Apuntes

Descargar estos apuntes

Resumen Interfaz de Usuario RecyclerView

Índice

Spinner

Spinners.

```
<Spinner
       android:id="@+id/spinner"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content" />
```

A continuación debemos definir el adaptador, en nuestro caso un objeto ArrayAdapter, que será el objeto que lanzaremos con nuestro Spinner.

```
class MainActivity : AppCompatActivity(),
                     AdapterView.OnItemSelectedListener {
   var colores = arrayOf("Rojo","Verde","Azul")
   lateinit  var listaColores: Spinner
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_main)
      //Si tuviera que cargar colores desde un xml
      //colores = resources.getStringArray(R.array.colores)
       val adaptador = ArrayAdapter(this, android.R.layout.
                                    simple_list_item_1, colores)
       listaColores = findViewById(R.id.spinner)
       listaColores.adapter=adaptador
       listaColores.setOnItemSelectedListener(this)
   }
```

```
   override fun onItemSelected(adapterView: AdapterView<*>?,
               view: View?, i: Int, l: Long){
  Toast.makeText(this,
      "El color elegido es: " +colores[i],
      Toast.LENGTH_LONG).show()
}
```

Podríamos definir los valores en un xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="colores">
        <item>ROJO</item>
        <item>VERDE</item>
        <item>AZUL</item>
    </string-array>
</resources>
```

RecyclerView

[RecyclerView]

Lo más habitual será implementar el Adapter y el ViewHolder, utilizar alguno de los LayoutManager predefinidos

> Añadir el recycler al lugar donde queremos que sea mostrado:
>
> ```
> <androidx.recyclerview.widget.RecyclerView
>   android:id="@+id/recyclerList"
>   android:background="@color/azul"
>   android:layout_width="match_parent"
>   android:layout_height="match_parent"/>
> ```

> También tenemos que pensar cual es el diseño que deseamos para cada uno de los elementos del recycler, esto se hace en un recurso layout, por ejemplo **recyclerlayout.xml**:
> ✋ Para conseguir una separación entre los elementos de la lista, se puede usar el contenedor `CardView` que ya conocemos de temas anteriores. RecyclerView no tiene la separación por elementos creada por defecto, se tendría que simular de distintas maneras, una es usando tarjetas como en este caso.

> El siguiente paso sería escribir nuestro adaptador. Este adaptador deberá extender de la clase `RecyclerView.Adapter` , de la cual tendremos que sobrescribir principalmente tres métodos:
>
> - *`onCreateViewHolder()`* . Encargado de crear los nuevos objetos ViewHolder necesarios para los elementos de la colección.
> - *`onBindViewHolder()`* . Encargado de actualizar los datos de un ViewHolder ya existente.
> - *`onItemCount()`* . Indica el número de elementos de la colección de datos.
>
> El método `onCreateViewHolder` devuelve un objeto de tipo **Holder**, por lo que primero deberemos crear una clase que herede de `RecyclerView.ViewHolder` , que podría ser como la siguiente, **Holder.kt**:

```
class Holder(v: View) : RecyclerView.ViewHolder(v) {
   val textNombre: TextView
   val textApellido: TextView
  private lateinit var binding: RecyclerlayoutBinding

   init {
       binding = RecyclerlayoutBinding.bind(v)
       textNombre = binding.textView
       textApellido = binding.textView2
   }

   fun bind(entity: Usuario) {
       textNombre.setText(entity.nombre)
       textApellido.setText(entity.apellidos)
   }
}
```

Ahora ta podemos crear el adaptador, que en nuestro proyecto podría ser **Adaptador.kt** que herede de `RecyclerView.Adapter` y que nos obligará a sobrescribir los métodos que sean necesarios, quedando el código como vemos en la imagen:

```
class Adaptador(val datos: ArrayList<Usuario>) :
                    RecyclerView.Adapter<Holder>()
{
   override fun onCreateViewHolder(viewGroup: ViewGroup,
                                   i: Int):Holder
   {
       val itemView: View =
          LayoutInflater.from(viewGroup.context)
           .inflate(R.layout.recyclerlayout, viewGroup, false)
       return Holder(itemView)
   }
   override fun onBindViewHolder(holder: Holder, position: Int) {
       val item: Usuario = datos[position]
       holder.bind(item)
   }
   override fun getItemCount(): Int {
       return datos.size
   }
}
```

Con esto tendríamos finalizado el adaptador, por lo que ya podríamos asignarlo al RecyclerView en nuestra actividad principal. Lo haremos con el siguiente código

