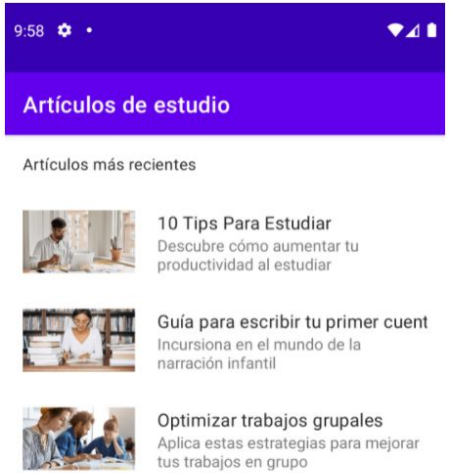
Programación multimedia y dispositivos móviles

UT-4. Listas. RecyclerView - 1

https://developer.android.com/guide/
https://www.sgoliver.net/blog/curso-de-programacion-android/indice-de-contenidos/

Controles de selección: RecyclerView

- **RecyclerView**: Este control llegó con Android 5.0 como una mejor alternativa a los antiguos controles ListView (lista) y GridView (tabla) de versiones anteriores. Su uso más común es la presentación de listas de ítems al usuario.



Controles de selección: RecyclerView

RecyclerView nos va a permitir mostrar en pantalla colecciones de datos. Pero RecyclerView no va a hacer «casi nada» por sí mismo, sino que se va a sustentar sobre otros componentes complementarios para determinar cómo acceder a los datos y cómo mostrarlos. Los más importantes serán los siguientes:

- RecyclerView.ViewHolder
- RecyclerView.Adapter
- LayoutManager
- ItemDecoration
- ItemAnimator

Controles de selección: RecyclerView.ViewHolder

El primer componente que necesitaremos será RecyclerView.ViewHolder.

Un view holder se encargará de contener y gestionar las vistas o controles asociados a cada elemento individual de la lista. El control RecyclerView se encargará de crear tantos view holder como sea necesario para mostrar los elementos de la lista que se ven en pantalla y los gestionará eficientemente de forma que no tenga que crear nuevos objetos para mostrar más elementos de la lista al hacer scroll, sino que tratará de «reciclar» aquellos que ya no sirven por estar asociados a otros elementos de la lista que ya han salido de la pantalla.

Nosotros necesitaremos crear una clase que herede de esta y en esa clase java únicamente necesitaremos “castear” aquellos controles de nuestras filas sobre los que queremos interactuar.

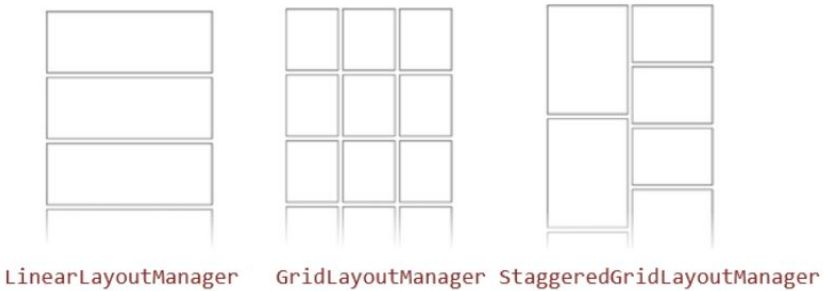
Controles de selección: RecyclerView.Adapter

El segundo componente es un adaptador y funciona de forma similar a como se utiliza en los spinners.

El adaptador nos permite trabajar con nuestros datos. En este caso un adaptador que herede de la clase RecyclerView.Adapter. La peculiaridad en esta ocasión es que este tipo de adaptador utilizará internamente el patrón view holder que dotará de una mayor eficiencia al control, y de ahí la necesidad del componente de la lista visto antes, RecyclerView.ViewHolder.

Controles de selección: RecyclerView

Una vista de tipo RecyclerView por el contrario no determina por sí sola la forma en que se van a mostrar en pantalla los elementos de nuestra colección, sino que va a delegar esa tarea a otro componente llamado LayoutManager, que también tendremos que crear y asociar al RecyclerView para su correcto funcionamiento. Por suerte, el SDK incorpora de serie algunos LayoutManager para las representaciones más habituales de los datos: lista vertical u horizontal (LinearLayoutManager), tabla tradicional (GridLayoutManager) y tabla apilada o de celdas no alineadas (StaggeredGridLayoutManager).



