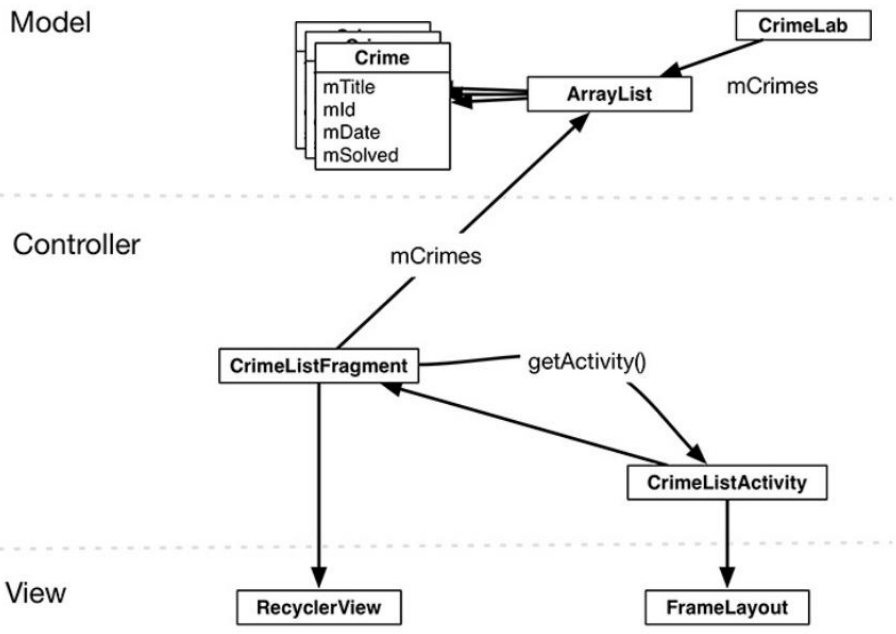
7. Listas y RecyclerView

Este capítulo corresponde al capítulo 8 del libro oficial *Android Programming Guide Ranch*.

La aplicación Criminal Intent por ahora tan sólo es una pantalla del detalle de un crimen. En este capítulo se actualizará para que sea una aplicación en la que tengamos una lista de crímenes.

Para que el alumno entienda qué es lo que se va a realizar y como se va a estructurar en este capítulo, se presenta el siguiente esquema.



En este esquema aparece la figura de un nuevo objeto llamado CrimeLab, que será la clase que contendrá todos los crímenes.

Para mostrar la lista de crímenes necesitaremos una nueva Activity, que será la CrimeListActivity, que dentro contendrá un fragment llamado CrimeListFragment.

Nota: La activity CrimeActivity y el fragment CrimeFragment que hemos creado en el capítulo anterior se utilizarán posteriormente como pantalla de detalle cuando cliquemos en un crimen de la lista.

El primer paso será recrear el modelo de datos para que la nueva clase CrimeLab contenga una lista de crímenes. Ésta se ha hecho mediante el patrón Singleton ya que la instancia de la clase perdurará durante toda la ejecución de la aplicación. La nueva clase CrimeLab no es una buena solución para almacenar la información de los crímenes a lo largo del tiempo, pero nos permitirá que la app funcione correctamente.

Cuando observemos la implementación de la clase CrimeLab, veremos que en la variable existe el prefijo s, que, según el convenio de escritura de código en Android, significa que ésta es una variable estática de la clase.

Por el momento haremos que los crímenes que almacenará esta clase no los ha creado el usuario, sino que los crearemos nosotros manualmente: tendrán como nombre “Crime X” y estarán resueltos aquellos en los que “X % 2 = 0”.

Nota: Como la activity que hemos creado anteriormente no se refiere a ningún fragment en particular, podríamos usar ese mismo layout (activity_crime.xml) para cualquier activity que contenga únicamente un fragment. Por ello se ha decidido cambiar el nombre a activity_fragment.xml.