```
class MainActivity : AppCompatActivity() {
   private lateinit var binding: ActivityMainBinding
   override fun onCreate(savedInstanceState: Bundle?)
   {
       super.onCreate(savedInstanceState)
       binding = ActivityMainBinding.inflate(layoutInflater)
       val view = binding.root
       setContentView(view)

       val datos = anadirDatos()
       val recyclerView = binding.mirecycler
       val adaptador = Adaptador(datos)
       recyclerView.adapter = adaptador
       recyclerView.layoutManager =
           LinearLayoutManager(this,
                           LinearLayoutManager.VERTICAL,
                           false)
   }

   private fun anadirDatos():ArrayList<Usuario>
   {
       var datos = ArrayList<Usuario>()
       for (i in 0..19)
           datos.add(Usuario("nombre$i",
                           "apellido1$i Apellido2$i"))
       return datos
   }
}
```

Click sobre un elemento de la lista

Para sorpresa de todos, la clase RecyclerView no tiene incluye un evento onItemClick(). RecyclerView delegará esta tarea al adaptador. Aprovecharemos la creación de cada nuevo ViewHolder para asignar a su vista asociada el evento onClick. Adicionalmente, para poder hacer esto desde fuera del adaptador, incluiremos el listener correspondiente como atributo del adaptador, y dentro de éste nos limitaremos a asignar el evento a la vista del nuevo ViewHolder y a lanzarlo cuando sea necesario desde el método onClick().

```
class Adaptador internal constructor(val datos: ArrayList<Usuario>) :
    RecyclerView.Adapter<Holder>(),View.OnClickListener
{
  lateinit var listenerClick:View.OnClickListener;
  override fun onCreateViewHolder(viewGroup: ViewGroup, i: Int):Holder{
        val itemView: View = LayoutInflater.from(viewGroup.context)
                    .inflate(R.layout.recyclerlayout, viewGroup, false)
        itemView.setOnClickListener(this)
        return Holder(itemView)
}
override fun onBindViewHolder(holder: Holder, position: Int) {
    val item: Usuario = datos[position]
    holder.bind(item)
}
override fun getItemCount(): Int {
    return datos.size
}
fun onClick(listener:View.OnClickListener){
    this.listenerClick=listener
}
override fun onClick(p0: View?) {
    listenerClick?.onClick(p0)
}
}
```

> 📌 El método onClick será al que tendremos que llamar desde donde queramos detectar el evento. Esto lo podemos ver en el código de la **MainActivity**.

```
adaptador.onClick(View.OnClickListener { valor ->
    Toast.makeText(
        this@MainActivity,
        "Has pulsado" + recyclerView.getChildAdapterPosition(valor),
        Toast.LENGTH_SHORT
        ).show()
        })
```

Click en cualquier lugar de la vista

Si quisiéramos detectar la pulsación de cualquier elemento de la línea del recycler, deberemos actuar sobre esa vista en el Holder (es donde podemos hacer referencia a cada uno de los view del layout). La forma de hacerlo es igual como se ha hecho anteriormente, usando un listener de la interfaz que nos haga falta y mandando la información a través de esta.

Para ver el funcionamiento, primero vamos a incluir una imagen en el layout **recyclerlayout.xml**, para que al pulsar sobre esa imagen se abra el dial del teléfono.

Podríamos añadir la imagen de la siguiente manera:

```
<androidx.cardview.widget.CardView>
  ...
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/imagen"
            android:src="@android:drawable/ic_menu_call"/>
    </LinearLayout>
</androidx.cardview.widget.CardView>
```

Ahora añadiremos en el código de la clase Holder (aquí es donde se sabe que vista se ha pulsado de entres todas), la llamada al intel que abre el díal:

```
class Holder(v: View, context: Context) : RecyclerView.ViewHolder(v),
                                          View.OnClickListener {
    val textNombre: TextView
    val textApellido: TextView
    val context:Context
    val imagen:ImageView
    fun bind(entity: Usuario) {
        textNombre.setText(entity.nombre)
        textApellido.setText(entity.apellidos)
    }
    init {
        this.context=context
        textNombre = v.findViewById(R.id.textView)
        textApellido = v.findViewById(R.id.textView2)
        imagen=v.findViewById(R.id.imagen)
        imagen.setOnClickListener(this)
    }
    override fun onClick(p0: View?) {
        val i = Intent(Intent.ACTION_DIAL)
        startActivity(context,i,null)
    }
}
```

> 📌 Para lanzar el intent necesitaremos el contexto, por lo que se lo pasaremos mediante el constructor del Holder. Al constructor del adaptador también le tendrá que llegar, de la misma manera, el contexto de la Actividad.

Click en cualquier lugar de la vista pasando información a la Actividad Principal

En este caso además de detectar la pulsación sobre un elemento, tendremos que devolver la información que nos interese hacia atrás. No se puede usar un ViewModel en el Holder, ya que no tiene ciclo de vida. ViewModel sólo podemos utilizarlo cuando hay ciclo de vida, como en los Fragments o Activities. Vamos a utilizar una interfaz

