IES MONTE NARANCO

APUNTES DE CLASE

PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES

RecyclerView

Recycler view

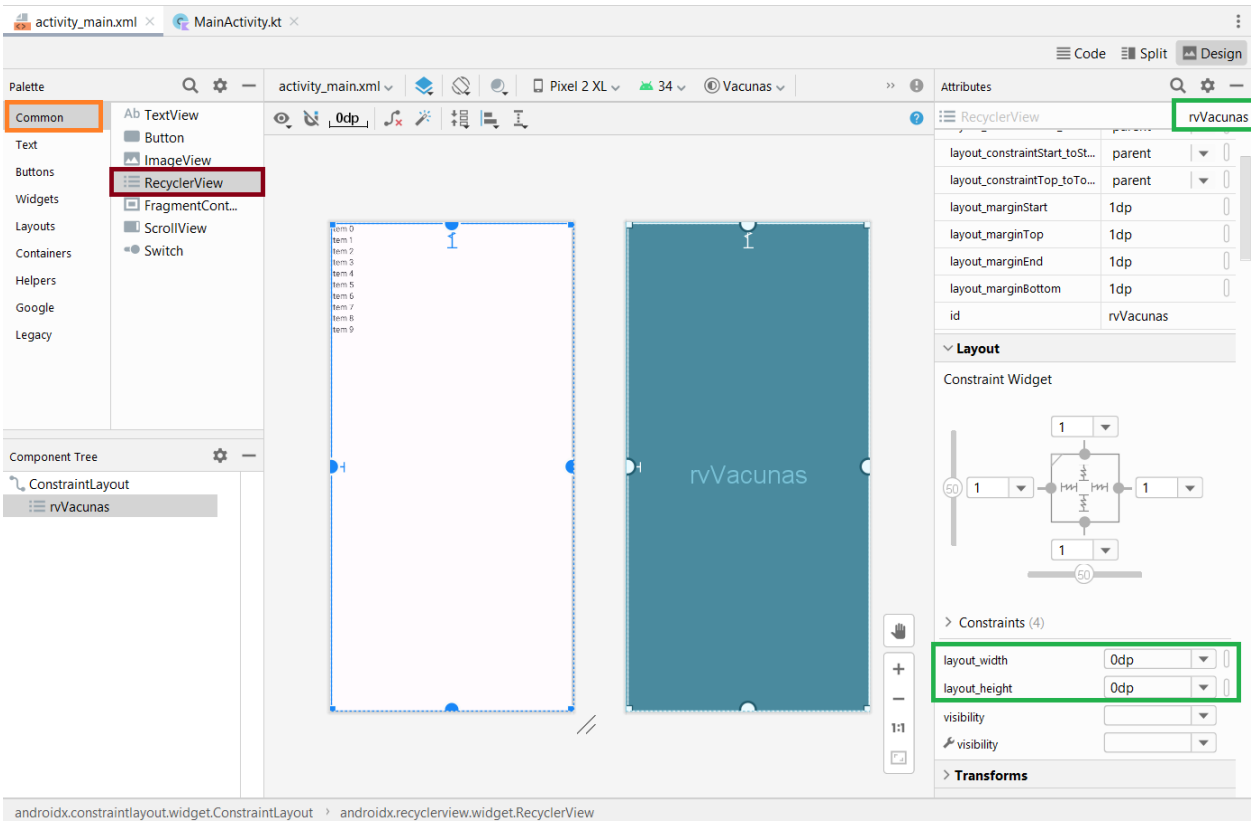
Recycler view mejora la eficiencia al manejar grandes conjuntos de datos, **reciclando** los items que se muestran en pantalla a medida que se desplazan y dejan de verse. El Recycler View, no destruye los componentes, sino que los nuevos elementos reutilizan los componentes que se desplazaron. Esto mejora el rendimiento, la velocidad de respuesta de la app y el consumo de energía.