Para crear la CrimeListActivity, necesitaremos exactamente el mismo código que el que hemos usado para crear la CrimeActivity. Lo único que deberemos cambiar será el tipo de fragment que instanciaremos. En vez de copiar el código para cada activity que contenga un fragment, crearemos una clase abstracta de activity a la cual tan solo le especificaremos que fragment deberá cargar.

```java
public abstract class SingleFragmentActivity extends AppCompatActivity {

    protected abstract Fragment createFragment();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);

        if (fragment == null) {
            fragment = createFragment();
            fm.beginTransaction()
                    .add(R.id.fragment_container, fragment)
                    .commit();
        }
    }
}
```

Como podemos ver, la clase SingleFragmentActivity, cargará el layout en el que tan sólo existe un fragment en toda la activity y además se le especificará que Fragment será el que deba cargar.

Si echamos un ojo a la clase CrimeActivity, veremos que ésta ha sido reducida considerablemente.

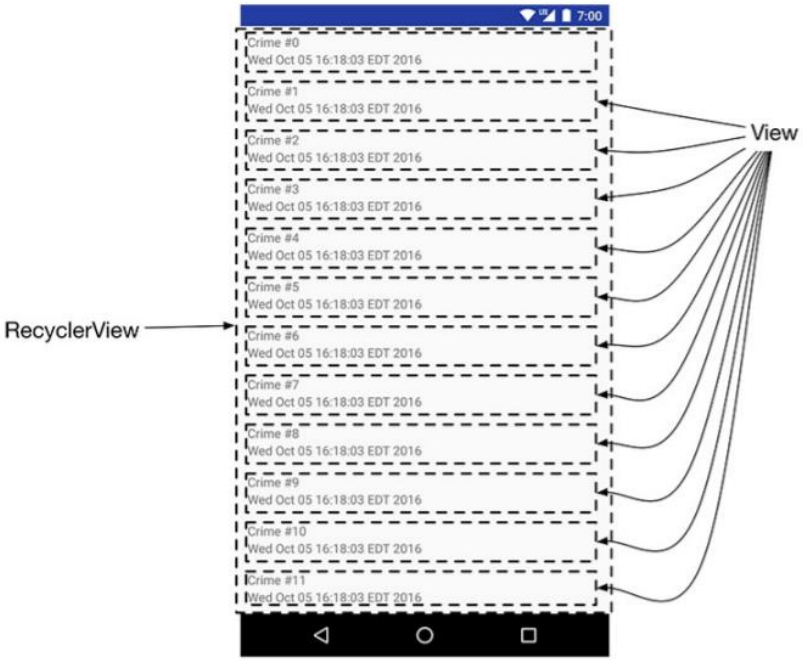
El siguiente paso será crear dos nuevas clases: CrimeListActivity y CrimeListFragment.

Cada vez que creemos una nueva Activity, esta deberá ser declarada en el archivo Manifest de la aplicación.

RecyclerView, Adapter y ViewHolder

Ahora deseamos que el CrimeListfragment muestre la lista de los crímenes al usuario. Para ello utilizaremos una RecyclerView.

La RecyclerView es una clase que muestra una lista de objetos, donde cada uno será un objeto del tipo View. Para nuestra aplicación de Criminal Intent, necesitaremos que cada una de esas Views muestre un LinearLayout vertical, que contenga dos TextView (uno para el identificador del crimen y otro para la fecha), tal y como se muestra en la figura siguiente.

