



Programación multimedia y dispositivos móviles

UT-4. Listas. RecyclerView - 2

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



Controles de selección: RecyclerView

Una vez que tenemos el RecyclerView hecho y podemos verlo, el siguiente paso sería implementar funcionalidad en alguno de sus eventos. Lo más habitual será necesitar implementar funcionalidad en el evento click sobre un elemento de la lista.

Hasta ahora, cuando hemos querido capturar un evento de alguno de los controles que hemos introducido, en la clase asociada al propio control se podía controlar sus eventos.

Pero la clase RecyclerView no incluye un evento `onItemClickListener`. Una vez más, RecyclerView delega también esta tarea a otro componente, en este caso a la propia vista que conforma cada elemento de la colección.

Será por tanto dicha vista a la que tendremos que asociar el código a ejecutar como respuesta al evento click.



RecyclerView - Recordamos

Dentro del adaptador tenemos implementado:

- ❑ contiene una clase interna **ViewHolder**, que permite obtener referencias de los componentes visuales (views) de cada elemento de la lista,
- ❑ presenta un **constructor** y/o métodos para gestionar los datos (añadir, editar o eliminar elementos),
- ❑ contiene un método **onCreateViewHolder** que infla el layout (archivo xml) que representa a nuestros elementos, y devuelve una instancia de la clase ViewHolder que antes definimos;
- ❑ contiene un método **onBindViewHolder** que enlaza nuestra data con cada ViewHolder.
- ❑ contiene un método **getItemCount** que devuelve un entero indicando la cantidad de elementos a mostrar en el RecyclerView.



Controles de selección: RecyclerView

Sobre el proyecto de la lista de animales vamos a programar la lógica de dos eventos:

- ❏ Con el evento Click vamos a poner el fondo del cuadro en rojo y texto en blanco.
- ❏ Con el evento LongClick vamos a devolverlo a fondo blanco y texto en negro.

La programación de estos eventos se puede hacer, al menos, de 2 formas. La más sencilla es aprovechar el método `onBindViewHolder()` para asignar a su vista asociada el evento `onClick`.

Otra alternativa es manejar los eventos de clics dentro del `ViewHolder`. No lo vamos a hacer en clase, pero al final de la presentación tenéis la explicación paso a paso de cómo hacerlo.

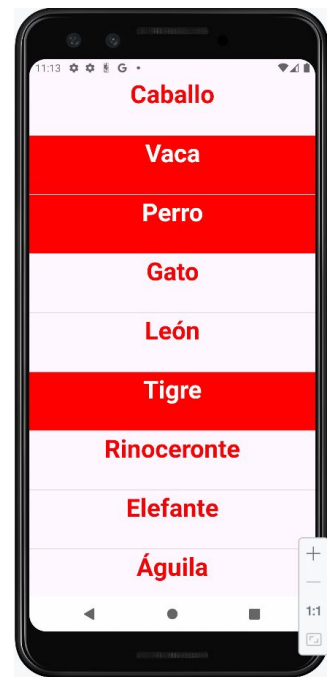


RecyclerView - Eventos en onBindViewHolder

En este adaptador, dentro del método onBindViewHolder que ya habíamos sobrescrito para poder ver el RecyclerView, dentro del holder, para cada Item capturo el evento de Click y LongClick y sobrescribo los métodos con la lógica que queremos.

RecyclerView - Eventos en onBindViewHolder

```
// 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]
    holder.tvAnimales.setOnClickListener{
        holder.tvAnimales.setTextColor(Color.WHITE);
        holder.tvAnimales.setBackgroundColor(Color.RED)
    }
    holder.tvAnimales.setOnLongClickListener {
        holder.tvAnimales.setTextColor(Color.RED);
        holder.tvAnimales.setBackgroundColor(Color.WHITE)
        true
    }
}
```



RecyclerView - Eventos en onBindViewHolder

```
private var selectedPos = RecyclerView.NO_POSITION
```

Antes de seguir vamos a codificar que cuando se pulse un registro, este se pondrá en rojo esté, pero se refrescará los anteriores dejándolos como estaban.