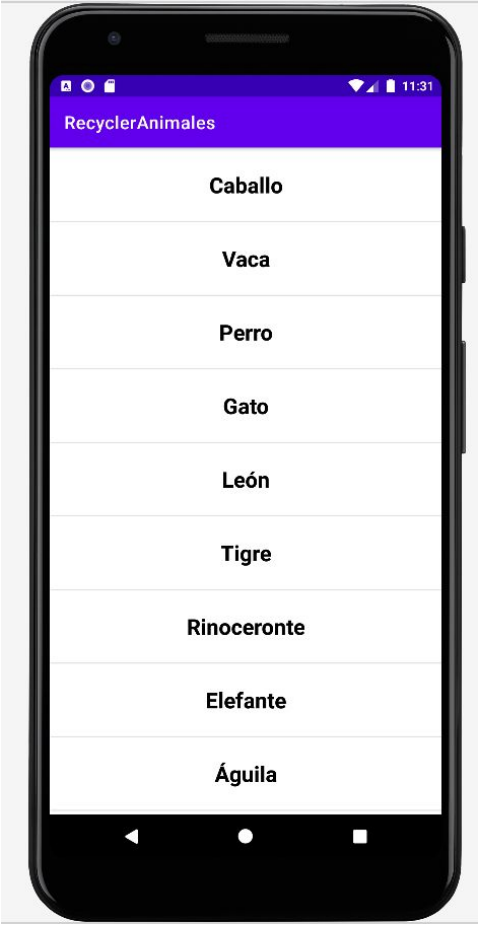
Controles de selección: RecyclerView

Los dos últimos componentes de la lista se encargarán de definir cómo se representarán algunos aspectos visuales concretos de nuestra colección de datos (más allá de la distribución definida por el LayoutManager), por ejemplo marcadores o separadores de elementos, y de cómo se animarán los elementos al realizarse determinadas acciones sobre la colección, por ejemplo al añadir o eliminar elementos.

No siempre será obligatorio implementar todos estos componentes para hacer uso de un RecyclerView. Lo más habitual será implementar el Adapter y el ViewHolder, utilizar alguno de los LayoutManager predefinidos, y sólo en caso de necesidad crear los ItemDecoration e ItemAnimator necesarios para dar un toque de personalización especial a nuestra aplicación.

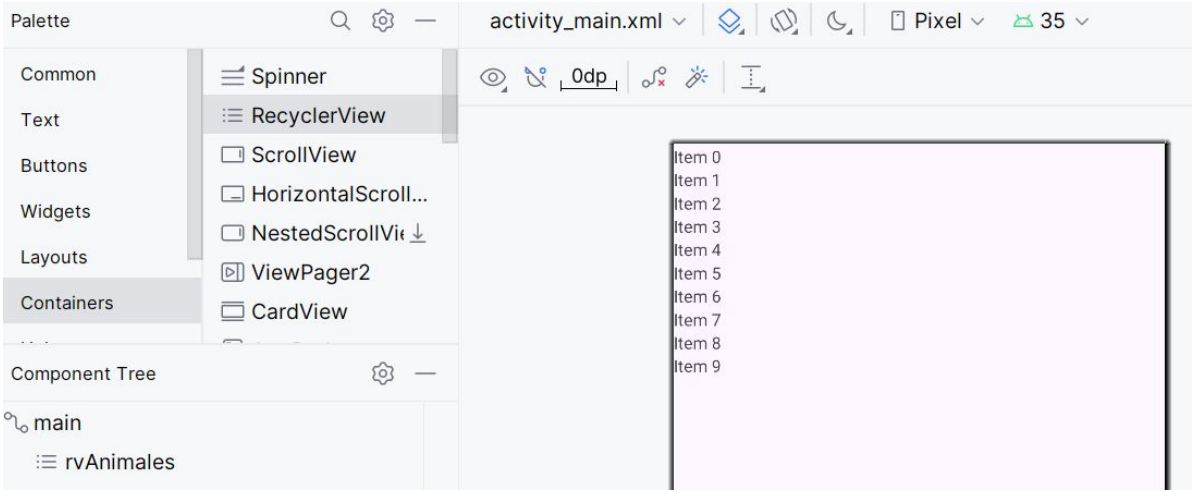
RecyclerView - Paso a paso

Construiremos paso a paso un RecyclerView que nos muestre una lista de animales:



RecyclerView - Paso 1: añadir el RV en la actividad

Añadimos el RV en el layout de nuestra actividad. Como siempre le damos tamaño (en mi caso match_parent ambas). En mi caso lo he renombrado como rvAnimales.



