.*isSuccessful*){ 🡨 Comprobar si funciona  
 Log.i("Fer","Funciona :)")  
 } else {  
 Log.i("Fer"," no Funciona :(")}  
 **}**}

Y en Logcat:

2024-10-03 16:44:00.042 13260-13310 Fer com.nanduky.cursokotlindesde0 I Funciona :)

Ahora que sabemos que funciona, para acceder al array con los resultados, tenemos que implementar la data class con los parámetros que faltan. Para acceder al array en vez de una variable **String**, tiene que ser una **List<>:**

import com.google.gson.annotations.SerializedName   
  
data class SuperHeroDataClassResponse(  
 @SerializedName("response") val respuestaSHDCR: String,  
 @SerializedName("results") val resultadoSuperHeroeSHDCR: List<> /🡨 para el array  
)

Cómo dentro del **List<>**  no se pueden meter muchos atributos, se creará otra **data class** que va a ser como un objeto hijo del **data class SuperHeroDataClassResponse() :**

/\* data class hijo del data class SuperHeroDataClassResponse \*/  
data class SuperHeroItemDataClasResponse(  
)

Y esta clase hijo se la pasamos como parámetro a la vriable de resultados:

package com.nanduky.cursokotlindesde0.superheroapp  
  
import com.google.gson.annotations.SerializedName  
  
data class SuperHeroDataClassResponse(  
 @SerializedName("response") val respuestaSHDCR: String,  
 @SerializedName("results") val resultadoSuperHeroeSHDCR: List<SuperHeroItemDataClasResponse> 🡨 nuevo parámetro  
)  
  
data class SuperHeroItemDataClasResponse(  
)

Y añadimos los valores que queremos recuperar:

package com.nanduky.cursokotlindesde0.superheroapp  
  
import com.google.gson.annotations.SerializedName  
  
data class SuperHeroDataClassResponse(  
 @SerializedName("response") val respuestaSHDCR: String,  
 @SerializedName("results") val resultadoSuperHeroeSHDCR: List<SuperHeroItemDataClasResponse>  
)  
  
*/\*\* data class hijo del data class SuperHeroDataClassResponse \*/*data class SuperHeroItemDataClasResponse(  
 @SerializedName("id") val superHeroId: String, 🡨 nuevos valores  
 @SerializedName("name") val superHeroNombre: String  
)

en el **activity\_super\_hero\_list.xml** añadiremos un nuevo componente para el progreso de la carga la pantalla:

<ProgressBar  
 android:id="@+id/pb\_super"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:visibility="gone"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 />

Esto hace que cuando le demos al botón de buscar de la app se vea lo que sería el reloj de arena de windos, pero moderno, como el atributo **invisible**  está en **gone,** al darle a buscar, hay que mostrar el progressBar para indicar al usuario que está cargando:

private fun buscarPorNombreSHLA(query: String) {  
 /\* Mostrar el pregressBar \*/  
 bindingSHLA.pbSuper.*isVisible* = true 🡨 Muestra el pregressBar  
 */\*\* Corrutina en un hilo secundarío\*/  
 CoroutineScope*(Dispatchers.IO).*launch* **{** */\*\* lo que se haga aquí es en el hilo secundarío \*/* val miRespuesta: Response<SuperHeroDataClassResponse> = retrofitSHLA.create(ApiServiceSuperInterface ::class.*java*).getSuperHeroInterface(query)  
 if(miRespuesta.*isSuccessful*){ 🡨 si miRepuesta no es nula  
 val respuestaBPN: SuperHeroDataClassResponse? = miRespuesta.body()  
 if(respuestaBPN != null){ }  
 } else { **}**

**}**}

Cuando termina de cargar la app, hay que esconder otra vez el **progressBar**  pero hay que recordar que estamos en una **coroutina**, en un hilo secundarío, y no podemos esconder el **progressBar** desde el hilo secundario, y para eso hay que meterlo entre la llaves de la función reservada para estos casos **runOnUiThread {},**  y todo lo que esté dentro del las llaves, correrá en el hilo principas:

private fun buscarPorNombreSHLA(query: String) {  
 */\*\* Mostrar el pregressBar \*/* bindingSHLA.pbSuper.*isVisible* = true  
 */\*\* Corrutina en un hilo secundarío\*/  
 CoroutineScope*(Dispatchers.IO).*launch* **{** */\*\* lo que se haga aquí es en el hilo secundarío \*/* val miRespuesta: Response<SuperHeroDataClassResponse> =  
 retrofitSHLA.create(ApiServiceSuperInterface ::class.*java*).getSuperHeroInterface(query)  
 if (miRespuesta.*isSuccessful*) {  
 val respuestaBPN: SuperHeroDataClassResponse? = miRespuesta.body()  
 if (respuestaBPN != null) {  
 /\* Corre en el hilo principal lo que meta entre las llaves \*/  
 runOnUiThread **{** bindingSHLA.pbSuper.*isVisible* = false **}**  
 }  
 } else { }  
 **}**}

Hay que hacer el **RecyclerView,** creamos primero el adspter:

package com.nanduky.cursokotlindesde0.superheroapp  
  
class SuperHiroAdapter ()

Y le pasamos una lista due es la de **SuperHeroItemDataClassResponse** vacia con el **emptuList().**

package com.nanduky.cursokotlindesde0.superheroapp  
  
class SuperHiroAdapter (var superHeroAdapter: List<SuperHeroItemDataClassResponse> = *emptyList*() )

Que va ser de tipo **RecyclerView.Adapter**  y hay que pasarle el viewHolder que aún no está creado, pues lo creamos:

package com.nanduky.cursokotlindesde0.superheroapp  
  
import android.view.View  
import androidx.recyclerview.widget.RecyclerView  
  
class SuperHeroViewHolder (view: View) : RecyclerView.ViewHolder(view){  
  
}

Creamos una función que recibirá el item de la data clas:

package com.nanduky.cursokotlindesde0.superheroapp  
  
import android.view.View  
import androidx.recyclerview.widget.RecyclerView  
  
class SuperHeroViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
 /\* Función que va a recibir uno de los items de SuperHeroItemDataClassResponse\*/  
 fun bindSHVH(superHeroItemDataClassResponse: SuperHeroItemDataClassResponse) {}  
}