Al construir la fila ponemos lógica para que si la posición es igual a la posición seleccionada (variable que me creó previamente a nivel de clase) se pinte de un color u otro

Luego en el onclick le decimos al recycler que ha cambiado primero la posición seleccionada (que al no estar actualizada tiene la posición anterior por lo que no coincide con la variable de position y la pinta normal, asignamos la variable de fila seleccionada y volvemos a decirle que esta posición ha cambiado, pero ahora si es la que hemos clickeado

```
// 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]
    if (selectedPos == position){
        holder.tvAnimales.setTextColor(Color.WHITE);
        holder.tvAnimales.setBackgroundColor(Color.RED)
    } else{
        holder.tvAnimales.setTextColor(Color.RED);
        holder.tvAnimales.setBackgroundColor(Color.WHITE)
    }
    holder.tvAnimales.setOnClickListener{
        notifyItemChanged(selectedPos);
        selectedPos = position;
        notifyItemChanged(selectedPos);
    }
}
```



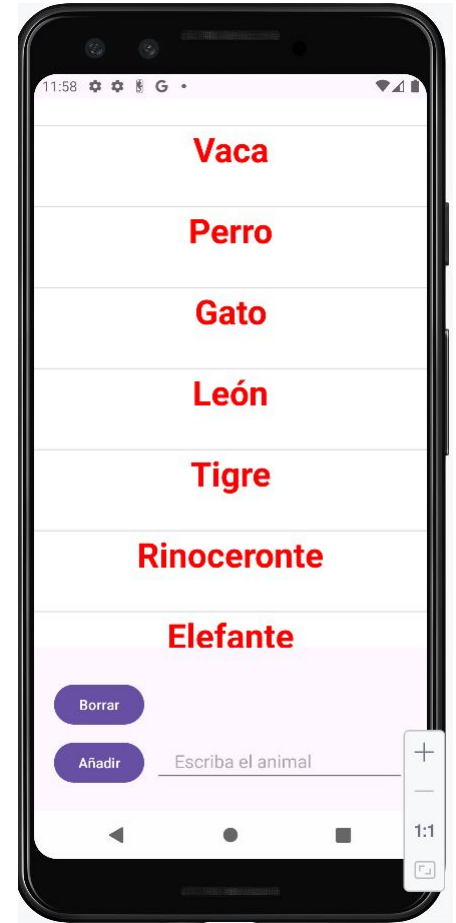
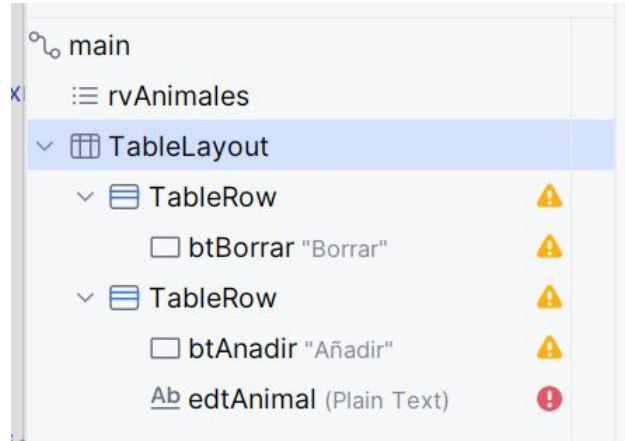
Controles de selección: RecyclerView

Ahora que ya tenemos el RecyclerView y tratamos 2 de sus eventos de click, vamos a acabar con este control viendo un par de funcionalidades más:

- ❏ Cómo implementar refrescos en la lista de datos
- ❏ Incorporar animación en estos refrescos

RecyclerView - Refrescando datos

Rediseñamos un poco la activity sobre la que estamos trabajando para incorporar 2 botones, uno para borrar registros y otro para insertarlos, poniendo además un texto para que el usuario elija el animal a introducir.



RecyclerView - Refrescando datos

Al hacer la lógica de refrescar datos, con los objetos de tipo `MutableStateFlow` no acababa de funcionar bien, hacía muchas cosas raras según tocabas el `RecyclerView`.

He probado con objetos `LiveData` y ha funcionado mucho mejor. ¿Porqué? Ni idea






RecyclerView - Refrescando datos - Datos

Para refrescar datos, tanto al eliminar como al añadir, necesitaremos 2 métodos. Estos deben devolver a la actividad tanto la posición eliminada o añadida como el nuevo array. Nos generamos una Data Class que contenga estos datos.

```
package com.example.myfirstrecyclerview
```

```
data class Datos(var animalNames: MutableList<String>, var posicion: Int)
```



Debe ser Mutable porque el array va ir cambiando

RecyclerView - Refrescando datos - MainState

Añadimos la lógica de borrar uno de los registros (debe ser el seleccionado por lo que debe llegarnos la posición que está seleccionada).