RecyclerView - Paso 2: Creamos el diseño de una fila

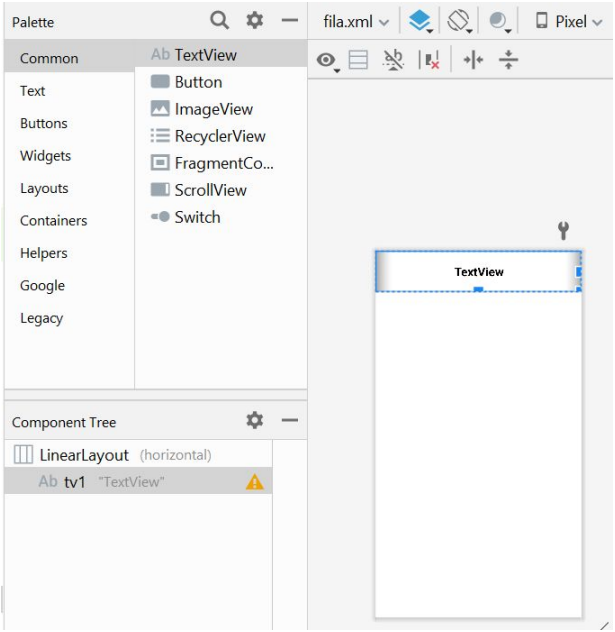
Dentro de la carpeta res/layout creamos un nuevo xml que será el diseño de cada fila (botón derecho New / Layout Resource File). En mi caso lo he llamado fila.xml.

Este layout que creamos puede ser de diversos tipos. Por simplicidad en cuanto a diseño he cogido LinearLayout.

Dentro he puesto únicamente un cuadro de texto. En este xml se ponen los controles que queramos presentar para cada fila.

A mi TextView le he dado tamaño de match_parent ambos. Al Linear Layout match parent de ancho y 80 dp de alto.

También he subido la fuente del texto y puesto gravity center del TextView en el linear layout.



RecyclerView - Paso 3: Creamos el ViewHolder

La clase RecyclerView.Adapter nos obligará a hacer uso del patrón view holder.

Por tanto, para poder seguir con la implementación del adaptador lo primero que definiremos será el ViewHolder necesario para nuestro caso de ejemplo.

Lo definiremos como un nueva **clase**, extendiendo de la clase RecyclerView.ViewHolder, y será bastante sencillo. Al extender de la clase RecyclerView.ViewHolder nos obliga a crear un constructor.

En este constructor tendremos que incluir el "casteo" a los controles de cada fila que con los que queramos trabajar en nuestro Recycler. Como "peculiaridad" debemos utlizar siempre el método findViewById() sobre la vista recibida como parámetro.

RecyclerView - Paso 3: Creamos el ViewHolder

```
class MyView(itemView: View) : RecyclerView.ViewHolder(itemView) {

    val tvAnimales: TextView = itemView.findViewById(R.id.tv1)

}
```

RecyclerView - Paso 4: Creamos el adaptador

Igual que hemos hecho con el Viewholder, crearemos una nueva **clase** que en este caso extenderá de RecyclerView.Adapter.

Al extender de esta clase nos obliga a implementar 3 funciones que son los que en realidad van construyendo el recycler.

Además de estos tres funciones, necesitaremos un constructor para cuando inicializamos el objeto pasar los datos necesarios en el Recycler.

RecyclerView - Paso 4: Creamos el adaptador

¿Cómo definir un adapter?

Un adapter:

- presenta un constructor y/o métodos para gestionar el conjunto de datos (añadir, editar o eliminar elementos),
- contiene la función onCreateViewHolder que infla el layout (archivo xml) que representa a nuestros elementos, y devuelve una instancia de la clase ViewHolder que antes definimos;
- contiene la función onBindViewHolder que enlaza nuestra dartos con cada ViewHolder.
- contiene la función getItemCount que devuelve un entero indicando la cantidad de elementos a mostrar en el RecyclerView.

Inflar. En Android se denomina así a la acción mediante la cual se agrega una jerarquía de vistas a la actividad en tiempo de ejecución.