Y con esto ya podemos pasárselo al adapter, implementar los 3 métodos de siempre y pasarle el tamaño al **getItemCount:**

package com.nanduky.cursokotlindesde0.superheroapp  
  
import android.view.ViewGroup  
import androidx.recyclerview.widget.RecyclerView  
  
class SuperHiroAdapter(var superHeroListAdapter: List<SuperHeroItemDataClassResponse> = *emptyList*()) :  
 RecyclerView.Adapter<SuperHeroViewHolder>() {  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): SuperHeroViewHolder {  
 *TODO*("Not yet implemented")  
 }  
  
 override fun onBindViewHolder(holderSHA: SuperHeroViewHolder, position: Int) {  
 *TODO*("Not yet implemented")  
 }  
  
 override fun getItemCount() = superHeroListAdapter.size  
  
}

en el **onFindViewHolder** creamos un item llamando a **superHerListAdapter** y accedemos a la posición que nos de el **onBindViewHolder,** se lo pasamosa la función bindSHVH del **SuperHeroViewHolder:**

package com.nanduky.cursokotlindesde0.superheroapp  
  
import android.view.ViewGroup  
import androidx.recyclerview.widget.RecyclerView  
  
class SuperHiroAdapter(var superHeroListAdapter: List<SuperHeroItemDataClassResponse> = *emptyList*()) :  
 RecyclerView.Adapter<SuperHeroViewHolder>() {  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): SuperHeroViewHolder {  
 *TODO*("Not yet implemented")  
 }  
  
 override fun onBindViewHolder(holderSHA: SuperHeroViewHolder, position: Int) {  
 val itemSHA = superHeroListAdapter[position] 🡨 creamos item  
 holderSHA.bindSHVH(itemSHA) 🡨 pasamos item  
 }  
 override fun getItemCount() = superHeroListAdapter.size  
}

Esto se puede optimizar quitando la variable y pasandolo directamente:

override fun onBindViewHolder(holderSHA: SuperHeroViewHolder, position: Int) {  
/\* o lo podemos optimizar quitando la variableholderSHA.bindSHVH(superHeroListAdapter[position])  
}

El que queda, el **onCreateViewHoder** es el que especifica el **layaut** que tenemos que usa, y como aún no está creado, lo creamos:

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="150dp"  
 android:orientation="horizontal">  
 <TextView  
 android:id="@+id/tv\_super\_nombre"  
 android:layout\_width="wrap\_content"  
 android:textSize="20sp"  
 android:layout\_height="wrap\_content">  
 </TextView>  
  
</LinearLayout>

Y ya podemos pasarle al **onCreateViewHolder** del adapdter el layout inflado:

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): SuperHeroViewHolder {  
 /\* Variable para inflar la vista y pasarlo alViewHolder\*/  
 return SuperHeroViewHolder(  
 LayoutInflater.from(parent.*context*).inflate(R.layout.*item\_superhero*, parent, false)  
 )  
}

Y quedaría así:

package com.nanduky.cursokotlindesde0.superheroapp  
  
import android.view.LayoutInflater  
import android.view.ViewGroup  
import androidx.recyclerview.widget.RecyclerView  
import com.nanduky.cursokotlindesde0.R  
  
class SuperHiroAdapter(var superHeroListAdapter: List<SuperHeroItemDataClassResponse> = *emptyList*()) :  
 RecyclerView.Adapter<SuperHeroViewHolder>() {  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): SuperHeroViewHolder {  
 /\* Variable para inflar la vista y pasarlo alViewHolder\*/  
 return SuperHeroViewHolder(  
 LayoutInflater.from(parent.*context*).inflate(R.layout.*item\_superhero*, parent, false)  
 )  
 }  
  
 override fun onBindViewHolder(holderSHA: SuperHeroViewHolder, position: Int) {  
 */\*\* Creamos un Item llamado al superheroelist y la posicón que le toque \*/* //val itemSHA = superHeroListAdapter[position]  
 */\*\* lo mandamos a la funcion bindSHVH del ViewHolder \*/* //holderSHA.bindSHVH(itemSHA)  
 */\*\* O lo optimizamos quitando la variable \*/* holderSHA.bindSHVH(superHeroListAdapter[position])  
 }  
  
 */\*\* Le pasamos el tamaño de la lista \*/* override fun getItemCount() = superHeroListAdapter.size  
}

En la activity principal creamos el adapter:

class SuperHeroListActivity : AppCompatActivity() {  
  
 */\*\* Crear el binding \*/* private lateinit var bindingSHLA: ActivitySuperHeroListBinding  
 private lateinit var retrofitSHLA: Retrofit  
  
 /\* Crear el adapter \*/  
 private lateinit var adapterSHLA : SuperHeroAdapter

En el uniUI:

private fun initUI() {  
 bindingSHLA.svSuper.setOnQueryTextListener(object : SearchView.OnQueryTextListener {  
 */\*\* Función que se llamará al pulsar el botón de buscar \*/* override fun onQueryTextSubmit(query: String?): Boolean {  
 */\*\* Nos pasa un query o una String vacía, pero no null \*/* buscarPorNombreSHLA(query.*orEmpty*())  
 return false  
 }  
  
 */\*\* Función que se llamará mientras se escribe, aquí quedará sin uso \*/* override fun onQueryTextChange(newText: String?) = false  
 })  
   
 /\* crear el adapter, no hay que pasar le parámetro porque se inicializó vacía con emptyList() \*/  
 adapterSHLA = SuperHeroAdapter()  
 /\* variable para no repetir el binding \*/  
 val rvSuper = bindingSHLA.rvSuper  
 /\* Evitar que el recyclerView cambie de tamaño \*/  
 rvSuper.setHasFixedSize(true)  
 /\* Hay que pasarle un manager \*/  
 rvSuper.*layoutManager* = LinearLayoutManager(this)  
 /\* Pasarle el adapte que emos creado \*/  
 rvSuper.*adapter* = adapterSHLA

}

Vamos al **SuperHeroViewHolder** y creamos el **binding,** que aquí no tiene que ser “lateinit”

package com.nanduky.cursokotlindesde0.superheroapp  
  
import android.view.View  
import androidx.recyclerview.widget.RecyclerView  
import com.nanduky.cursokotlindesde0.databinding.ItemSuperheroBinding  
  
class SuperHeroViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
  
 /\* Crear el binding para el recyclerView \*/  
 private val bindigSHVH = ItemSuperheroBinding.bind(view)  
  
 */\*\* Función que va a recibir uno de los items de SuperHeroItemDataClassResponse\*/* fun bindSHVH(superHeroItemDataClassResponse: SuperHeroItemDataClassResponse) {}  
}

Y con esto ya podemos acceder a las vistas:

*/\*\* Función que va a recibir uno de los items de SuperHeroItemDataClassResponse\*/*fun bindSHVH(superHeroItemDataClassResponse: SuperHeroItemDataClassResponse) {  
 /\* pasarle el nombre \*/  
 bindigSHVH.tvSuperNombre.*text* = superHeroItemDataClassResponse.superHeroNombre  
}

Pero esto ahora no hace nada, porque hemos creado el reclyclerView, pero la lista está vacía. Tenemos que rellenar esta lista cada vez que actualicemos el listado, que el API nos respinda. Creamos método nuevo que va a recibir otro listado de super héroes nuevo:

class SuperHeroAdapter(var superHeroListAdapter: List<SuperHeroItemDataClassResponse> = *emptyList*()) :  
 RecyclerView.Adapter<SuperHeroViewHolder>() {  
  
 fun actualizaListaSHA(

superHeroListAdapter:List<SuperHeroItemDataClassResponse>){  
 /\* el "this" es para referirse al valor de la clase y sil el al de la función \*/  
 this.superHeroListAdapter = superHeroListAdapter  
 }  
 ...

El **this** es para referirse al superHeroListAdapter de la clase, mientras que si no se pone el **this,** se refiere al de la función

Una ves pasados los datos a la nueva lista, hay que avisar al adapter:

fun actualizaListaSHA(superHeroListAdapter: List<SuperHeroItemDataClassResponse>){  
 */\*\* el "this" es para referirse al valor de la clase y sil el al de la función \*/* this.superHeroListAdapter = superHeroListAdapter  
 /\* Actualiza el adapte\*/  
 notifyDataSetChanged()  
}

ahora este método (o función) hay que llamarla cuando hayan cargado los valores de la búsqueda de la app, si la respuesta es distinta de **null**, dentro del hilo principal **runOnUiThread:**

private fun buscarPorNombreSHLA(query: String) {  
 */\*\* Mostrar el pregressBar \*/* bindingSHLA.pbSuper.*isVisible* = true  
 */\*\* Corrutina en un hilo secundarío\*/  
 CoroutineScope*(Dispatchers.IO).*launch* **{** */\*\* lo que se haga aquí es en el hilo secundarío \*/* val miRespuesta: Response<SuperHeroDataClassResponse> =  
 retrofitSHLA.create(ApiServiceSuperInterface ::class.*java*).getSuperHeroInterface(query)  
 if (miRespuesta.*isSuccessful*) {  
 val respuestaBPN: SuperHeroDataClassResponse? = miRespuesta.body()  
 Log.i("Fer", "Funciona :)")  
 if (respuestaBPN != null) {  
 */\*\* Corre en el hilo principal lo que meta entre las llaves \*/* runOnUiThread **{** bindingSHLA.pbSuper.*isVisible* = false  
 /\*Se llama al método o función con la lista de heroes de la busqueda \*/ adapterSHLA.actualizaListaSHA(respuestaBPN.resultadoSuperHeroeSHDCR)  
 **}** Log.i("Fer", respuestaBPN.toString())  
 }  
 } else {  
 Log.i("Fer", " no Funciona :(")  
 }  
 **}**}

Se tiene que crear otra **data class** para recuperar la url, que está dentro de image:

{

"response": "success", 🡨 @SerializedName("response") val respuestaSHDCR: String

"results-for": "batman",

"results": [🡨@SerializedName("results") val resultadoSuperHeroeSHDCR: List<SuperHeroItemDataClassResponse>

{

"id": "69",

"name": "Batman",

"powerstats": { },

"biography": { },

"appearance": { },

"work": { },

"connections": { },

"image": { 🡨 Carpeta image que contiene la url

"url": "https://www.superherodb.com/pictures2/portraits/10/100/10441.jpg"

}

},

{

...

En este ejemplo, “**id”** y **“name”** están en el primer nivel del array, mientras que **“url”** está en el segundo nivel, Primero hay que acceder a **image** y luego a **url**.

Para ello se crea una **data class**  que contendrá la **url:**

/\* Nueva data class para recuperar la url de la imagen \*/  
data class SuperHeroImageDataClassResponse(  
 @SerializedName("url") val urlImage : String  
)

Y esta clase será llamada por la **data class**  que contiene los ítems, en concreto la image:

package com.nanduky.cursokotlindesde0.superheroapp  
  
import com.google.gson.annotations.SerializedName  
  
data class SuperHeroDataClassResponse(  
 @SerializedName("response") val respuestaSHDCR: String,  
 @SerializedName("results") val resultadoSuperHeroeSHDCR: List<SuperHeroItemDataClassResponse>  
)  
  
*/\*\* data class hijo del data class SuperHeroDataClassResponse \*/*data class SuperHeroItemDataClassResponse(  
 @SerializedName("id") val superHeroId: String,  
 @SerializedName("name") val superHeroNombre: String,  
 /\* item para pasar la url de la imagen al adapter \*/  
 @SerializedName("image") val superHeroImage: SuperHeroImageDataClassResponse  
)  
  
*/\*\* Nueva data class para recuperar la url de la imagen \*/*data class SuperHeroImageDataClassResponse(  
 @SerializedName("url") val urlImage : String  
)

Este **superHiroImage** nos va a devolver una (url), al **ImageView**no podemos pasarle una (url), modificaremos la vista “**item\_suoergero.xml”,** básicamente, al Framelayout lo pasamos a hijo dentro del un padre CardView, al textView que tenemos le añadimos un color con transparencia, un textColor y lo metemos dentro de otro frameLayout que le ponemos una medida y un color y un gravity center. Encima creamos un ImageView con el atributo nuevo **android:scaleType="centerCrop":**

<?xml version="1.0" encoding="utf-8"?>  
  
<androidx.cardview.widget.CardView 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:layout\_width="match\_parent"  
 android:layout\_height="300dp"  
 app:cardCornerRadius="22dp"  
 android:layout\_marginVertical="8dp"  
 android:layout\_marginHorizontal="16dp"  
 app:cardElevation="8dp">  
  
 <FrameLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
  
 <ImageView  
 android:id="@+id/iv\_super\_image"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:scaleType="centerCrop"/>  
  
 <FrameLayout  
 android:id="@+id/fl\_super\_nombre"  
 android:layout\_width="match\_parent"  
 android:background="#8B673AB7"  
 android:layout\_gravity="bottom"  
 android:layout\_height="30dp">  
 <TextView  
 android:id="@+id/tv\_super\_nombre"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="center"  
 tools:text="SuperHero"  
 android:textColor="@color/white"  
 android:textSize="20sp">  
  