```
    class Holder(v: View) : RecyclerView.ViewHolder(v),
                                                View.OnClickListener {
        val textNombre: TextView
        val textApellido: TextView
5       lateinit var pasarCadenaInterface: PasarCadenaInterface
        fun bind(entity: Usuario) {
            textNombre.setText(entity.nombre)
            textApellido.setText(entity.apellidos)
        }
        init {
            textNombre = v.findViewById(R.id.textView)
            textApellido = v.findViewById(R.id.textView2)
13          textNombre.setOnClickListener(this)
14          textApellido.setOnClickListener(this)
        }
        override fun onClick(p0: View?) {
            var cadena:String
            if(p0?.id==R.id.textView) cadena=textNombre.text.toString()
            else cadena=textApellido.text.toString()
20          pasarCadenaInterface.pasarCadena(cadena)
        }
        fun pasarCadena(pasarCadenaInterface: PasarCadenaInterface)
        {
24          this.pasarCadenaInterface=pasarCadenaInterface
        }
    }
```

📌 **Línea 5** creamos un objeto del tipo de Interface creada, en este caso sería así:

```
interface PasarCadenaInterface{
   fun pasarCadena(cadena:String)
}
```

En el **Adaptador** tendremos que hacer los siguientes cambios:

```
class Adaptador internal constructor(val datos: ArrayList<Usuario>) :
    RecyclerView.Adapter<Holder>(),View.OnClickListener,
    View.OnLongClickListener{
    ...
    lateinit var pasarCadenaInterface: PasarCadenaInterface
    override fun onCreateViewHolder(viewGroup: ViewGroup, i: Int):Holder{
        ...
        val holder=Holder(itemView)
        holder.pasarCadena(object :PasarCadenaInterface{
            override fun pasarCadena(cadena: String) {
                pasarCadenaInterface.pasarCadena(cadena)
            }
        })
        return holder
    }
    ...
    fun pasarCadena(pasarCadenaInterface: PasarCadenaInterface)
    {
        this.pasarCadenaInterface=pasarCadenaInterface
    }
}
```

En la **Main** tendremos que llamar a este último método de la misma forma que en el adaptador.

```
adaptador.pasarCadena(object : PasarCadenaInterface {
          override fun pasarCadena(cadena: String) {
              Toast.makeText(
                  applicationContext,
                  "Has pulsado " +cadena,
                  Toast.LENGTH_SHORT ).show() }
      })
```

Detectar swipe izq/der sobre un elemento del recycler

Una opción muy utilizada en las listas, es la de controlar el deslizamiento a la izquierda o hacia la derecha sobre uno de sus elementos. Para ello tendremos que usar una clase que ya está definida y que tendremos que copiar en nuestro proyecto, llamada `SwipeDetector` (se pasa en los recursos). En el Adaptador tendríamos que implementar la interfaz onTouch, para que nos detecte el desplazamiento, esto lo haremos como lo hicimos en puntos anteriores. **Líneas marcadas en el Adaptador**.

```
class Adaptador internal constructor(val datos: ArrayList<Usuario>) :
    RecyclerView.Adapter<Holder>(),View.OnClickListener,
    View.OnLongClickListener, View.OnTouchListener
{
    lateinit var listenerClick:View.OnClickListener;
    lateinit var listenerLong:View.OnLongClickListener
    lateinit var listenerOnTouch:View.OnTouchListener

    override fun onCreateViewHolder(viewGroup: ViewGroup, i: Int): Holder {
        val itemView: View = LayoutInflater.from(viewGroup.context)
            .inflate(R.layout.recyclerlayout, viewGroup, false)
        itemView.setOnClickListener(this)
        itemView.setOnLongClickListener(this)
        itemView.setOnTouchListener(this)
        val holder=Holder(itemView)
        return holder
    }
    fun onTouch(listenerOnTouch:View.OnTouchListener)
    {
        this.listenerOnTouch=listenerOnTouch
    }
    override fun onTouch(p0: View?, p1: MotionEvent?): Boolean {
        listenerOnTouch.onTouch(p0,p1)
        return  false
    }
}
```

En la ActivityMain, tendremos que llamar al método `OnTouch` del adaptador pasándole un objeto de la clase `SwipeDetector` **Línea 2**, pero a la hora de detectar el movimiento tendremos que usar la interfaz onClikListener **Línea 3**, como vemos en el código siguiente. Si nos fijamos, podremos ver que el objeto swipeDetector detectar si el movimiento se ha producido a la derecha o a la izquierda.