RecyclerView - Paso 4: Creamos el adaptador

Extendemos de RecyclerView.Adapter y tipamos con el objeto MyView que acabamos de hacer

```
class MyAdapter (private val dataSet: Array<String>) : RecyclerView.Adapter<MyView>() {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyView {
        TODO( reason: "Not yet implemented")
    }

    override fun getItemCount(): Int {
        TODO( reason: "Not yet implemented")
    }

    override fun onBindViewHolder(holder: MyView, position: Int) {
        TODO( reason: "Not yet implemented")
    }
}
```

Definimos un constructor con los datos que necesitamos. En este caso es sólo un array de strings

Más adelante tendremos que cambiar el Array<String> por List<String>, así que mejor definirlo así de primeras

RecyclerView - Paso 4: Creamos el adaptador

La función onCreateViewHolder() nos limitaremos a inflar (construir) una vista a partir del layout correspondiente a los elementos de la lista (row), y crear y devolver un nuevo ViewHolder.

```
// Create new views (invoked by the layout manager)
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyView {
    // Create a new view, which defines the UI of the list item
    val view = LayoutInflater.from(parent.context)
        .inflate(R.layout.mi_fila, parent, attachToRoot: false)

    return MyView(view)
}
```

RecyclerView - Paso 4: Creamos el adaptador

En onBindViewHolder() tan sólo tendremos que recuperar el objeto Animal correspondiente a la posición recibida como parámetro y asignar sus datos sobre el ViewHolder también recibido como parámetro.

```
// Replace the contents of a view (invoked by the layout manager)
override fun onBindViewHolder(holder: MyView, position: Int) {
    // Get element from your dataset at this position and replace the
    // contents of the view with that element
    holder.tvAnimales.text = dataSet[position]
}
```

RecyclerView - Paso 4: Creamos el adaptador

En getItemCount() tan sólo retornaremos un entero con el tamaño del array.

```
// Return the size of your dataset (invoked by the layout manager)
override fun getItemCount() = dataSet.size
```

RecyclerView - Paso 5: Inicializar RV en actividad

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

Tan sólo tendremos que crear nuestro adaptador pasándole como parámetro la lista de datos y asignarlo al control RecyclerView a la propiedad Adapter.

Esto, aunque ahora no tenga excesivo sentido al ser una lista estática, lo vamos a hacer con el patrón MVVM de forma que los datos los asignaremos en el modelo y será desde el view, desde la actividad, donde “montaremos” la llamada al Recycler.

Así que mi primer paso va ser crear el modelo o estado donde se recuperarían los datos. En este caso, tenemos una lista fija de datos, así que tendremos una función que nos devuelva esta lista

RecyclerView - MainState

```
class MainState {
    fun devuelveArray(): List<String> {

        var animalNames = mutableListOf("Caballo", "Vaca", "Perro", "Gato", "León", "Tigre",
                            "Rinoceronte", "Elefante", "Águila", "Mariposa", "Serpiente", "Oso")

        return animalNames
    }
}
```

RecyclerView - MainViewModel

Acordémonos de meter las dependencias del ciclo de vida en el proyecto. Por lo demás el ViewModel es bastante sencillo. Yo lo he hecho con StateFlow, pero con LiveData sería casi igual

```
class MainViewModel: ViewModel() {
    private val _datos  = MutableStateFlow<List<String>>(emptyList())
    val datos: StateFlow<List<String>> get() = _datos.asStateFlow()
    val myEstado = MainState()

    fun devuelveArray(){
        viewModelScope.launch {
            var retornoDatos = myEstado.devuelveArray()
            _datos.value = retornoDatos
        }
    }
}
```

RecyclerView - View

A nivel de clase nos declaramos las variables para la vinculación de vistas, el ViewModel y una para el adaptador del Recycler

```
private lateinit var binding: ActivityMainBinding
private val myViewModel: MainViewModel by viewModels()
lateinit var myAdapter: MyAdapter
```

RecyclerView - View

A nivel de clase nos declaramos las variables para la vinculación de vistas, el ViewModel y una para el adaptador del Recycler.

```
private lateinit var binding: ActivityMainBinding
private val myViewModel: MainViewModel by viewModels()
lateinit var myAdapter: MyAdapter
```

RecyclerView - View

En el onCreate llamamos al método del ViewModel que devuelve el aaray (que no nos engañe el nombre, lo hará de forma asincrona) y le damos la forma que queremos al Recycler (lineal)

```
// En el onCreate llamamos al viewModel para cargar el array
myViewModel.devuelveArray();
// Asignamos la forma en que queremos ver el Recycler
val mLayout = LinearLayoutManager( context: this@MainActivity)
rvAnimales.layoutManager = mLayout
```