 </TextView>  
 </FrameLayout>  
 </FrameLayout>  
</androidx.cardview.widget.CardView>

Y ahora ya le podemos pasar la imagen al ViewHolder a la vista a través de la librería **Picasso**:

class SuperHeroViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
  
 */\*\* Crear el binding para el recyclerView \*/* private val bindigSHVH = ItemSuperheroBinding.bind(view)  
  
 */\*\* Función que va a recibir uno de los items de SuperHeroItemDataClassResponse\*/* fun bindSHVH(superHeroItemDataClassResponse: SuperHeroItemDataClassResponse, ) {  
 */\*\* pasarle el nombre \*/* bindigSHVH.tvSuperNombre.*text* = superHeroItemDataClassResponse.superHeroNombre  
 /\* Pasar la url de "image" a través de Picasso \*/ Picasso.get().load(superHeroItemDataClassResponse.superHeroImage.urlImage).into(bindigSHVH.ivSuperImage)  
 }  
}

Con esto, la app ya funciona y muestra una lista de Super Heroes al introducir un nombre en el buscador, ahora hay que hacer que al pulsar en un super héroe, te lleve a una nueva pantalla con todos los datos de ese super héroe.

Creamos una nueva activity, creamos el **binding**  y lo inflamos:

package com.nanduky.cursokotlindesde0.superheroapp  
  
import android.os.Bundle   
import androidx.appcompat.app.AppCompatActivity   
import com.nanduky.cursokotlindesde0.databinding.ActivityDetallesSuperHeroBinding  
  
class DetallesSuperHeroActivity : AppCompatActivity() {  
  
 /\* Crear el binding \*/  
 private lateinit var bindingDSHA: ActivityDetallesSuperHeroBinding  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *enableEdgeToEdge*()  
 /\* Inflar el binding \*/  
 bindingDSHA = ActivityDetallesSuperHeroBinding.inflate(*layoutInflater*)  
 setContentView(bindingDSHA.*root*) }  
}

Creamos en el **SuperHeorListActivity**, la activity principal, un método/función para navegar a la nueva activity con un parámetro **id**:

/\* Función para ir a la DetalleSuperHeroActivity con un parámetro\*/  
private fun irDetalleSuperHeroActivity(id: String) {  
 val detallesSuperHeroActivityIntent = Intent(this,DetallesSuperHeroActivity::class.*java*)  
 detallesSuperHeroActivityIntent.putExtra("",id) 🡨 falta el identificador  
 startActivity(detallesSuperHeroActivityIntent)  
}

Para no equivocarnos al recuperar el identificador del **putExtra,** lo meteremos en un **companion object,** que es una forma de acceder a lo que hay ahí dentro desde cualquier lugar. Se creará una varable constante, y esto se hará en **DetallesSuperHeroActivity**:

/\* Companion para acceder a su contenido desde cualquier lugar \*/  
companion object{  
 /\* Constante para el putExtra\*/  
 const val EXTRA\_ID = "extra\_id"  
}

En la función **irDetalleSuperHeroActivity** ya podemos pasar el identificado si miedo a equivocarnos, habrá que importarlo.

*/\*\* Función para ir a la DetalleSuperHeroActivity con un parámetro\*/*private fun irDetalleSuperHeroActivity(id: String) {  
 val detallesSuperHeroActivityIntent = Intent(this,DetallesSuperHeroActivity::class.*java*)  
 /\* Hay que importa el EXTRA\_ID \*/  
 detallesSuperHeroActivityIntent.putExtra(*EXTRA\_ID*,id)  
 startActivity(detallesSuperHeroActivityIntent)  
}

Para llamar esta función cada vez que se pulse en la imagen de un super, hay que hacerlo a través de una función **lambda.**

En el **SuperHeroAdapter,** además de recibir la **List<SuperHero…>** a recibir la función **lambda:**

class SuperHeroAdapter(var superHeroListAdapter: List<SuperHeroItemDataClassResponse> = *emptyList*(), private val onItemSeleccionado:()) :  
 RecyclerView.Adapter<SuperHeroViewHolder>() {

...

**onItemSeleccionado()** necesita una **id,** y las id’s están dentro del viewHoder

class SuperHeroAdapter(  
 var superHeroListAdapter: List<SuperHeroItemDataClassResponse> = *emptyList*(),  
 private val onItemSeleccionado: (String) -> Unit 🡨 Función Lambda  
) :  
 RecyclerView.Adapter<SuperHeroViewHolder>() {

La función **lambda** se la pasamos al **onBindViewHolder** para que se la envie al **bindSHVH.**

Se la mandamos sin los paréntesis, porque sólo queremos pasarlo, con paréntesis lo ejecutaría:

override fun onBindViewHolder(holderSHA: SuperHeroViewHolder, position: Int) {  
 */\*\* Creamos un Item llamado al superheroelist y la posicón que le toque \*/* //val itemSHA = superHeroListAdapter[position]  
 */\*\* lo mandamos a la funcion bindSHVH del ViewHolder \*/* //holderSHA.bindSHVH(itemSHA)  
 */\*\* O lo optimizamos quitando la variable \*/* holderSHA.bindSHVH(superHeroListAdapter[position],onItemSeleccionado) 🡨 Enviamos lambda  
}

Ahora el **bindSHVH**  del **SuperHEroViewHOlder** nos pide un segundo parámetro, pues le pasamos el parámetro de la función lambda:

package com.nanduky.cursokotlindesde0.superheroapp  
  
import android.view.View  
import androidx.recyclerview.widget.RecyclerView  
import com.nanduky.cursokotlindesde0.databinding.ItemSuperheroBinding  
import com.squareup.picasso.Picasso  
  
class SuperHeroViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
  
 */\*\* Crear el binding para el recyclerView \*/* private val bindigSHVH = ItemSuperheroBinding.bind(view)  
  
 */\*\* Función que va a recibir uno de los items de SuperHeroItemDataClassResponse\*/* fun bindSHVH(  
 superHeroItemDataClassResponse: SuperHeroItemDataClassResponse,  
 onItemSeleccionado: (String) -> Unit 🡨 Parámetro de la función lambda  
 ) {  
 */\*\* pasarle el nombre \*/* bindigSHVH.tvSuperNombre.*text* = superHeroItemDataClassResponse.superHeroNombre  
 /\* Pasar la url de "image" a través de Picasso \*/  
 Picasso.get().load(superHeroItemDataClassResponse.superHeroImage.urlImage)  
 .into(bindigSHVH.ivSuperImage)  
 }  
}