Inicializamos un objeto de tipo datos con los animales y posición a 0

```
class MainState {  
  
    var datos = Datos(mutableListOf("Caballo", "Vaca", "Perro", "Gato", "León", "Tigre",  
        "Rinoceronte", "Elefante", "Águila", "Mariposa", "Serpiente", "Oso"), posicion: 0)  
  
    fun borrar(posicion: Int): Datos {  
        datos.posicion = posicion  
        datos.animalNames.removeAt(posicion)  
        return datos  
    }  
}
```



RecyclerView - Refrescando datos - MainState

Añadimos la lógica de añadir uno de los registros.

```
fun anyadir( posicion: Int, animal : String): Datos {  
    datos.animalNames.add(posicion,animal)  
    return datos  
}
```



RecyclerView - Refrescando datos - MainModelView

Como digo he cambiado la lógica a LiveData porque con StateFlow no conseguí que funcionara de forma estable

```
class MainViewModel: ViewModel() {  
  
    val myEstado = MainState()  
    private val _datos = MutableLiveData<List<String>>(emptyList())  
    private val _borrado = MutableLiveData<Datos>(Datos(ArrayList(), posicion: 0))  
    private val _anyadido = MutableLiveData<Datos>(Datos(ArrayList(), posicion: 0))  
    val datos: LiveData<List<String>> get() = _datos  
    val borrado: LiveData<Datos> get() = _borrado  
    val anyadido: LiveData<Datos> get() = _anyadido  
}
```



RecyclerView - Refrescando datos - MainModelView

```
fun devolverArray(){  
    viewModelScope.launch {  
        _datos.value = myEstado.devolverArray()  
    }  
}  
  
fun borrar(posicion: Int){  
    viewModelScope.launch {  
        _borrado.value = myEstado.borrar(posicion)  
    }  
}  
  
fun anyadir(posicion: Int, animal: String){  
    viewModelScope.launch {  
        _borrado.value = myEstado.anyadir(posicion, animal)  
    }  
}
```



RecyclerView - Refrescando datos - Activity

Aquí tenemos 2 partes. La primera son los eventos que simplemente validarán que hay datos (posición para borrar y nombre de animal para añadir) y llamaran a los métodos del viewModel.

```
btBorrar.setOnClickListener{  
    val posicion = myAdapter.selectedPos  
  
    if (posicion < 0) {  
        Toast.makeText(applicationContext, text: "Debe Seleccionar un animal para borrar",  
            Toast.LENGTH_SHORT).show()  
        return@setOnClickListener  
    }  
    myViewModel.borrar(posicion)  
}
```




RecyclerView - Refrescando datos - Activity

```
btAnadir.setOnClickListener {  
    var posicion = myAdapter.selectedPos  
    if (posicion < 0) {  
        posicion = 0  
    } else {  
        posicion++  
    }  
    val animal = edtAnimal.text.toString()  
    if (animal == "") {  
        Toast.makeText(applicationContext, text: "Debe introducir un animal", Toast.LENGTH_SHORT).show()  
        return@setOnClickListener  
    }  
    myViewModel.anyadir(posicion, animal)  
}
```



RecyclerView - Refrescando datos - Activity

Y ahora ponemos los observadores para cada uno de los casos:

```
myViewModel.datos.observe( owner: this@MainActivity){  
    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 - Refrescando datos - Activity

```
myViewModel.borrado.observe( owner: this@MainActivity){  
    myAdapter.notifyItemRemoved(it.posicion)  
    myAdapter.selectedPos = RecyclerView.NO_POSITION  
    myAdapter.notifyItemRangeChanged( positionStart: 0, it.animalNames.size)  
}  
  
myViewModel.anyadido.observe( owner: this@MainActivity){  
    myAdapter.notifyItemInserted(it.posicion);  
    myAdapter.notifyItemRangeChanged( positionStart: 0, it.animalNames.size);  
}
```

De dónde saca el Recycler el nombre del animal añadido?? Por referencia??



RecyclerView - Refrescando datos

