


Open · Opened 3 days ago by  **Rubén Montero**

Completando el RecyclerView

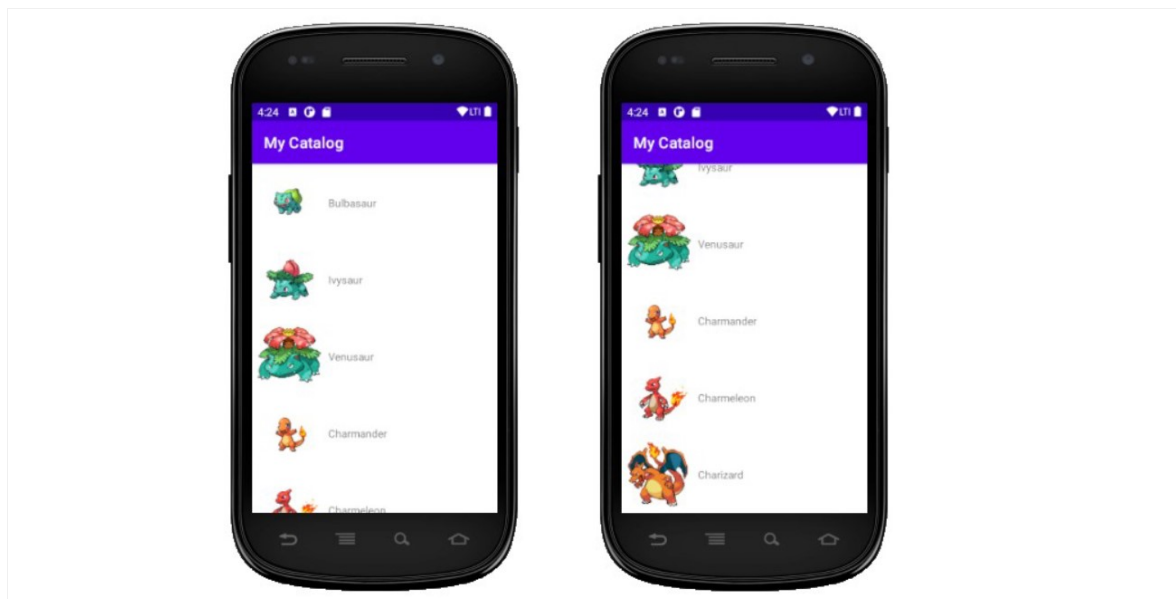
Resumen

- Añadiremos un nuevo archivo `ClipsAdapter.java`
- Entenderemos e implementaremos los tres métodos esenciales de un `RecyclerView.Adapter`
- Pondremos en marcha nuestro `RecyclerView` en `MainActivity.java`

Descripción

Como ya sabemos, un [RecyclerView](#) es un componente que presenta una lista de elementos al usuario, y *recicla* cíclicamente las celdas para ahorrar memoria RAM.

Así, el usuario puede hacer *scroll* en una cantidad indefinida de elementos:



Lo bueno es que nosotros no tenemos que implementar esa [complicada lógica de reciclaje](#). Sólo tenemos que darle lo que ella necesita y hará el trabajo por nosotros.

¿Qué necesita?

Necesita la respuesta a estas tres preguntas:

1. *¿Cuántas celdas debo mostrar?*
2. *¿Cómo es (visualmente) cada celda?*
3. *¿Qué contenido le corresponde a cada celda?*

Vamos a darle esas respuestas mediante un *adaptador*. En concreto, una clase `RecyclerView.Adapter`.

La tarea

Crea una nueva clase Java y llámala `ClipsAdapter`. **Incluye** un atributo privado de tipo `ClipsList`, y un método constructor sencillo, donde asumimos que *alguien* le pasará al adaptador los datos del servidor. **Así:**

```
public class ClipsAdapter {
    private ClipsList clipsToPresent;

    public ClipsAdapter(ClipsList clips) {
        this.clipsToPresent = clips;
    }
}
```

A continuación, **añade** la cláusula de herencia (`extends`) a `ClipsAdapter.java` , como se indica a continuación:

```
import androidx.recyclerview.widget.RecyclerView;

public class ClipsAdapter extends RecyclerView.Adapter<ClipViewHolder> {
    private ClipsList clipsToPresent;

    // ...
```

(Observa que incluimos `<ClipViewHolder>` en la declaración. ¡Nuestra clase! Este uso entre los símbolos "mayor que" y "menor que" `< >` se denomina genericidad, y es algo similar a usar `ArrayList<String>`).

Verás un error. **Usa** la ayuda de Android Studio para autogenerar los tres métodos obligatorios en un `RecyclerView.Adapter` .

`getItemCount`

- ¿Cuántas celdas debo mostrar?