Ejemplo

Tras crear un proyecto nuevo el primer paso será definir el layout de la app.

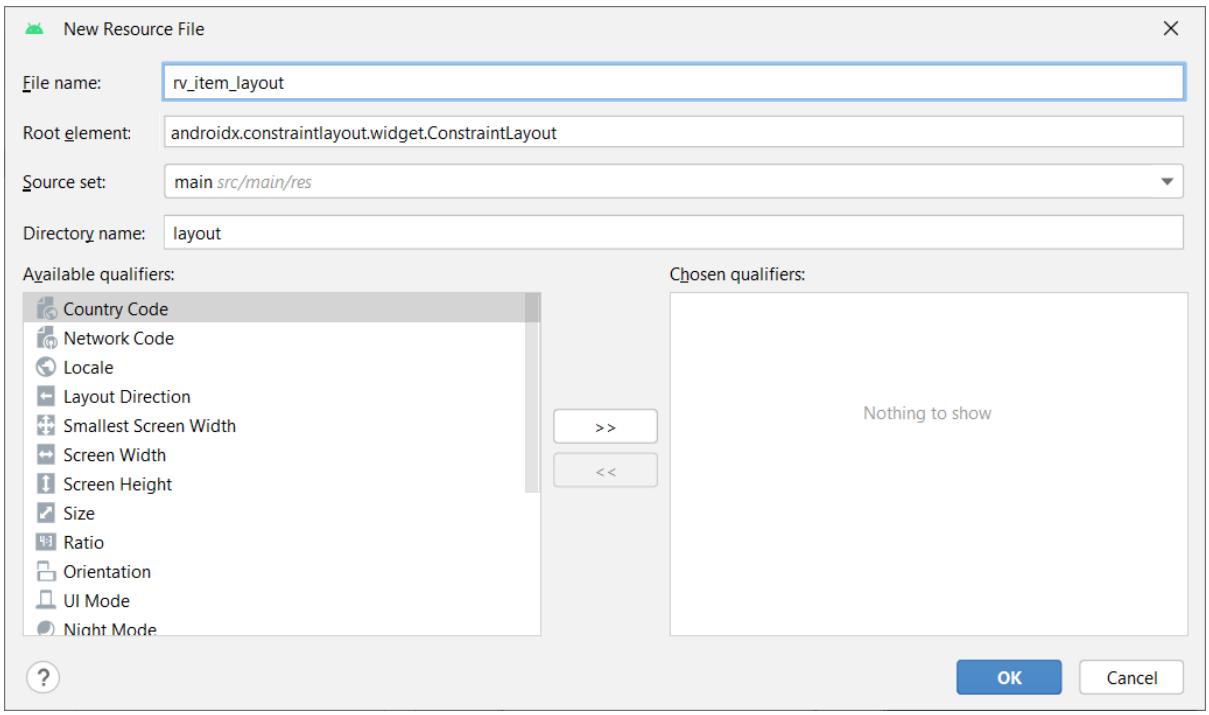
Activity layout

Arrastraremos un componente Recycler View desde la paleta de componentes, inferiremos sus constraints y ajustemos sus dimensiones a 0dp (ajustar a las constraints), dejando de margen sólo 1 dp. Le llamaremos rvVacunas



