# Bloque 10 . Ejercicio Resuelto Barra Progreso

Descargar este ejercicio

Vamos a partir del ejercicio resuelto **Barra de Progreso** del **Bloque 6**. Como se explica en este ejercicio, la manera de resolverlo con AsynTask ha sido deprecada en versiones recientes, por lo que ahora aplicaremos los conocimientos de `corrutinas` para resolverlo. Además tendremos en cuenta los reinicios de la aplicación por giros y demás.

**main_activity.xml**

Añadiremos un botón para iniciar el funcionamiento de la barra de progreso.

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

    <Button
        android:id="@+id/buttonProgress"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Load"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Además se deberá de tener un `layout` para la vista del dialogo personalizado. Podría ser un archivo como el siguiente.

**main_activity.xml**

```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:padding="13dp"
    android:id="@+id/Dialogoprogreso"
    android:layout_centerHorizontal="true"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <com.google.android.material.progressindicator.LinearProgressIndicator
        android:id="@+id/loader"
        style="@style/Widget.MaterialComponents.LinearProgressIndicator"
        android:layout_width="match_parent"
        android:layout_height="65dp"
        android:max="100"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

    <TextView
        android:id="@+id/loading_msg"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:text="Loading..."
        android:textAppearance="?android:textAppearanceSmall"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/loader"/>

    <TextView
        android:id="@+id/progres_msg"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="x/x"
        android:textAppearance="?android:textAppearanceSmall"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/loader" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

////////////////////////////////

Para este ejercicio vamos a usar dos elementos `navigation`. Para el caso de vista **vertical** utilizaremos un gráfo de navegación como el siguiente:

Como se puede ver en la imagen, también tendremos que crear un `FragmentDialog`, para que se muestre un dialogo al realizar una pulsación larga sobre algún elemento de la lista para ser eliminado.

Y para la vista horizontal usaremos otro, solo con la parte de navegación entre fragmentLista y el dialogEliminar. La navegación del elemento Detalle, se ha realizado usando el `FragmentManager` y `FragmentTransaction`

Para poder hacer los gráficos de navegación, antes debemos crear el código `Kotlin` necesario. Vamos a pasar a ello, junto con sus comentarios por líneas.

En cuanto al código del adaptador y del holder del recycler, será idéntico de como lo hacemos normalmente, en el que implementamos los listeners para el Cick corto y largo:

**Adapter.kt**

```
class Adapter(var datos: ArrayList<Datos>) :
                                RecyclerView.Adapter<Adapter.Holder>(),
                                View.OnClickListener, OnLongClickListener
{
    var listener: View.OnClickListener? = null
    var listenerLong: OnLongClickListener? = null
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
                                Holder {
        val view: View
        view = LayoutInflater.from(parent.context)
                        .inflate(R.layout.elemento_lista, parent, false)
        val holder = Holder(view)
        view.setOnClickListener(this)
        view.setOnLongClickListener(this)
        return holder
    }
    override fun onBindViewHolder(holder: Holder, position: Int) {
        (holder as Holder?)!!.bind(datos[position])
    }
    override fun getItemCount(): Int {
        return datos.size
    }
    fun miOnClick(listener: View.OnClickListener?) {
        this.listener = listener
    }
    override fun onClick(v: View) {
        if (listener != null) listener!!.onClick(v)
    }
    fun miOnLongClick(listener: OnLongClickListener?) {
        listenerLong = listener
    }
    override fun onLongClick(v: View): Boolean {
        if (listenerLong != null) listenerLong!!.onLongClick(v)
        return false
    }

    class Holder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        var lineTexto: TextView
        var imagen: ImageView
        var cardView: CardView
        fun bind(dato: Datos) {
            lineTexto.text = dato.nombre
            imagen.setImageResource(dato.icono)
        }
        init {
            lineTexto = itemView.findViewById(R.id.textView_titulo)
            imagen = itemView.findViewById(R.id.imageView_imagen_miniatura)
            cardView = itemView.findViewById(R.id.cardView)
        }
    }
}
```

La clase que implementa el detalle al pulsar sobre un elemento de la lista, quedaría con el siguiente código:

**FragmentDetalle.kt**

```
class FragmentDetalle : Fragment() {
    var mItem: Int? = null
    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
6       if(savedInstanceState!=null) mItem=savedInstanceState?.getInt("POSICION")
        else {
            if (arguments != null) {
9               mItem = arguments?.getInt("POSICION")
10          } else mItem = 0
        }
    }
     override  fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        val rootView: View = inflater.inflate(R.layout.fragment_detalle,
                                              container, false)
        //Mostramos el contenido al usuario
        if(MainActivity.datos!!.size > 0) {
            val elemento = MainActivity.datos?.get(mItem!!)
            (rootView.findViewById<TextView>(R.id.textView_superior)).text =
                                                    elemento?.nombre
            (rootView.findViewById<TextView>(R.id.textView_inferior)).text =
                                                    elemento?.desc
            (rootView.findViewById<ImageView>(R.id.imageView_imagen)).
                                          setImageResource(elemento!!.icono)
        }
        return rootView
    }
32  override fun onSaveInstanceState(outState: Bundle) {
        super.onSaveInstanceState(outState)
        outState.putInt("POSICION", mItem!!)
    }
}
```

📌 las líneas a destacar de este código son la creadas para guardar la posición pulsada en determinado momento, para el caso de que ocurra algún giro de pantalla no perder el elemento pulsado. Guardaremos la posición mediante el método `onSaveInstanceState` por el que se pasará justo antes de que la aplicación se destruya, **Línea 32**. Esta posición se podrá recuperar mediante el Bundle que entra en el método `onCreate` **Línea 6**. Las **Líneas 9 y 10**

> son para recuperar la posición cuando se pulsa el elemento de la lista en el `FragmentLista`, se recupera mediante la propiedad `arguments`

El código del fragment que contendrá el recycler, será una clase que derivará de Fragment y en la que inflaremos el layout recycler e implementaremos el OnClick, mandando la posición seleccionada mediante el Bundle a el Fragment Detalle:

**FragmentLista.kt**

