Bloque 9. Ejercicio Resuelto ActionMode

Descargar este ejercicio

Para crear una selección múltiple en un recyclerView, necesitaremos implementar por un lado los elementos que hacen posible la gestión de la selección, y por otro el elemento que nos muestra la barra contextual superpuesta con las opciones a realizar sobre estos elementos. Deberemos partir de un proyecto que tenga un recyclerView Funcionando. Por ejemplo, como el siguiente:

Primero se crearán los archivos con las vistas necesarias en la carpeta layout.

activity_main.xml que implementará el contenedor para el fragment que contendrá a su vez el recycler

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    tools:context=".MainActivity">
    <androidx.fragment.app.FragmentContainerView
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:defaultNavHost="true"
        app:navGraph="@navigation/navegacion_fragments"
        android:id="@+id/contenedor"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

FragmentRecycler.xml, solo tendrá el recycler que mostrará la lista, lo asociaremos con el fragment.

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" tools:context=".MainActivity">
    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerList"
        android:background="@color/azul"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

linea_recycler.xml será la vista que cargará la línea.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    card_view:cardCornerRadius="4dp"
    card_view:cardUseCompatPadding="true"
    card_view:cardElevation="2dp">
    <LinearLayout
        android:padding="8dp"
        android:background="#493DEC"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:orientation="horizontal">
        <LinearLayout
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="0.75"
            android:orientation="vertical">
            <TextView
                android:id="@+id/textView"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Large Text"
                android:textColor="@android:color/white"
                android:textSize="20sp" />
            <TextView
                android:id="@+id/textView2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Medium Text"
                android:textColor="@android:color/white"
                android:textSize="15sp" />
        </LinearLayout>
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/imagen"
            android:visibility="invisible"
            android:src="@android:drawable/star_big_off"/>
    </LinearLayout>
</androidx.cardview.widget.CardView>
```

Por otro lado necesitaremos las siguiente clases para su funcionamiento:

Usuario.kt

```
class Usuario(nombre:String, apellidos:String) {
    var nombre: String
    var apellidos: String
    init {
        this.nombre = nombre
        this.apellidos = apellidos
    }
}
```

ViewModel.kt que contendrá el MutableLiveData sobre una lista de Usuarios. Aprovecharemos el constructor para añadir los datos en esta lista.

```
class ViewModelUsuario: ViewModel() {
    var listaUsuarios:MutableLiveData<ArrayList<Usuario>>
    init{
        listaUsuarios=MutableLiveData<ArrayList<Usuario>>()
        setListaUsuarios(anadirDatos())
    }
    fun setListaUsuarios(lista:ArrayList<Usuario>)
    {
        listaUsuarios.value=lista
    }
    fun getLiveDataUsuarios()=listaUsuarios
    fun getUsuarios():ArrayList<Usuario> =listaUsuarios.value!!
    fun addUsuario(usuario:Usuario){
        listaUsuarios.value?.add(usuario)
    }
    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
    }
}
```

Holder.kt gestionará la carga de cada elemento de la línea del recycler.

```
class Holder(v: View, context: Context) : RecyclerView.ViewHolder(v) {
    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)
    }
}
```

Adapter.kt el adaptador con la gestión de cada elemento de la lista.

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

FragmentRecycler.kt que cargará los datos recuperados del ViewModel en el recycler, usando el adaptador.

```
class FragmentRecycler: Fragment() {

    var actionMode: ActionMode? = null
    val model:ViewModelUsuario by activityViewModels()
    lateinit var recyclerView:RecyclerView
    lateinit var adaptador:Adaptador

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view=inflater.inflate(R.layout.fragment_recycler,container,false)

        recyclerView = view.findViewById(R.id.recyclerList)
        adaptador = Adaptador(model.getUsuarios(),requireActivity())
        recyclerView.adapter = adaptador

        recyclerView.layoutManager =
            LinearLayoutManager(requireActivity(), LinearLayoutManager.VERTICAL, false)

        return view
    }
}
```

MainActivity.kt

```
class MainActivity : AppCompatActivity() {

    val model:ViewModelUsuario by viewModels()
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

Gestionar Selección

Para gestionar la selección usaremos dos tipo de datos que nos ayudarán en el proceso `ItemDetailsLookup<T>()` y `SelectionTracker<T>`. Como ya se comentó en los apuntes del tema8, el la clase que anule a `RecyclerView.ViewHolder` tendremos que añadir una función que devuelva los identificadores del elemento pulsado de tipo `ItemDetailsLookup.ItemDetails<Long>`, elegiremos el tipo Long para la clave porque es más sencillo de gestionar.