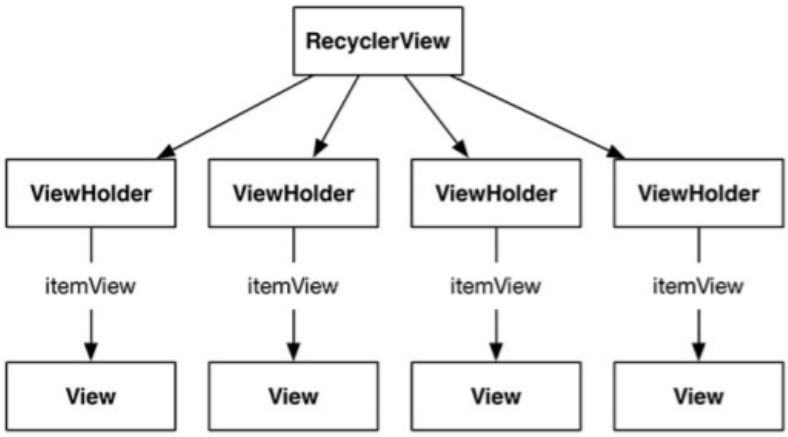


A modo de demostración de cómo funciona la RecyclerView, dispondremos de 100 crímenes. ¿Eso significa que tendremos 100 crímenes cargados en memoria? Como era de espera: NO.

Como bien indica el nombre de RecyclerView, ésta recicla los elementos en pantalla, ya que no se van a poder mostrar los 100 crímenes a la vez. La RecyclerView se encargará de cargar en memoria tantos elementos en memoria como ítems de la lista sea capaz de mostrar (por espacio). Con lo cual, si en pantalla tan solo se pueden mostrar 12 crímenes, la RecyclerView cargará 13, ya que se mostrarán siempre 12 pero cuando hagamos un scroll por la pantalla se deberá cargar uno más y mantener los 11 previos, que se seguirán mostrando, en memoria. Se cargan siempre n+1 elementos en memoria ya que al hacer un pequeño scroll podremos ver la parte inferior de un crimen y la parte superior de otro.

ViewHolders y Adapters

La única responsabilidad de la RecyclerView es la de reciclar los TextViews (en este ejemplo) y posicionarlos en pantalla. La RecyclerView como tal, nunca crea las Views por sí misma, sino que crea ViewHolders, que será los encargados de montar cada una de sus vistas.

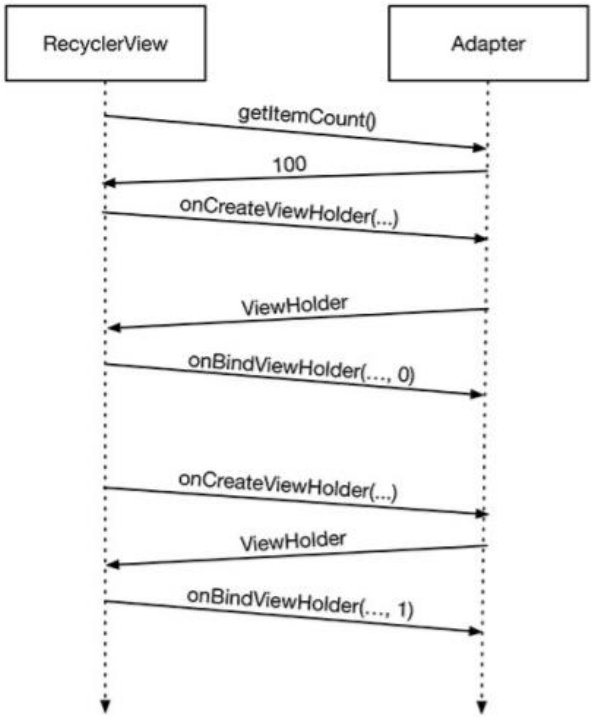


Para ser más precisos, una RecyclerView no crea tampoco por si misma los ViewHolders, sino que delega esa faena a un **Adapter**. Un Adapter es un controlador que permanece en la capa intermedia entre la vista de la RecyclerView y los datos. Las funciones del Adapter son las siguientes:

- Crear los ViewHolders que sean necesarios.
- Llenar los ViewHolders con la información del crimen de cada uno de ellos.

Para crear un adapter, primero lo definiremos como una subclase de **RecylcerView.Adapter**, que contendrá la lista de los crímenes que obtendremos de la clase **CrimeLab**.

Cuando la RecyclerView necesite mostrar un nuevo elemento, esta consultará al adapter para que éste le de el ViewHolder a mostrar.



El procedimiento que sigue una RecyclerView para mostrar el contenido es el siguiente.