Capturamos el click que se haga en cualquier parte de la vista

package com.nanduky.cursokotlindesde0.superheroapp  
  
import android.view.View  
import androidx.recyclerview.widget.RecyclerView  
import com.nanduky.cursokotlindesde0.databinding.ItemSuperheroBinding  
import com.squareup.picasso.Picasso  
  
class SuperHeroViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
  
 */\*\* Crear el binding para el recyclerView \*/* private val bindigSHVH = ItemSuperheroBinding.bind(view)  
  
 */\*\* Función que va a recibir uno de los items de SuperHeroItemDataClassResponse\*/* fun bindSHVH(  
 superHeroItemDataClassResponse: SuperHeroItemDataClassResponse,  
 onItemSeleccionado: (String) -> Unit   
 ) {  
 */\*\* pasarle el nombre \*/* bindigSHVH.tvSuperNombre.*text* = superHeroItemDataClassResponse.superHeroNombre  
 */\*\* Pasar la url de "image" a través de Picasso \*/* Picasso.get().load(superHeroItemDataClassResponse.superHeroImage.urlImage)  
 .into(bindigSHVH.ivSuperImage)  
 /\* Capturar el click en cualquier parte de la vista \*/  
 bindigSHVH.*root*.setOnClickListener**{}** }  
}

Le pasamos la fusión lambda con la id del super héroe seleecionado:

*/\*\* Capturar el click en cualquier parte de la vista \*/*bindigSHVH.*root*.setOnClickListener**{** onItemSeleccionado(superHeroItemDataClassResponse.superHeroId)

Ahora en la activity principal , el **adapterSHLA = SuperHeroAdapte()** marca un error porque también necesita la función lambda como parámetro:

private fun initUI() {  
 bindingSHLA.svSuper.setOnQueryTextListener(object : SearchView.OnQueryTextListener {  
 */\*\* Función que se llamará al pulsar el botón de buscar \*/* override fun onQueryTextSubmit(query: String?): Boolean {  
 */\*\* Nos pasa un query o una String vacía, pero no null \*/* buscarPorNombreSHLA(query.*orEmpty*())  
 return false  
 }  
  
 */\*\* Función que se llamará mientras se escribe, aquí quedará sin uso \*/* override fun onQueryTextChange(newText: String?) = false  
 })  
  
 */\*\* crear el adapter, no hay que pasar le parámetro porque se inicializó vacía con emptyList() \*/* /\* ahora le pasamos la función lambda para llamar al onclick \*/  
 adapterSHLA = SuperHeroAdapter **{**superHeroIdIT **->** irDetalleSuperHeroActivity(superHeroIdIT)**}** */\*\* variable para no repetir el binding \*/* val rvSuper = bindingSHLA.rvSuper  
 */\*\* Evitar que el recyclerView cambie de tamaño \*/* rvSuper.setHasFixedSize(true)  
 */\*\* Hay que pasarle un manager \*/* rvSuper.*layoutManager* = LinearLayoutManager(this)  
 */\*\* Pasarle el adapte que emos creado \*/* rvSuper.*adapter* = adapterSHLA  
}

Y con esto ya podemos navegar a la siguiente pantalla, aunque ahora está vacía.

Para pasar los datos a la nueva activity, hay que recuperar el putExtra **id** que le pasemos en el último intent:

import com.nanduky.cursokotlindesde0.databinding.ActivityDetallesSuperHeroBinding  
  
class DetallesSuperHeroActivity : AppCompatActivity() {  
  
 /\* Companion para acceder a su contenido desde cualquier lugar \*/  
 companion object{  
 /\* Constante para el putExtra\*/  
 const val EXTRA\_ID = "extra\_id"  
 }  
  
 */\*\* Crear el binding \*/* private lateinit var bindingDSHA: ActivityDetallesSuperHeroBinding  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *enableEdgeToEdge*()  
 */\*\* Inflar el binding \*/* bindingDSHA = ActivityDetallesSuperHeroBinding.inflate(*layoutInflater*)  
 setContentView(bindingDSHA.*root*)  
 ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.*main\_detalles\_super*)) **{** v, insets **->** val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())  
 v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)  
 insets  
 **}** /\* Recuperar el putExtra de irDetalleSuperHeroActivity de la activity principal\*/  
 val idPutExtra = *intent*.getStringExtra(EXTRA\_ID)  
 }  
}

Si ponemos el cursor encima de esta variante (**idPutExtra)** y le damos a la bombillita y seleccionamos que nos **Specify type explicity,**  nos dice que es una **string nullable:**

/\* Recuperar el putExtra de irDetalleSuperHeroActivity de la activity principal\*/  
val idPutExtra: String? = *intent*.getStringExtra(EXTRA\_ID)

No queremos que sea nulo. Le decimos que nos devuelva la string con la **id o vacía**:

/\* Recuperar el putExtra de irDetalleSuperHeroActivity de la activity principal\*/  
val idPutExtra: String = *intent*.getStringExtra(EXTRA\_ID).*orEmpty*()

Se le pada un método que ser´el encargado de hacer la petición a internet, y lo creamos también:

*/\*\* Recuperar el putExtra de irDetalleSuperHeroActivity de la activity principal\*/* val idPutExtra: String = *intent*.getStringExtra(EXTRA\_ID).*orEmpty*()  
 /\* Nuevo método que se encargará de hacer la petición a internet \*/  
 getSuperHeroInformacion(idPutExtra)  
 }  
  
 /\* Método que se encargará de hacer la petición a internet \*/  
 private fun getSuperHeroInformacion(idPutExtra: String) {  
 *TODO*("Not yet implemented")  
 }  
}

Volvemos a crear la función de retrofit:

...  
 /\* Método que se encargará de hacer la petición a internet \*/  
 private fun getSuperHeroInformacion(idPutExtra: String) {  
 //*TODO("Not yet implemented")* }  
  
 */\*\* Creación de Retrofit \*/* private fun getRetrofitDHA(): Retrofit {  
 return Retrofit  
 .Builder()  
 .baseUrl("https://superheroapi.com/")  
 */\*\* Siempre con el "/" final \*/* .addConverterFactory(GsonConverterFactory.create())  
 */\*\* para convertir \*/* .build()  
 */\*\* Construye te \*/* }  
  
}

En el método / función **getSuperHeroInformacion** Volvemos a llamar al hilo secundario:

*/\*\* Método que se encargará de hacer la petición a internet \*/*private fun getSuperHeroInformacion(idPutExtra: String) {  
 /\* Volver a llamar al hilo secundario \*/  
 *CoroutineScope*(Dispatchers.IO).*launch* **{**  
 **}**}

Tambien llamamos al retrofit:

*/\*\* Método que se encargará de hacer la petición a internet \*/*private fun getSuperHeroInformacion(idPutExtra: String) {  
 /\* Volver a llamar al hilo secundario \*/  
 *CoroutineScope*(Dispatchers.IO).*launch* **{** /\* Llamamos al retrofit \*/  
 getRetrofitDSHA().create(ApiServiceSuperInterface::class.*java*)  
 **}**}

Pero aquí hace falta pasarle un método y sólo tenemos creado en la API el **getSuperHeroInterface** y este no nos vale, hace falta uno que busque por la **id** y no por el nombre.