RecyclerView - View

Por último recuperamos los datos a cargar en el Recycler. Con estos datos, recordemos si solo hay un parámetro la convención es it, construimos el adaptador y lo asignamos al Recycler.

```
lifecycleScope.launch {
    repeatOnLifecycle(Lifecycle.State.STARTED){
        myViewModel.datos.collect{
            // lo que recibimos (recordemos si es un solo paramétro puede
            // llamarse it y es una lista de animales se lo asignamos al adaptador
            myAdapter = MyAdapter (it)
            //Y el adaptador se lo asignamos al recycler.
            rvAnimales.adapter = myAdapter
        }
    }
}
```

RecyclerView - Paso 5: Inicializar RV en actividad

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

Tan sólo tendremos que crear nuestro adaptador personalizado pasándole como parámetro la lista de datos y asignarlo al control RecyclerView mediante el método setAdapter.

Además, y esto es opcional, he añadido un divisor simple entre las filas.

Crear un ItemDecoration personalizado no es una tarea inmediata. Pero para el caso de los separadores de lista, Android ya nos proporciona de serie la clase DividerItemDecoration. Para utilizar este componente deberemos simplemente crear el objeto y asociarlo a nuestro RecyclerView mediante addItemDecoration() en nuestra actividad principal

RecyclerView - View

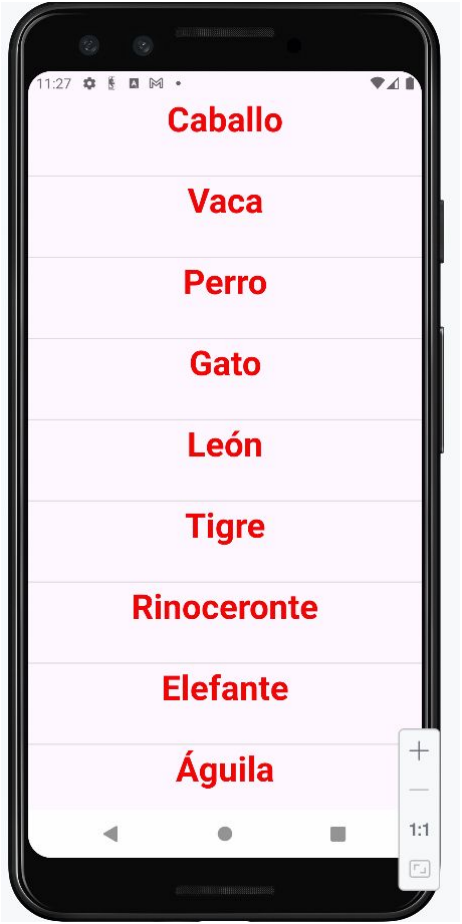
Ya tenemos la lista de animales que podemos recorrer.

Además, y esto es opcional, voy a añadir un divisor simple entre las filas.

Crear un ItemDecoration personalizado no es una tarea inmediata. Pero para el caso de los separadores de lista, Android ya nos proporciona de serie la clase DividerItemDecoration. Para utilizar este componente deberemos simplemente crear el objeto y asociarlo a nuestro RecyclerView mediante addItemDecoration() en nuestra actividad principal

RecyclerView - ItemDecoration

```
lifecycleScope.launch {
    repeatOnLifecycle(Lifecycle.State.STARTED){
        myViewModel.datos.collect{
            // lo que recibimos (recordemos si es un solo paramétro puede
            // llamarse it y es una lista de animales se lo asignamos al adaptador
            myAdapter = MyAdapter (it)
            //Y el adaptador se lo asignamos al recycler.
            rvAnimales.adapter = myAdapter
            // Metemos una división entre las filas
            val midividerItemDecoration =
                DividerItemDecoration(
                    rvAnimales.getContext(),
                    mLayout.orientation
                )
            rvAnimales.addItemDecoration(midividerItemDecoration)
        }
    }
}
```



RecyclerView - Paso 6: algunos comentarios

Tras obtener la referencia al RecyclerView podríamos haber incluido también una llamada a setHasFixedSize(). Aunque esto no es obligatorio, sí es conveniente hacerlo cuando tengamos certeza de que el tamaño de nuestro RecyclerView no va a variar (por ejemplo debido a cambios en el contenido del adaptador), ya que esto permitirá aplicar determinadas optimizaciones sobre el control.