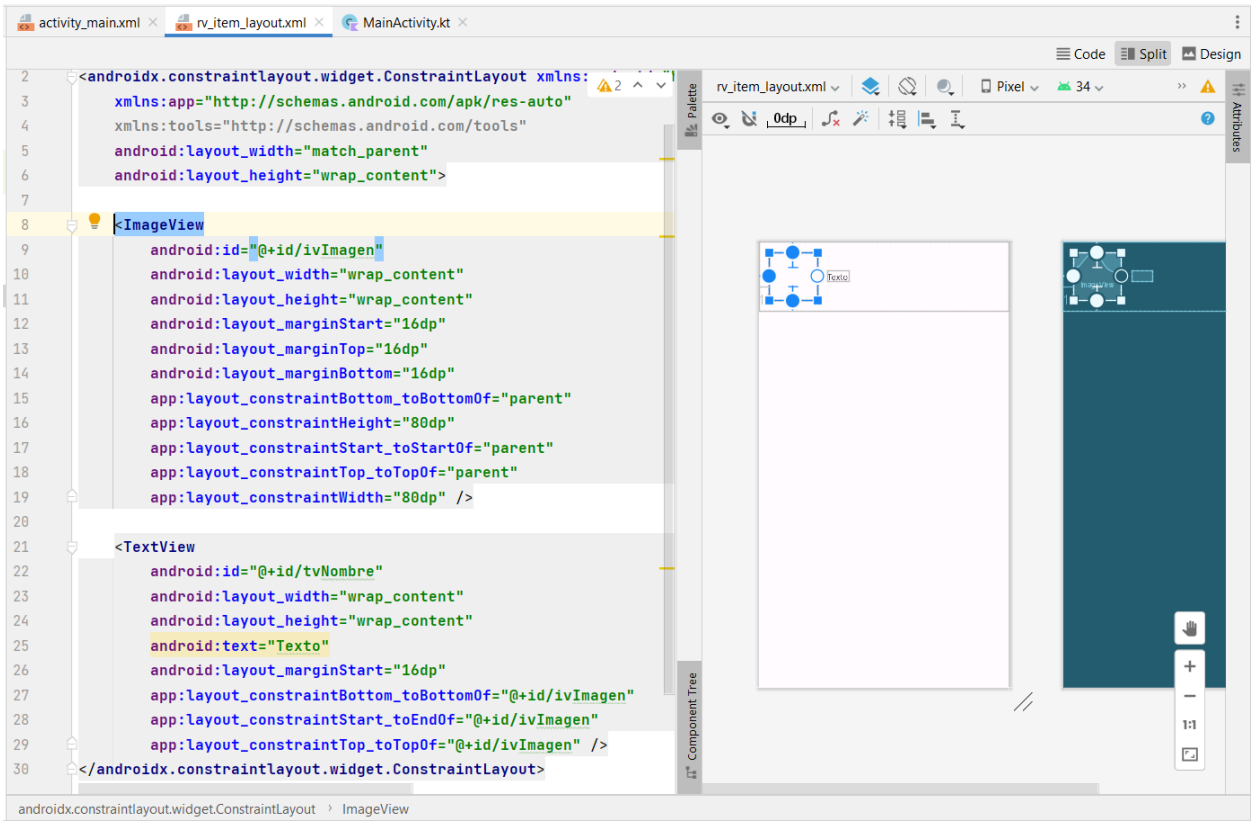
Items layout

El segundo paso es crear el layout para los items que se mostrarán en el Recycler View. En la carpeta res/layout creamos un layout nuevo, que llamaremos rv_item_layout



En este layout vamos a definir cómo será cada uno de los elementos que mostrará el Recycler View. Una vez definido el ítem, configuraremos el layout para que su largo sea el del ítem, es decir el de su contenido.

Incluiremos una imagen y un texto que llamaremos respectivamente ivImagen y tvNombre.



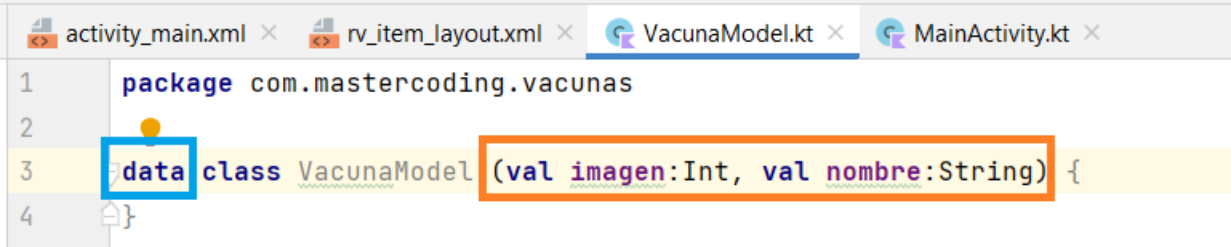
La imagen tendrá constraints arriba y a la izquierda con un margen de 16dp. Una vez establecido que el layout se ajuste al contenido (layout_height="wrap_content"), se ajusta que la imagen tenga también como constraint la parte inferior del layout con un margen de 16dp. El texto tiene como constraints verticales la parte superior e inferior de la imagen y horizontales el fin de la imagen con un margen de 16dp.