```
val swipeDetector = SwipeDetector()
adaptador.onTouch(swipeDetector)
adaptador.onClick(View.OnClickListener { v ->
            if (swipeDetector.swipeDetected()) {
                when (swipeDetector.action) {
                    SwipeDetector.Action.LR -> {
                        Toast.makeText(
                            applicationContext,
                            "Has pulsado Izquierda",
                            Toast.LENGTH_SHORT
                        ).show()
                    }
                    SwipeDetector.Action.RL -> {
                        Toast.makeText(
                            applicationContext,
                            "Has pulsado Derecha",
                            Toast.LENGTH_SHORT
                        ).show()
                    }
                }
            } else
                Toast.makeText(
                    applicationContext,
                    "Has pulsado" + recyclerView.getChildAdapterPosition(v),
                    Toast.LENGTH_SHORT
                ).show()
        })
```

PASOS RESUMEN RECYCLERVIEW

1. Creamos el **Holder**:

```
class Holder(v: View) : RecyclerView.ViewHolder(v)
{
    //Declaramos las propiedades
    val textNombre: TextView

    init {
        //inicializamos el binding o v.findViewById
        //esto se hace en el contructor, una sola vez
        textNombre = v.findViewById(R.id.textView)
        //Si queremos tratar la pulsación sobre un elemento debemos
        //ponerle el escuchador al elemento
    }

    fun bind(entity: Usuario) {
        //Asignamos a la propiedad la de la clase POJO
        textNombre.setText(entity.nombre)

    }

    //En caso de querer
    override fun onClick{
        //Si queremos pasar un dato se realizará mediante interfaz
    }

}
```

2. Creamos el **Adaptador**

```
class Adaptador(val datos: ArrayList<Usuario>) :
                    RecyclerView.Adapter<Holder>()
//además heredará de los que necesite: OnClick, OnTouch...
{
   override fun onCreateViewHolder(viewGroup: ViewGroup,
                                   i: Int):Holder
   {
       //Se construye el Holder nuevo, teniendo en cuenta el layout
       // con el recycler que vamos a usar
       val itemView: View =
          LayoutInflater.from(viewGroup.context)
           .inflate(R.layout.recyclerlayout, viewGroup, false)
       val holder = Holder(itemView)

       //Aquí pondremos los escuchadores sobre los elementos que queramos
       //y adaptador deberá implementar la interfaz y la funcion onClick
       //itemView.setOnClickListener(this)

       //si hemos recibido el dato de una interfaz
       //holder.pasarDatos(object: PadarDatosInterfaz{
       // override fun pasarDatos(datos: Datos){
       //     //con la propiedad creada de esa interfaz,
       //     //llamamos el método pasarDatos creado
       //   }
       //})
       //No olvidar crear el método fun
       // pasarDatos que nos pasa la interfaz
       // y es llamado desde la Activity

       return Holder(itemView)
   }

   override fun onBindViewHolder(holder: Holder, position: Int) {
       //Actualizamos los holder existentes
       val item: Usuario = datos[position]
       holder.bind(item)
   }

   override fun getItemCount(): Int {
       //Devuelve el tamaño
       return datos.size
   }
}
```

3. Ya podemos asignar el **adaptador a la Activity**

```
class MainActivity : AppCompatActivity() {
   private lateinit var binding: ActivityMainBinding

   override fun onCreate(savedInstanceState: Bundle?)
   {
       super.onCreate(savedInstanceState)
       binding = ActivityMainBinding.inflate(layoutInflater)
       val view = binding.root
       setContentView(view)

       //val datos = anadirDatos()
       val recyclerView = binding.mirecycler
       val adaptador = Adaptador(datos)
       recyclerView.adapter = adaptador
       recyclerView.layoutManager =
           LinearLayoutManager(this,
                           LinearLayoutManager.VERTICAL,
                           false)
      //Aquí podemos llamar a la función creada para el adaptador
      //Podemos saber la posición pulsada getChildAdapterPosition()
      //adaptador.onClick{ .... }

       //Para actualizar el adaptador si eliminamos posicion
       //adaptador.notyfyItemRemoved(posicionEliminada)
   }

}
```