- RecyclerView recyclerView = findViewById(R.id.rvAnimales);
- recyclerView.setHasFixedSize(true);

Adicionalmente, como último paso antes de asociar el adaptador al RecyclerView, le hemos creado y asignado un LayoutManager para determinar la forma en la que se distribuirán los datos en pantalla. Nuestra intención es mostrar los datos en forma de lista o tabla y no hemos tenido que implementar nuestro propio LayoutManager (una tarea nada sencilla), ya que el SDK proporciona varias clases predefinidas para los tipos más habituales. En nuestro caso particular queremos mostrar los datos en forma de lista con desplazamiento vertical. Para ello tenemos disponible la clase LinearLayoutManager, por lo que tan sólo hemos tenido que instanciar un objeto de dicha clase.

RecyclerView - Paso 6: algunos comentarios

Si cambiáramos de idea y quisiéramos mostrar los datos de forma tabular tan sólo tendríamos que cambiar la asignación del LayoutManager anterior y utilizar un GridLayoutManager, al que pasaremos como parámetro el número de columnas a mostrar.

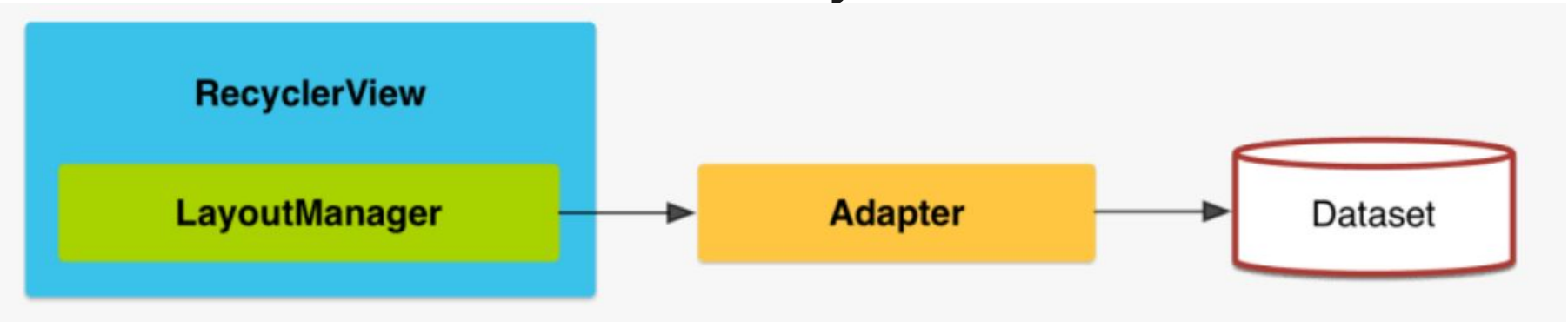
```
// Asignamos la forma en que queremos ver el Recycler
val mLayout = GridLayoutManager( context: this@MainActivity,
                                 spanCount: 2)
```

Controles de selección: RecyclerView - Resumen

RecyclerView por sí mismo no proporciona mucho más que este reciclado y un listener para seguir el scroll. De hecho, ni siquiera posiciona los elementos en pantalla o gestiona eventos que no sean el scroll y delega en otras clases la realización de algunas funcionalidades.

Esta simplicidad y modularidad posibilita la configuración y personalización de RecyclerView y lo convierten en un poderoso componente porque permite construir cualquier interfaz gráfica en la que se requiera mostrar un listado.

Controles de selección: RecyclerView - Resumen



Si comprendemos estos conceptos. Lo hemos comprendido todo sobre RecyclerViews.

Veamos uno por uno:

Controles de selección: RecyclerView - Resumen

- RecyclerView: Nuestro RecyclerView se va a "pintar" en función al LayoutManager que reciba como parámetro. También hará uso de un Adapter, que funcionará de acuerdo a un Dataset.
- LayoutManager: Este "gestor del diseño" va a definir la disposición de los elementos. Es decir, si van formando una lista vertical u horizontal, si van formando una cuadrícula, u otra variante.
- ViewHolder: simplemente castearemos los controles sobre los queremos trabajar.
- Adapter: El adaptador se encargará de adaptar el dataset a lo que finalmente verá el usuario. Es el encargado de traducir datos en UI. También de gestionar los eventos sobre el RecyclerView.
- Dataset: Es el conjunto de datos que se espera mostrar en el RecyclerView. Se puede representar por un simple array de objetos String; o ser algo más elaborado, como un ArrayList de objetos que presentan múltiples atributos.

https://developer.android.com/guide/topics/ui/layout/recyclerview