Model class

A continuación, se definen la clase que contendrá los datos a mostrar. Un array de estos objetos conformará los datos a mostrar.

Crearemos una nueva clase Kotlin llamada VacunaModel e indicaremos que es de tipo data class, de forma que Kotlin nos facilite el trabajar con datos como hemos visto en el apartado de data class, con métodos como la comparación o la copia ya implementada.

Esta clase tendrá una propiedad para almacenar la imagen (que al ser un recurso drawable se referenciará por su Id que es un entero) y una propiedad para almacenar el nombre. Las haremos formar parte del constructor primario para poder utilizar los métodos pre-implementados en las data class.



Adapter

El adaptador, será el encargado de proporcionar los items a mostrar al componente Recycler View de nuestro Layout. En el arranque de la aplicación, le diremos al Recycler View rvVacunas, utilice como Adaptador el que vamos a diseñar, de forma que cuando el usuario se mueva por la lista, sea dicho adaptador el que se encargue de crear los items que deben mostrarse en cada momento.

La clase que actúa como Adaptador, tiene como propiedades la lista de items a mostrar, que le pasaremos antes de asignarlo al Recycler View durante el arranque de la aplicación, y que contendrá los datos a mostrar en el Recycler View.

Es decir, una vez creado el adaptador con la lista de items a mostrar y asignado este al Recycler View, el widget ya será autónomo, mostrando los items con los datos del array, y reutilizando sus items a medida que el usuario se desplaza por el componente.

Este adaptador, en lugar de heredar del Base Adapter, heredará de RecyclerView.Adapter. Un Adapter más sencillo, creado a medida para los Recycler View.

La clase heredada necesita conocer en su tipo genérico el View Holder, así que paso fundamental para continuar, definir el View Holder.

View Holder

activity_main.xml | rv_item_layout.xml | VacunaModel.kt | VacunasAdapter.kt | MainActivity.kt

```
1  package com.mastercoding.vacunas
2  import androidx.recyclerview.widget.RecyclerView
3
4  class VacunasAdapter(val listaVacunas:ArrayList<VacunaModel>)
5      : RecyclerView.Adapter<VacunaViewHolder>
6  {
7  }
```

Para continuar debemos definir la clase View Holder; la clase que permitirá acceder a los widgets del componente que muestra el Recycler View, y para el que hemos creado un layout custom. El layout del componente, se "inflará", es decir, se creará en tiempo de ejecución y se pasará al View Holder, para que este pueda inicializar sus componentes con los del widget creado a partir del layout. Y a diferencia del ListView, sólo tendremos que enviarle al Recycler View el View Holder, pues la relación entre View Holder y View Group (el ítem) va ahora en el View Holder (la clase ha heredado de RecyclerView.ViewHolder que se encarga de guardar dicha relación) y no en la propiedad tag del ViewGroup.

activity_main.xml | rv_item_layout.xml | VacunaModel.kt | VacunasAdapter.kt | MainActivity.kt