```
class FragmentLista : Fragment() {
    var listenerLargo: View.OnLongClickListener? = null
    var recyclerView: RecyclerView? = null
    var valores: ArrayList<Datos>? = null
    var posicion:Int=0

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        super.onCreateView(inflater, container, savedInstanceState)
        val rootView: View = inflater.inflate(R.layout.fragment_recycler,
                                              container, false)
        recyclerView = rootView.findViewById(R.id.recycler)
        adapter = Adapter(MainActivity.datos!!)
        recyclerView?.adapter = adapter
        recyclerView?.setHasFixedSize(true)
        recyclerView?.layoutManager =LinearLayoutManager(getActivity(),
                                        LinearLayoutManager.VERTICAL, false)

        adapter!!.miOnClick { v ->
            val bundle = Bundle()
            bundle.putInt("POSICION", recyclerView!!.getChildAdapterPosition(v))
            val navController = NavHostFragment.findNavController(this)
            if (navController.currentDestination?.id == R.id.fragmentLista) {
                if (resources.configuration.orientation ==
                    Configuration.ORIENTATION_PORTRAIT) navController.navigate(
                    R.id.action_fragmentLista_to_fragmentDetalle,
                    bundle
                )
               else
                {
                    val fT = requireActivity().supportFragmentManager.
                                                            beginTransaction()
                    fT.replace(R.id.detalle,FragmentDetalle::class.java,bundle).
                                                            commit()
                }
            }
        }
        adapter!!.miOnLongClick { v ->
            val navController = NavHostFragment.findNavController(this)
            if (navController.currentDestination?.id == R.id.fragmentLista) {
                val bundle = Bundle()
                posicion=recyclerView!!.getChildLayoutPosition(v)
                bundle.putInt("POSICION",posicion)
                if (resources.configuration.orientation ==
                                                Configuration.ORIENTATION_PORTRAIT)
                 navController.navigate(R.id.action_fragmentLista_to_dialogEliminar,
                                        bundle)
                else {
```

```
                        val navHostFragment = requireActivity().
                        supportFragmentManager.findFragmentById(R.id.lista)
                        as NavHostFragment
                        val navControllerLista = navHostFragment.navController
                        navControllerLista.navigate(R.id.
                                        action_fragmentLista_to_dialogEliminar, bundle)
58                  }
                }
                true
            }

            return rootView
        }

         override fun onAttach(activity: Context) {
            super.onAttach(activity)
            try {
                valores = MainActivity.datos
                listenerLargo = activity as View.OnLongClickListener?
            } catch (e: ClassCastException) {  }
        }

        companion object {
            var adapter: Adapter? = null
        }
    }
```

📌Desde **Línea 24 a 38** al pulsar sobre un elemento de la lista se creará un Bundle para guardar la posición del elemento pulsado, dependiendo si el dispositivo está en posición horizontal o vertical: en el primer caso se navegará al fragment detalle utilizando la acción que habremos creado en el grafo de navegación, en el segundo utilizaremos la carga de fragment de forma tradicional. En el código que hay desde la **Línea 43 a 58**, también se tiene que tener en cuenta la posición del dispositivo, pero solo para usar uno u otro `NavHostFragment` , en todo caso navegaremos del elemento pulsado al dialogo que nos permite eliminar un elemento de la lista.

**DialogEliminar.kt**

```
class DialogEliminar : DialogFragment() {
    override  fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        val builder: AlertDialog.Builder = AlertDialog.Builder(getActivity())
        if (arguments != null) {
            val posicion = arguments?.getInt("POSICION")
            builder.setMessage("Deseas Eliminar el item?")
                .setTitle("AVISO")
                .setPositiveButton("OK", DialogInterface.OnClickListener {
                     dialog, id -> MainActivity.datos?.removeAt(posicion!!)
                    FragmentLista.adapter!!.notifyItemRemoved(posicion!!)})
        }
        return builder.create()
    }
}
```

> 📌Este dialogo fragment es igual a los que hemos visto en ejercicios anteriores, pasaremos la posición del elemento a eliminar mediante un bundle

En cuanto al código de la MainActivity, solo se encargará de rellenar los datos pasados como recurso y de guardar una instancia de estos datos en caso de que la aplicación pare de forma brusca, para que pueda ser recuperada al volver a iniciar.

```
class MainActivity : FragmentActivity(){
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        //Recuperar datos después giro
        if (savedInstanceState != null) {
            datos = savedInstanceState.getParcelableArrayList("DATOS")
        }
        else  datos = rellenarDatos()
    }

    override fun onSaveInstanceState(outState: Bundle) {
        super.onSaveInstanceState(outState)
        outState.putParcelableArrayList("DATOS", datos)
    }
    fun rellenarDatos(): ArrayList<Datos> {
        ...}

    companion object {
         var datos: ArrayList<Datos>? = null
    }
}
```