Vamos al **ApiServiceSuperInterface**  y creamos un nuevo método

package com.nanduky.cursokotlindesde0.superheroapp  
  
import retrofit2.Response  
import retrofit2.http.GET  
import retrofit2.http.Path  
  
interface ApiServiceSuperInterface {  
 //https://superheroapi.com/api/access-token/search/name  
 */\*\* Identificamos el tipo de llamada que vamos a hacer. Esta API sólo tiene GET \*/* @GET("/api/3609252f23e4fc59e44367838366f23b/search/{nombre}")  
 */\*\* https://superheroapi.com/ -> en retrofit de SuperHeroListActivity \*/  
 /\*\* Aquí no son private, sino que son suspend \*/* suspend fun getSuperHeroInterface(@Path("nombre") superHeroNombreInterface: String):  
 Response<SuperHeroDataClassResponse>  
  
 //https://superheroapi.com/api/access-token/character-id  
 @GET("/api/3609252f23e4fc59e44367838366f23b/{idSuperHero}")  
 */\*\* https://superheroapi.com/ -> en retrofit de DetallesSuperHeroActivity \*/* suspend fun getSuperHeroIdInterface(@Path("idSuperHero") superHeroIdInterface: String):  
 Response</\* nueva data class\*/>  
}

Hace falta una nueva **data class** para este método:

package com.nanduky.cursokotlindesde0.superheroapp  
  
import com.google.gson.annotations.SerializedName  
  
data class SuperHeroDataClassDetallesResponse(  
 @SerializedName("name") val nombreSuperSHDCDR : String,  
 @SerializedName("powerstats") val powerstatsSuperSHDCDR : PowerStarItemResponse)  
  
data class PowerStarItemResponse(  
  
 @SerializedName("intelligence") val inteligencia: String,  
 @SerializedName("strength") val fuerza: String,  
 @SerializedName("speed") val velocidad: String,  
 @SerializedName("durability") val durabilidad: String,  
 @SerializedName("power") val poder: String,  
 @SerializedName("combat") val combate: String,  
)

Al **ApiServiceSuperInterface**  ya se le puede pasar la **data class**:

package com.nanduky.cursokotlindesde0.superheroapp  
  
import retrofit2.Response  
import retrofit2.http.GET  
import retrofit2.http.Path  
  
interface ApiServiceSuperInterface {  
 //https://superheroapi.com/api/access-token/search/name  
 */\*\* Identificamos el tipo de llamada que vamos a hacer. Esta API sólo tiene GET \*/* @GET("/api/3609252f23e4fc59e44367838366f23b/search/{nombre}")  
 */\*\* https://superheroapi.com/ -> en retrofit de SuperHeroListActivity \*/  
 /\*\* Aquí no son private, sino que son suspend \*/* suspend fun getSuperHeroInterface(@Path("nombre") superHeroNombreInterface: String):  
 Response<SuperHeroDataClassResponse>  
  
 //https://superheroapi.com/api/access-token/character-id  
 @GET("/api/3609252f23e4fc59e44367838366f23b/{idSuperHero}")  
 */\*\* https://superheroapi.com/ -> en retrofit de DetallesSuperHeroActivity \*/* suspend fun getSuperHeroIdInterface(@Path("idSuperHero") superHeroIdInterface: String):  
 Response<SuperHeroDataClassDetallesResponse>  
}

Si volvemos a **DetallesSuperHeroActivity,** al ,método **getSuperHeroInformacion** ya le podemos pasar el nuevo método creado:

*/\*\* Método que se encargará de hacer la petición a internet \*/*private fun getSuperHeroInformacion(idPutExtra: String) {  
 /\* Volver a llamar al hilo secundario \*/  
 *CoroutineScope*(Dispatchers.IO).*launch* **{** /\* Llamamos al retrofit \*/ getRetrofitDSHA().create(ApiServiceSuperInterface::class.*java*).getSuperHeroIdInterface(idPutExtra)  
 **}**

Y lo metemos en una variable:

*/\*\* Método que se encargará de hacer la petición a internet \*/*private fun getSuperHeroInformacion(idPutExtra: String) {  
 */\*\* Volver a llamar al hilo secundario \*/  
 CoroutineScope*(Dispatchers.IO).*launch* **{** /\* Llamamos al retrofit y lo guardamos en una variable\*/  
 val miDetalleRespuesta =  
 getRetrofitDSHA().create(ApiServiceSuperInterface::class.*java*).getSuperHeroIdInterface(idPutExtra)  
 **}**}

Y con elresto de código queda así:

*/\*\* Método que se encargará de hacer la petición a internet \*/*private fun getSuperHeroInformacion(idPutExtra: String) {  
 */\*\* Volver a llamar al hilo secundario \*/  
 CoroutineScope*(Dispatchers.IO).*launch* **{** /\* Llamamos al retrofit y lo guardamos en una variable\*/  
 val miDetalleRespuesta =  
 getRetrofitDSHA().create(ApiServiceSuperInterface::class.*java*).getSuperHeroIdInterface(idPutExtra)  
 /\* Aqui simplificamos los 2 if de SuperHeroListActivity/buscarPorNombreSHLA a 1 \*/  
 if (miDetalleRespuesta.body() != null) {  
 /\* Llamamos la hilo principal \*/  
 runOnUiThread**{** /\* Creamos una función para sacar código de aquí y quede más ordenado \*/  
 creaUI(  
 /\* le pasamos de parámetro el body() y le decimos que no es nulo seguro \*/  
 miDetalleRespuesta.body()!! 🡨 hay que crea la funcion  
 )  
 **}** }  
 **}**}