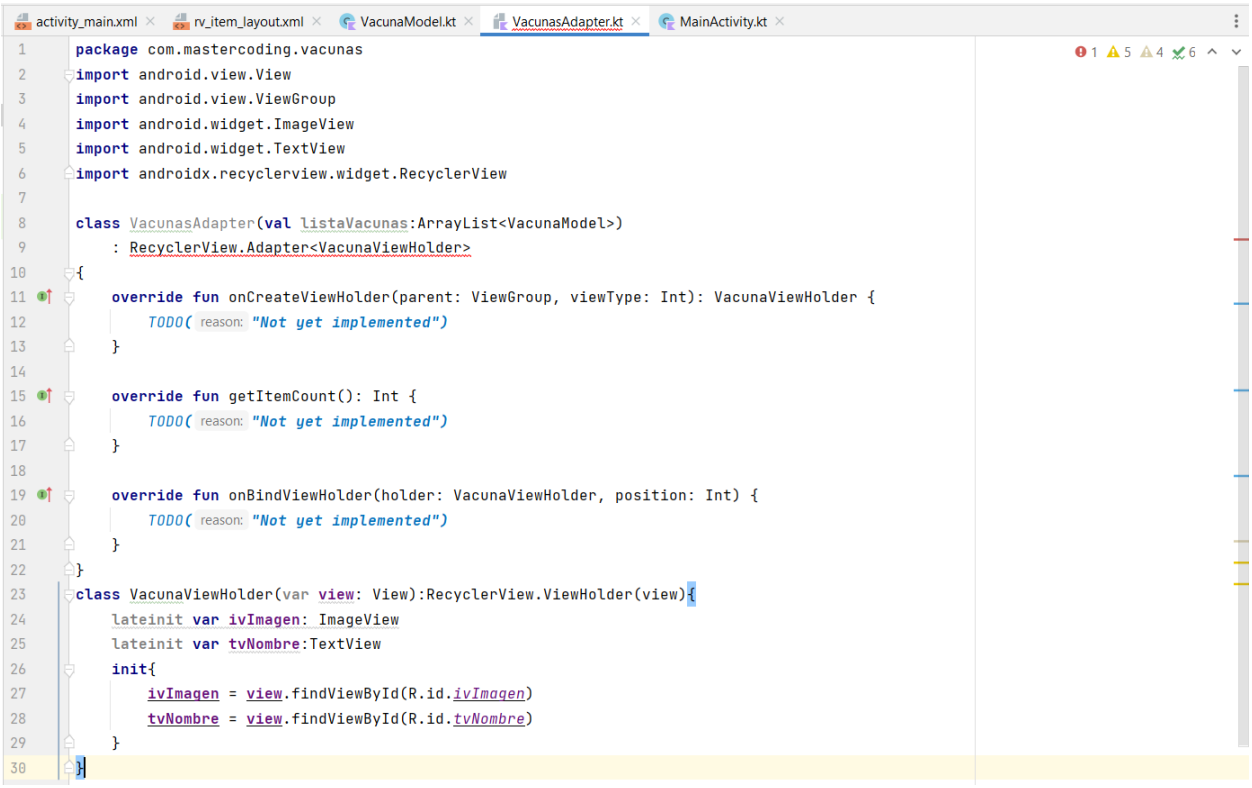
```
1  package com.mastercoding.vacunas
2  import android.view.View
3  import android.widget.ImageView
4  import android.widget.TextView
5  import androidx.recyclerview.widget.RecyclerView
6
7  class VacunasAdapter(val listaVacunas:ArrayList<VacunaModel>)
8      : RecyclerView.Adapter<VacunaViewHolder>
9  {
10 }
11 class VacunaViewHolder(var view: View):RecyclerView.ViewHolder(view){
12     lateinit var ivImagen: ImageView
13     lateinit var tvNombre:TextView
14     init{
15         ivImagen = view.findViewById(R.id.ivImagen)
16         tvNombre = view.findViewById(R.id.tvNombre)
17     }
18 }
```

Es habitual crear el View Holder como una *inner class* del Adaptador, aunque no es necesario.