```
public void borrar(View v){
    int posicion = adapter.getSelectedPos();
    Context context = getApplicationContext();
    int duration = Toast.LENGTH_SHORT;

    if (posicion<0){
        String text = "Debe Seleccionar un animal para borrar";
        Toast toast = Toast.makeText(context, text, duration);
        toast.show();
        return;
    }
    animalNames.remove(posicion);
    adapter.notifyItemRemoved(posicion);
    adapter.setSelectedPos();
    adapter.notifyItemRangeChanged( positionStart: 0, animalNames.size());
}
```



RecyclerView - Animación por defecto

Para acabar hablaremos un poco del `ItemAnimator`. Un componente `ItemAnimator` nos permitirá definir las animaciones que mostrará nuestro `RecyclerView` al realizar las acciones más comunes sobre un elemento (añadir, eliminar, mover, modificar).

Este componente tampoco es sencillo de implementar, pero el SDK también proporciona una implementación por defecto que puede servir en la mayoría de ocasiones. Esta implementación por defecto se llama `DefaultItemAnimator` y podemos asignarla al `RecyclerView` mediante su propiedad `itemAnimator`.



RecyclerView - Animación por defecto

```
rvAnimales.addItemDecoration(midividerItemDecoration)  
rvAnimales.itemAnimator = DefaultItemAnimator()  
rvAnimales.animate()
```

Lo cierto es que la implementación es muy sencilla, pero al menos en mi caso no he conseguido que funcione con la lista de animales.



RecyclerView - Animación personalizada

Si he conseguido hacer funcionar una animación personalizada que he encontrado en esta dirección y cuya implementación es bastante sencilla:

<https://gastack.mx/programming/26724964/how-to-animate-recyclerview-items-when-they-appear>



RecyclerView - Animación personalizada - Paso 1

Dentro de la carpeta res creamos otra carpeta llamada anim para tener la ruta res / anim. Dentro genero 2 xmls:

res / anim / layout_animation.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<layoutAnimation xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:animation="@anim/item_animation_fall_down"
```

```
    android:animationOrder="normal"
```

```
    android:delay="15%" />
```




RecyclerView - Animación personalizada - Paso 1

res / anim / item_animation_fall_down.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="500">

    <translate
        android:fromYDelta="-20%"
        android:toYDelta="0"
        android:interpolator="@android:anim/decelerate_interpolator"
    />

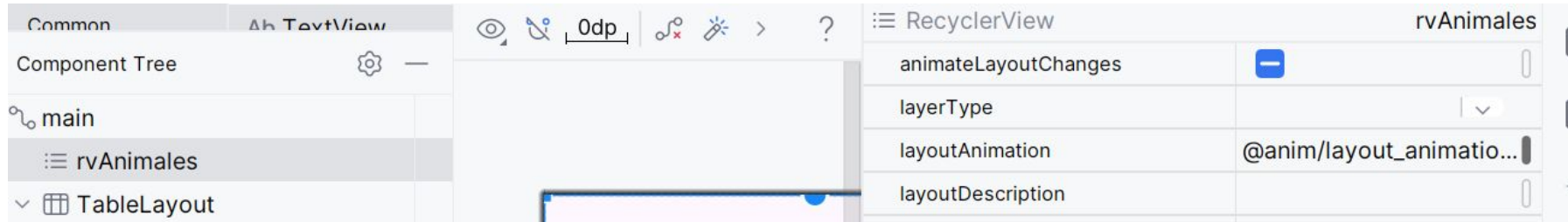
    <alpha
        android:fromAlpha="0"
        android:toAlpha="1"
        android:interpolator="@android:anim/decelerate_interpolator"
    />

    <scale
        android:fromXScale="105%"
        android:fromYScale="105%"
        android:toXScale="100%"
        android:toYScale="100%"
        android:pivotX="50%"
        android:pivotY="50%"
        android:interpolator="@android:anim/decelerate_interpolator"
    />

</set>
```

RecyclerView - Animación personalizada - Paso 2

Dentro RecyclerView añadimos el atributo de layoutanimacion





RecyclerView - Animación personalizada - Paso 3

Dentro de los observadores que refrescan el RecyclerView incluimos una llamada al evento `scheduleLayoutAnimation()` para que utilice la animación cada vez que la lista cambie.