Completa este método devolviendo el tamaño del atributo `ClipsList` que añadimos al principio de la tarea. **Así**:

```
@Override
public int getItemCount() {
    return this.clipsToPresent.getClips().size();
}
```

`onCreateViewHolder`

- ¿Cómo es (visualmente) cada celda?

Aquí toca devolver un `ViewHolder` . ¡Nuestro `ClipViewHolder` !

Añade este código al método `onCreateViewHolder` :

```
@NonNull
@Override
public ClipViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    // 1. Necesitamos un LayoutInflater. Lo creamos a partir de un Context
    LayoutInflater inflater = LayoutInflater.from(parent.getContext());
    // 2. Con el LayoutInflater, 'inflamos' el XML y generamos una View
    View cellView = inflater.inflate(R.layout.recycler_view_cell, parent, false);
    // 3. Esta View es la que pasamos al constructor de ClipViewHolder.
    // ¡Y ya está listo!
    ClipViewHolder cellViewHolder = new ClipViewHolder(cellView);
    return cellViewHolder;
}
```

(Como puedes ver, en este método "inflamos" manualmente la `recycler_view_cell.xml` y se la pasamos al constructor de `ClipViewHolder.java` de la tarea anterior).

`onBindViewHolder`

- ¿Qué contenido le corresponde a cada celda?

Este es el único método en el que no haremos un `return` . Simplemente, recibimos por parámetro:

- Un `ClipViewHolder`
- Un `int` indicando la posición

...y en el `ClipViewHolder` debemos *pintar* los datos asociados a esa posición.

Para empezar, cambia a la clase `ClipViewHolder.java` . Allí, **añade** este nuevo método, sustituyendo el comentario adecuadamente:

```
public void bindData(Clip clip) {
    this.textView.setText(/* Usamos el getter de clip para obtener el título */);
}
```

¡Ya puedes regresar a `ClipsAdapter.java`! **Completa** `onBindViewHolder` invocando el método que acabas de crear. **Así:**

```
@Override
public void onBindViewHolder(@NonNull ClipViewHolder holder, int position) {
    // Usamos .get(position) para acceder al 'enésimo' elemento de la lista
    // O sea, el correspondiente a la posición 'position'
    Clip dataForThisCell = this.clipsToPresent.getClips().get(position);
    holder.bindData(dataForThisCell);
}
```

Hemos terminado nuestro trabajo en `ClipsAdapter.java`. Ahora falta darle uso.

Encajando las piezas

En `MainActivity.java`, **añade** un atributo de tipo `RecyclerView` e **inicialízalo** mediante `findViewById` en `onCreate`:

```
public class MainActivity extends AppCompatActivity {
    // ...
    private RecyclerView recyclerView;
    // ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // ...
        this.recyclerView = findViewById( /* Aquí, el ID que le diste a tu <RecyclerView> en activity_main.xml */ );
        // ...
    }
}
```

Luego, en el método `setClips` (que es donde trabajamos con los datos del servidor) **instancia** un `ClipsAdapter` y **vincúlalo** a tu `RecyclerView` invocando `.setAdapter`. **Así:**

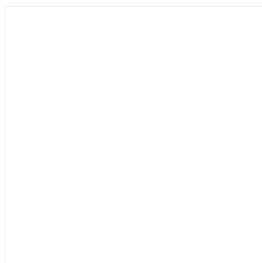
```
public void setClips(ClipsList clips) {
    this.clips = clips;
    ClipsAdapter myAdapter = new ClipsAdapter(this.clips);
    recyclerView.setAdapter(myAdapter);
}
```

Por último, **invoca** también `.setLayoutManager`¹ así:

```
public void setClips(ClipsList clips) {
    this.clips = clips;
    ClipsAdapter myAdapter = new ClipsAdapter(this.clips);
    recyclerView.setAdapter(myAdapter);
    + recyclerView.setLayoutManager(new LinearLayoutManager(this));
}
```


¡Enhorabuena! Has implementado tu primer `RecyclerView`. ¡Es un paso muy importante!

Según cómo hayas creado tu `recycler_view_cell.xml`, el resultado será *más o menos* como el siguiente:



Por último

Sube tus cambios al repositorio en un nuevo *commit*.

-
1. Un `RecyclerView` debe contar con un `LayoutManager`. Sirven para posicionar las celdas de maneras distintas. Nosotros, como queremos mostrar una lista (los elementos ocupan todo el ancho), empleamos `LinearLayoutManager`. ¡Podríamos mostrar una cuadrícula empleando un `GridLayoutManager` ! 
-



[Rubén Montero @ruben.montero](#) changed milestone to [%Sprint 2](#) 3 days ago