Tenemos que rellenar la vista, el xml que ahora no hay nada. Paso el código completo del 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"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:id="@+id/main\_detalles\_super"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".superheroapp.DetallesSuperHeroActivity">  
  
 <androidx.appcompat.widget.AppCompatImageView  
 android:id="@+id/im\_superhero"  
 android:layout\_width="match\_parent"  
 android:layout\_height="250dp"  
 android:scaleType="centerCrop"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <androidx.cardview.widget.CardView  
 android:layout\_width="0dp"  
 android:layout\_height="0dp"  
 android:layout\_marginTop="215dp"  
 android:elevation="8dp"  
 app:cardCornerRadius="42dp"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent">  
  
 <androidx.appcompat.widget.LinearLayoutCompat  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical">  
  
 <TextView  
 android:id="@+id/tv\_super\_hero\_nombre"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:gravity="center\_horizontal"  
 android:textColor="@color/black"  
 android:textSize="28sp"  
 android:textStyle="bold"  
 tools:text="Texto prueba" />  
  
 <TextView  
 android:id="@+id/tv\_super\_hero\_nombre\_real"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:gravity="center\_horizontal"  
 android:textColor="@color/black"  
 android:textStyle="italic"  
 tools:text="Texto prueba" />  
  
 <TextView  
 android:id="@+id/tv\_super\_hero\_editorial"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:gravity="end"  
 android:layout\_marginEnd="8dp"  
 android:textStyle="italic"  
 android:textSize="11sp"  
 tools:text="Texto prueba" />  
  
 <androidx.appcompat.widget.LinearLayoutCompat  
 android:layout\_width="match\_parent"  
 android:layout\_height="100dp"  
 android:layout\_marginHorizontal="16dp"  
 android:layout\_marginTop="24dp"  
 android:gravity="bottom"  
 android:orientation="horizontal">  
  
 <View  
 android:id="@+id/view\_inteligencia"  
 android:layout\_width="0dp"  
 android:layout\_height="1dp"  
 android:layout\_marginHorizontal="4dp"  
 android:layout\_weight="1"  
 android:background="@color/superhero\_stat\_inteligencia" />  
  
 <View  
 android:id="@+id/view\_fuerza"  
 android:layout\_width="0dp"  
 android:layout\_height="1dp"  
 android:layout\_marginHorizontal="4dp"  
 android:layout\_weight="1"  
 android:background="@color/superhero\_stat\_fuerza" />  
  
 <View  
 android:id="@+id/view\_velocidad"  
 android:layout\_width="0dp"  
 android:layout\_height="1dp"  
 android:layout\_marginHorizontal="4dp"  
 android:layout\_weight="1"  
 android:background="@color/superhero\_stat\_velocidad" />  
  
 <View  
 android:id="@+id/view\_durabilidad"  
 android:layout\_width="0dp"  
 android:layout\_height="1dp"  
 android:layout\_marginHorizontal="4dp"  
 android:layout\_weight="1"  
 android:background="@color/superhero\_stat\_durabilidad" />  
  
 <View  
 android:id="@+id/view\_poder"  
 android:layout\_width="0dp"  
 android:layout\_height="1dp"  
 android:layout\_marginHorizontal="4dp"  
 android:layout\_weight="1"  
 android:background="@color/superhero\_stat\_poder" />  
  
 <View  
 android:id="@+id/view\_combate"  
 android:layout\_width="0dp"  
 android:layout\_height="1dp"  
 android:layout\_marginHorizontal="4dp"  
 android:layout\_weight="1"  
 android:background="@color/superhero\_stat\_combate" />  
  
 </androidx.appcompat.widget.LinearLayoutCompat>  
  
 <androidx.appcompat.widget.LinearLayoutCompat  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginHorizontal="16dp"  
 android:layout\_marginTop="4dp"  
 android:gravity="bottom"  
 android:orientation="horizontal">  
  
 <TextView  
 android:id="@+id/tv\_inteligencia"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginHorizontal="4dp"  
 android:layout\_weight="1"  
 android:maxLines="1"  
 android:ellipsize="marquee"  
 android:text="inteligencia" />  
  
 <TextView  
 android:id="@+id/tv\_fuerza"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginHorizontal="4dp"  
 android:layout\_weight="1"  
 android:maxLines="1"  
 android:ellipsize="marquee"  
 android:text="fuerza" />  
  
 <TextView  
 android:id="@+id/tv\_velocidad"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginHorizontal="4dp"  
 android:layout\_weight="1"  
 android:maxLines="1"  
 android:ellipsize="marquee"  
 android:text="velocidad" />  
  
 <TextView  
 android:id="@+id/tv\_durabilidad"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginHorizontal="4dp"  
 android:layout\_weight="1"  
 android:maxLines="1"  
 android:ellipsize="marquee"  
 android:text="durabilidad" />  
  
 <TextView  
 android:id="@+id/tv\_poder"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginHorizontal="4dp"  
 android:layout\_weight="1"  
 android:maxLines="1"  
 android:ellipsize="marquee"  
 android:text="poder" />  
  
  
 <TextView  
 android:id="@+id/tv\_combate"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginHorizontal="4dp"  
 android:layout\_weight="1"  
 android:maxLines="1"  
 android:ellipsize="marquee"  
 android:text="combate" />  
  
  
 </androidx.appcompat.widget.LinearLayoutCompat>  
  
 </androidx.appcompat.widget.LinearLayoutCompat>  
 </androidx.cardview.widget.CardView>  
  
</androidx.constraintlayout.widget.ConstraintLayout>

Y el resto del código **(SuperHeroDAtaClasDetalleResponse):**

package com.nanduky.cursokotlindesde0.superheroapp  
  
import com.google.gson.annotations.SerializedName  
  
data class SuperHeroDataClassDetallesResponse(  
 @SerializedName("name") val nombreSuperSHDCDR : String,  
 @SerializedName("powerstats") val powerstatsSuperSHDCDR : PowerStarItemResponse,  
 @SerializedName("image") val imageSuperSHDCDR : ImageSuperResponse,  
 @SerializedName("biography") val biografiaSHDCDR: BiographySuperResponse  
 )  
  
data class PowerStarItemResponse(  
  
 @SerializedName("intelligence") val inteligencia: String,  
 @SerializedName("strength") val fuerza: String,  
 @SerializedName("speed") val velocidad: String,  
 @SerializedName("durability") val durabilidad: String,  
 @SerializedName("power") val poder: String,  
 @SerializedName("combat") val combate: String,  
)  
  