Métodos del Adaptador

La clase heredada, RecyclerView.Adapter sólo requiere la implementación de tres métodos en lugar de los cuatro que requeriría el BaseAdapter.

Con botón derecho sobre el error, pulsamos implementar métodos.



onCreateViewHolder

El Recycler View ejecuta este método cuando necesita un nuevo View Holder, es decir, un nuevo componente, pues ahora View Holder y View Group viajan en el View Holder. En este método se "inflará el layout", es decir se construirá un ítem nuevo desde un layout XML, por el procedimiento de inflado, o creación dinámica de un View Group con un layout. Ese View Group, lo recibirá el View Holder, que mapeará en sus propiedades los widgets que forman el ViewGroup. Y lo devolverá al Recycler View, para que sea un nuevo ítem. Pero sin datos, este método sólo crea ítems nuevos. El número de veces que se llama suele coincidir con los items que se pueden ver simultáneamente en pantalla.

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): VacunaViewHolder {
    var itemLayout = LayoutInflater.from(parent.context)
                    .inflate(R.layout.rv_item_layout, parent, attachToRoot: false)
    return VacunaViewHolder(itemLayout)
}
```

onBindViewHolder

Este método es ejecutado por el Recycler View para que se carguen los datos a mostrar en una determinada posición de la lista, usando el View Holder que nos pasa. En este método tenemos que rellenar los datos a mostrar, en las propiedades del view Holder que recibimos, con los datos que están la posición de la lista, que nos indica el argumento position.

Para ello, con la lista de objetos que tiene el adaptador, que es donde tenemos los datos a mostrar, cada uno en un objeto VacunaModel, y que le hemos pasado al inicio de la app al adaptador, cargamos el View Holder, accediendo al objeto en la posición indicada en la lista.

```
override fun onBindViewHolder(holder: VacunaViewHolder, position: Int) {
    holder.ivImagen.setImageResource(listaVacunas[position].imagen)
    holder.tvNombre.text = listaVacunas[position].nombre
}
```

getItemCount

Este es el método más sencillo de programar, pues hay que devolverle el tamaño de la lista de objetos VacunaModel, que le hemos pasado al adaptador en el arranque.

Con estos métodos hemos finalizado el Adaptador, que tiene un método que creará nuevos elementos, uno que los actualizará con los datos de una determinada posición y uno que le indica cuantos items tiene la lista, quedando como sigue:

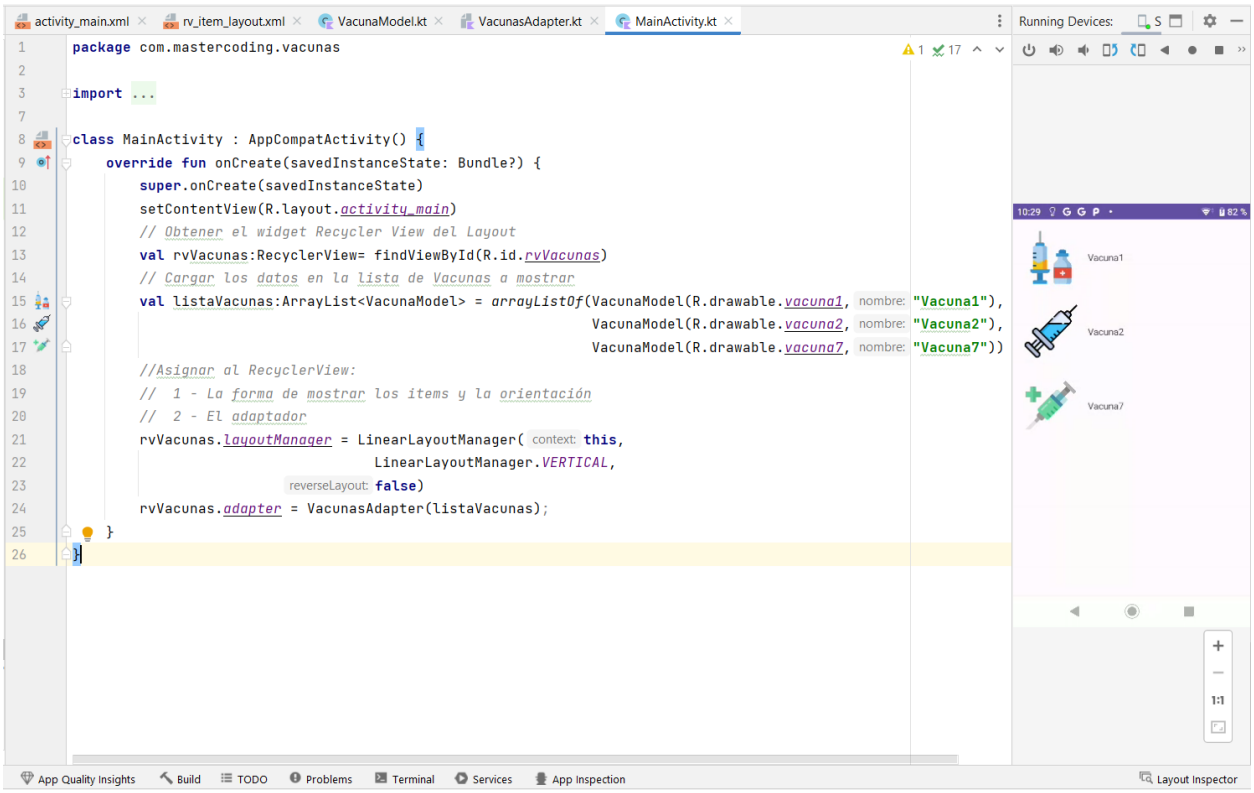
```
class VacunasAdapter(val listaVacunas:ArrayList<VacunaModel>)
    : RecyclerView.Adapter<VacunaViewHolder>()
{
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): VacunaViewHolder {
        var itemLayout = LayoutInflater.from(parent.context)
                        .inflate(R.layout.rv_item_layout, parent, attachToRoot: false)
        return VacunaViewHolder(itemLayout)
    }