1. Añadir en **Holder.kt**

```
fun  getItemDetails():ItemDetailsLookup.ItemDetails<Long>
{
        var obj=object:ItemDetailsLookup.ItemDetails<Long>()
        {
            override fun getPosition(): Int {
                return adapterPosition
            }
            override fun getSelectionKey(): Long? {
               return itemId
            }
        }
        return obj
}
```

2. Además también tendremos que modificar el método **bind** de esta clase. De forma que le llegará un `SelectionTracker`, que utilizaremos para controlar si la vista a sido seleccionada y realizar los cambios correspondientes.

```
fun bind(entity: Usuario, t: SelectionTracker<Long>) {
        textNombre.setText(entity.nombre)
        textApellido.setText(entity.apellidos)
        if(t.isSelected(adapterPosition.toLong())) {
            imagen.setImageResource(android.R.drawable.star_big_on)
            imagen.visibility=View.VISIBLE
        }
        else
            imagen.visibility=View.INVISIBLE
    }
```

3. Crearemos una clase **CapturaVista** que herede de `ItemDetailsLookup<Long>()`, que accederá a este método para devolver el identificador de la vista seleccionada.

```
class CapturaVista(var rv:RecyclerView): ItemDetailsLookup<Long>() {
    override fun getItemDetails(e: MotionEvent): ItemDetails<Long>? {
        val view =rv.findChildViewUnder(e.x, e.y)
        if(view != null) {
            return (rv.getChildViewHolder(view) as Holder).getItemDetails()
        }
        return null
    }
}
```

4. En la clase **Adapter.kt** tendremos que añadir los dos métodos, el primero devolverá el id asociado con la posición y el segundo servirá para asignar el `SelectionTracker` al adaptador, para posteriormente pasarlo al Holder mediante la llamada al bind.

```
override fun getItemId(position: Int): Long {
        return position.toLong()
}
fun setTracker(tracker: SelectionTracker<Long>?) {
    if (tracker != null) {
        this.tracker = tracker
    }
}
```

Y modificaremos:

```
//..
//Añadiremos la propiedad tracker
private var tracker: SelectionTracker<Long>? = null
//...
override fun onBindViewHolder(...)
{
  //...
  holder.bind(item,tracker!!)
  //..
}
```

Acabar de Gestionar la Selección y añadir el CAB

5. La clase **FragmenRecycler.kt** tendrá algunos cambios más significativos, por lo que añadiremos todo el código:

```
class FragmentRecycler: Fragment() {
    var actionMode: ActionMode? = null
    val model:ViewModelUsuario by activityViewModels()
    lateinit var recyclerView:RecyclerView
    lateinit var adaptador:Adaptador
    lateinit var tracker: SelectionTracker<Long>
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view=inflater.inflate(R.layout.fragment_recycler,container,false)

        recyclerView = view.findViewById(R.id.recyclerList)
        recyclerView.setHasFixedSize(true)
        adaptador = Adaptador(model.getUsuarios(),requireActivity())
        //Indicamos que el adaptador debe tener id estables,
        //muy importante hacer antes de asignar al recycler
        adaptador.setHasStableIds(true)
        recyclerView.adapter = adaptador
        //Iniciamos el tracker
        tracker = SelectionTracker.Builder(
            "selecccion",
            recyclerView,
            StableIdKeyProvider(recyclerView),
            CapturaVista(recyclerView),
            StorageStrategy.createLongStorage()
        ).withSelectionPredicate(
            SelectionPredicates.createSelectAnything()
        ).build()
        //Pasamos el tracker al adaptador
        adaptador.setTracker(tracker)
        if(savedInstanceState != null)
            tracker.onRestoreInstanceState(savedInstanceState)
       //Aprovechamos el observer para activar la CAB
        tracker.addObserver(
            object: SelectionTracker.SelectionObserver<Long>() {
                override fun onSelectionChanged() {
                    if (tracker.hasSelection()) {
                        if (actionMode == null) {
                            actionMode = (requireActivity() as AppCompatActivity).
                            startSupportActionMode(actionModeCallback)
                        }
                        actionMode?.title = "${tracker.selection.size()}"
                    } else {
                        actionMode?.finish()
                    }
                }
            })
        recyclerView.layoutManager =
            LinearLayoutManager(requireActivity(), LinearLayoutManager.VERTICAL, false)
```

```
        return view
    }

    override fun onSaveInstanceState(outState: Bundle) {
        super.onSaveInstanceState(outState)
            tracker.onSaveInstanceState(outState)
    }

//OBJETO que crea y gestiona la CAB ActionMode
private val actionModeCallback = object : ActionMode.Callback {
        override fun onCreateActionMode(mode: ActionMode, menu: Menu): Boolean {
            requireActivity().menuInflater.inflate(R.menu.action_mode_menu, menu)
            return true
        }

        override fun onPrepareActionMode(mode: ActionMode, menu: Menu): Boolean {
            return false
        }

        override fun onActionItemClicked(mode: ActionMode, item: MenuItem): Boolean {
            return when (item.itemId) {
                R.id.delete -> {
                    tracker.selection.sorted().reversed().forEach { id ->
                        model.getUsuarios().removeAt(id.toInt())}
                    recyclerView.getRecycledViewPool().clear();
                    adaptador.notifyDataSetChanged();
                    tracker.clearSelection()
                    actionMode = null
                    true
                }

                else -> false
            }
        }

        override fun onDestroyActionMode(mode: ActionMode) {
            tracker.clearSelection()
            actionMode = null
        }
    }
}
```