```
myViewModel.borrado.observe( owner: this@MainActivity){  
    myAdapter.notifyItemRemoved(it.posicion)  
    myAdapter.selectedPos = RecyclerView.NO_POSITION  
    myAdapter.notifyItemRangeChanged( positionStart: 0, it.animalNames.size)  
    rvAnimales.scheduleLayoutAnimation()  
}
```



RecyclerView - Documentación utilizada

Para el RecyclerView os dejo algunas de las páginas que he utilizado o que no he utilizado pero me parece que pueden ser muy útiles sobre RecyclerView, animaciones, decoraciones,.. (creo que son todas las que he consultado, pero no estoy seguro. No incluyo las que ya he añadido previamente) :

<https://danielme.com/2015/08/15/android-recycler-view-listas/>

<https://www.it-swarm-es.com/es/android/ejemplo-simple-de-android-recyclerview/829425156/>

<https://programmerclick.com/article/3799300450/>

<https://stevengarcia83.wordpress.com/2015/11/18/itemdecoration-itemanimator/>

<https://programacionymas.com/blog/listas-dinamicas-android-usando-recycler-view-card-view>

<https://developer.android.com/guide/topics/resources/animation-resource>



Anexo: RecyclerView - Eventos en ViewHolder - Paso 1

Dentro de MiAdaptador defino una interfaz en la que a su vez defino métodos para los eventos que queremos capturar.

```
public class MiAdaptador extends RecyclerView.Adapter<MiAdaptador.ViewHolder>
{
```

```
    private List<String> mData;
    private LayoutInflater mInflater;
    private myItemClickListener mClickListener;
```

```
    // Creamos una interfaz con métodos para los eventos a tratar
    public interface myItemClickListener {
        void onItemClick(View view);
        void onItemLongClick(View view);
    }
```



Anexo: RecyclerView - Eventos en ViewHolder - Paso 2

Dentro de MiAdaptador defino un método para capturar en la interfaz el escuchador de los eventos

```
// Defino un método para que desde la actividad nos pasen el escuchador
// y asignarlo a la variable declarada
void setClickListener(myItemClickListener itemClickListener) {
    this.mClickListener = itemClickListener;
}
```



Anexo: RecyclerView - Eventos en ViewHolder - Paso 3

En la clase que tenemos que extiende la clase RecyclerView.ViewHolder, implementamos además los métodos de los eventos a capturar de la clase View.

```
public class ViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener, View.OnLongClickListener{
```



Anexo: RecyclerView - Eventos en ViewHolder - Paso 4

En la clase ViewHolder tenemos que implementar los siguientes pasos:

- Capturar los eventos que queremos tratar en el constructor
- Sobrescribir los eventos que queremos capturar para que llamen a los eventos de la variable del interfaz.



Anexo: RecyclerView - Eventos en ViewHolder - Paso 4

```
public class ViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener, View.OnLongClickListener {

    TextView myTextView;

    ViewHolder(View itemView) {
        super(itemView);
        myTextView = itemView.findViewById(R.id.tvAnimalNombre);
        itemView.setOnClickListener(this);
        itemView.setOnLongClickListener(this);
    }

    @Override
    public void onClick(View view) {

        if (mClickListener != null) mClickListener.onItemClick(view);
    }

    @Override
    public boolean onLongClick(View view) {

        if (mClickListener != null) mClickListener.onItemLongClick(view);
        return true;
    }
}
```



Anexo: RecyclerView - Eventos en ViewHolder - Paso 5

En la clase MainActivity implementamos también la interfaz definida.

```
public class MainActivity extends AppCompatActivity implements MiAdaptador.myItemClickListener{
```

Asignamos al adaptador el escuchador del click y definimos el tratamiento de los eventos

```
adapter = new MiAdaptador( context: this, animalNames);  
adapter.setOnClickListener(this);  
recyclerView.setAdapter(adapter);
```

Anexo: RecyclerView - Eventos en ViewHolder - Paso 5

```
RecyclerView recyclerView = findViewById(R.id.rvAnimales);
recyclerView.setHasFixedSize(true);
LinearLayoutManager layoutManager = new LinearLayoutManager(context, this);
recyclerView.setLayoutManager(layoutManager);
adapter = new MiAdaptador(context, this, animalNames);
adapter.setClickListener(this);
recyclerView.setAdapter(adapter);

DividerItemDecoration midividerItemDecoration = new DividerItemDecoration(recyclerView.getContext(),
    layoutManager.getOrientation());
recyclerView.addItemDecoration(midividerItemDecoration);
}

@Override
public void onItemClick(View view) {
    TextView myTV = view.findViewById(R.id.tvAnimalNombre);
    myTV.setBackgroundColor(Color.RED);
}

@Override
public void onItemLongClick(View view) {
    TextView myTV = view.findViewById(R.id.tvAnimalNombre);
    myTV.setBackgroundColor(Color.WHITE);
}
}
```