    override fun getItemCount(): Int {
        return listaVacunas.size
    }

    override fun onBindViewHolder(holder: VacunaViewHolder, position: Int) {
        holder.ivImagen.setImageResource(listaVacunas[position].imagen)
        holder.tvNombre.text = listaVacunas[position].nombre
    }
}

class VacunaViewHolder(var view: View):RecyclerView.ViewHolder(view){
    lateinit var ivImagen: ImageView
    lateinit var tvNombre:TextView
    init{
        ivImagen = view.findViewById(R.id.ivImagen)
        tvNombre = view.findViewById(R.id.tvNombre)
    }
}
```

Inicializar la Recycler View

Antes de nada se deben copiar a res/drawable los iconos con las vacunas a mostrar.

A continuación se obtiene el widget Recycler View del layout de la aplicación y se inicializan los datos a mostrar en una lista de objetos VacunaModel.

Por último se asigna al widget del layout, que es del tipo Recycler List, el adaptador indicándole que queremos mostrar los items usando un layout lineal vertical.



Selección de un ítem

Para gestionar la pulsación de uno de los items, tenemos que programar el evento setOnClickListener en el View Holder, en el momento que recibe el ViewGroup. De esta forma el ViewGroup que se almacena con el View Holder, lleva incluido el evento de pulsación de botón.

```
class VacunaViewHolder(var view: View):RecyclerView.ViewHolder(view){
    lateinit var ivImagen: ImageView
    lateinit var tvNombre:TextView
    init{
        ivImagen = view.findViewById(R.id.ivImagen)
        tvNombre = view.findViewById(R.id.tvNombre)
        view.setOnClickListener(){ it: View!
            Toast.makeText(view.context, text: "${tvNombre.text.toString()}",Toast.LENGTH_LONG).show();
        }
    }
}
```