1. Pregunta al adapter cuantos elementos existen en la lista – **getItemCount()**.
2. Por cada elemento:
    a. Crear el elemento - **onCreateViewHolder(…)**
    b. Mostrar el elemento – **onBindViewHolder(…, index)**

Una vez se haya llenado la pantalla y no quepan más crímenes en pantalla, se dejará de llamar a las funciones **onCreateViewHolder** y **onBindViewHolder** para así ahorrar memoria.

¡Pongámonos manos a la obra!

Nuestra RecyclerView estará albergada en el CrimeListFragment, así que deberemos crear un archivo XML para ello: *fragment_crime_list.xml*.

En el archivo de la clase CrimeListFragment.java, ahora deberemos recuperar la RecyclerView de la vista (de la misma forma que recuperábamos los campos de texto y los botones).

Además, deberemos configurar el layout manager de la RecyclerView para que ésta sepa cómo debe mostrar los ítems de la lista y en qué dirección irá el scroll.

Como bien hemos comentado anteriormente, cada uno de los elementos constará de una View, que se mostrará en la lista. Esta View la definiremos en un nuevo archivo XML llamado list_item_crime.xml.

Este archivo será, como comentábamos, un linear layout vertical con dos campos de texto en su interior.

El ViewHolder, del que hemos hablado antes, lo podemos definir como una inner class dentro de la clase CrimeListFragment. Este ViewHolder inflará la vista (list_item_crime.xml) de cada elemento de la lista.

En el constructor inflaremos la vista mediante el constructor del padre del que hereda nuestro ViewHolder, es decir, mediante el constructor de la clase RecylcerView.ViewHolder.

Crearemos el adapter también como una inner class de CrimeListFragment, y recibirá como parámetro en su constructor la lista de los crímenes, que obtendremos del CrimeLab.

Implementamos también los métodos getItemCount, onBindViewHolder y onCreateViewHolder que hemos explicado previamente su uso.[1]

Una vez creadas nuestras propias clases ViewHolder y CrimeAdapter, creamos una instancia de nuestro adapter en la clase CrimeListFragment y añadimos el método updateUI, que nos inicializará el adapter y lo vinculará a la RecyclerView.

Si ahora probamos de ejecutar nuestra aplicación veremos que la lista contiene muchos elementos, pero no tienen información, tan sólo dos textos que hemos predefinido nosotros.

[1] Ver código fuente de todas las implementaciones de las funciones del ViewHolder y del Adapter que hemos creado.

Ahora nos falta añadir al ViewHolder, que recupere los dos textos del ítem que está construyendo y le ponga la información que le definiremos como argumento en su constructor, ya que un ViewHolder se crea para un único ítem.

Finalmente necesitaremos que el ViewHolder y el CrimeAdapter tengan un método llamado bind:

- En el ViewHolder, el método bind se encargará de, a partir de un Crimen pasado como parámetro, cargar el texto en los campos de la vista.
- En el CrimeAdapter, el método bind se encargará de llamar al método bind del ViewHolder pasado como parámetro para el crimen con el índice pasado como parámetro también. De esta forma, cuando se llame a la función onBindViewHolder, llamaremos nosotros a la función bind del ViewHolder que queremos mostrar.

Por último, añadiremos un Listener al ViewHolder. Ese Listener será el propio ViewHolder, ya que lo modificaremos para que demás de que extienda de RecyclerView.ViewHolder, implemente también Views.OnClickListener. Ahora podremos añadirle como OnClickListener del itemView del propio ViewHolder, él mismo, e implementar el método onClick para que la aplicación muestre un Toast cuando se aprieta un elemento de la lista.

Challenges

Para los alumnos más curiosos, se les propone intentar solucionar los distintos retos que se proponen a continuación.

- Añadir un campo a la clase Crime, para que un crimen sea más serio que otro, y mostrar aquellos que sean crímenes más serios con una vista distinta. Se deja al alumno que haga esta nueva vista del crimen a su gusto. Cuando quede resuelto, se podrá ver en una misma lista, como hay elementos que se muestran distintos a otros.