data class ImageSuperResponse(  
 @SerializedName("url") val urlImageSuper:String  
)  
  
data class BiographySuperResponse(  
 @SerializedName("full-name") val nombreReal:String,  
 @SerializedName("publisher") val editorial:String  
)

y del **(DetalleSuperHeroActivity)**:

package com.nanduky.cursokotlindesde0.superheroapp  
  
import android.os.Bundle  
import android.util.TypedValue  
import android.view.View  
import androidx.activity.enableEdgeToEdge  
import androidx.appcompat.app.AppCompatActivity  
import androidx.core.view.ViewCompat  
import androidx.core.view.WindowInsetsCompat  
import com.nanduky.cursokotlindesde0.R  
import com.nanduky.cursokotlindesde0.databinding.ActivityDetallesSuperHeroBinding  
import com.squareup.picasso.Picasso  
import kotlinx.coroutines.CoroutineScope  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.launch  
import retrofit2.Retrofit  
import retrofit2.converter.gson.GsonConverterFactory  
import kotlin.math.roundToInt  
  
class DetallesSuperHeroActivity : AppCompatActivity() {  
  
 /\* Companion para acceder a su contenido desde cualquier lugar \*/  
 companion object {  
 /\* Constante para el putExtra\*/  
 const val EXTRA\_ID = "extra\_id"  
 }  
  
 */\*\* Crear el binding \*/* private lateinit var bindingDSHA: ActivityDetallesSuperHeroBinding  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *enableEdgeToEdge*()  
 */\*\* Inflar el binding \*/* bindingDSHA = ActivityDetallesSuperHeroBinding.inflate(*layoutInflater*)  
 setContentView(bindingDSHA.*root*)  
 ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.*main\_detalles\_super*)) **{** v, insets **->** val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())  
 v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)  
 insets  
 **}** */\*\* Recuperar el putExtra de irDetalleSuperHeroActivity de la activity principal\*/* val idPutExtra: String = *intent*.getStringExtra(EXTRA\_ID).*orEmpty*()  
 */\*\* Nuevo método que se encargará de hacer la petición a internet \*/* getSuperHeroInformacion(idPutExtra)  
 }  
  
 */\*\* Método que se encargará de hacer la petición a internet \*/* private fun getSuperHeroInformacion(idPutExtra: String) {  
 */\*\* Volver a llamar al hilo secundario \*/  
 CoroutineScope*(Dispatchers.IO).*launch* **{** */\*\* Llamamos al retrofit y lo guardamos en una variable\*/* val miDetalleRespuesta =  
 getRetrofitDSHA().create(ApiServiceSuperInterface::class.*java*)  
 .getSuperHeroIdInterface(idPutExtra)  
 */\*\* Aqui simplificamos los 2 if de SuperHeroListActivity/buscarPorNombreSHLA a 1 \*/* if (miDetalleRespuesta.body() != null) {  
 */\*\* Llamamos la hilo principal \*/* runOnUiThread **{** */\*\* Creamos una función para sacar código de aquí y quede más ordenado \*/* creaUI(  
 */\*\* le pasamos de parámetro el body() y le decimos que no es nulo seguro \*/* miDetalleRespuesta.body()!!  
 )  
 **}** }  
 **}** }  
  
 private fun creaUI(superHeroBody: SuperHeroDataClassDetallesResponse) {  
 /\* Pasarle a la vista la imagen \*/  
 Picasso.get().load(superHeroBody.imageSuperSHDCDR.urlImageSuper)  
 .into(bindingDSHA.imSuperhero)  
 /\* Pasar el nombre del super \*/  
 bindingDSHA.tvSuperHeroNombre.*text* = superHeroBody.nombreSuperSHDCDR  
 /\* Pasarle el nombre real \*/  
 bindingDSHA.tvSuperHeroNombreReal.*text* = superHeroBody.biografiaSHDCDR.nombreReal  
 bindingDSHA.tvSuperHeroEditorial.*text* = superHeroBody.biografiaSHDCDR.editorial  
 /\* Función para pintar la grafica de los stat \*/  
 preparaStats(superHeroBody.powerstatsSuperSHDCDR)  
 }  
  
 /\* Función para modificar los valores del xml \*/  
 private fun preparaStats(powerstatsSuperSHDCDR: PowerStarItemResponse) {  
 /\* Se llama a la función pasando le el nombre de la view y el int de @SerializedName("intelligence") val inteligencia: String, \*/  
 actualizaAltura(bindingDSHA.viewInteligencia, powerstatsSuperSHDCDR.inteligencia)  
 actualizaAltura(bindingDSHA.viewFuerza, powerstatsSuperSHDCDR.fuerza)  
 actualizaAltura(bindingDSHA.viewVelocidad, powerstatsSuperSHDCDR.velocidad)  
 actualizaAltura(bindingDSHA.viewDurabilidad, powerstatsSuperSHDCDR.durabilidad)  
 actualizaAltura(bindingDSHA.viewPoder, powerstatsSuperSHDCDR.poder)  
 actualizaAltura(bindingDSHA.viewCombate, powerstatsSuperSHDCDR.combate)  
  
 }  
  
 /\* Función para cambiar los parámetro a la view y no repetir código \*/  
 private fun actualizaAltura(view: View, stat: String) {  
 /\* Se crea una variable que contendrá los parámetros de los layouts de la vista \*/  
 val parametros = view.*layoutParams* /\* a los parámetros de la vista que recibe, se le cambia la altura con la fun pxToDp \*/  
 parametros.height = pxToDp(stat.*toFloat*())  
 /\* a la vista que llega por parámetros se le pasa los nuevos parámetros \*/  
 view.*layoutParams* = parametros  
  
 }  
  
 /\* Función para convertir los pixeles a DensidadPantalla \*/  
 private fun pxToDp(px: Float): Int {  
 return TypedValue.applyDimension(TypedValue.*COMPLEX\_UNIT\_DIP*, px, *resources*.*displayMetrics*)  
 .*roundToInt*()  
 }  
  
 */\*\* Creación de Retrofit \*/* private fun getRetrofitDSHA(): Retrofit {  
 return Retrofit  
 .Builder()  
 .baseUrl("https://superheroapi.com/")  
 */\*\* Siempre con el "/" final \*/* .addConverterFactory(GsonConverterFactory.create())  
 */\*\* para convertir \*/* .build()  
 */\*\* Construye te \*/* }  
  
}

**\*\*\* ACLARACIÓN \*\*\***

**\*\*\* FIN ACLARACIÓN \*\*\***

fin