Creación de RecyclerView con Adapter personalizado

1. Añadimos el Fragment en el XML del activity. Si es un fragmento dinámico, añadimos un <FrameLayout> con id=”container”. Se puede poner como elemento padre quitándole el padding.  
   Si es un fragmento estático, este paso lo haremos tras el paso 3, puesto que debemos arrastrarlo desde la pestaña ‘Design’ un <fragment>, y nos pedirá de qué tipo es el Fragment, y eso sólo se puede seleccionar si ya hemos creado previamente el Fragment.
2. Crear los siguientes paquetes:  
   -activities  
   -fragments  
   -adapters  
   -pojos  
   -interfaces
3. Crear ‘New -> Fragment -> Fragment (List)
   1. **Object Kind:** El POJO que se utilizará. Poner el nombre deseado, aunque aún no se haya creado dicha clase.
   2. **Fragment class name:** Poner ‘ListadoPOJOFragment’, donde POJO lo sustituimos por el nombre de la POJO elegido en la línea anterior.
   3. Se deja marcado ‘Include fragment Factory methods’.
   4. **Column count:** Número de elementos que se visualizarán por cada fila. Si es más de 1, se usará un GridView.
   5. **Object content layout file name:** Al nombre que está le añadimos al final \_item, para diferenciar que será el xml utilizado para diseñar los elementos.
   6. **Adapter class name:** El nombre del adapter. Pulsar Finish.
4. Solucionar poco a poco todos los errores relacionados con:
   1. **DUMMY.**
      1. **Fragment:** Eliminar paquete dummy, buscar en el Fragment los objetos de tipo DummyItem y sustituirlos por la clase POJO, que vamos a crear en este mismo momento (se puede utilizar la bombilla roja para crear rápidamente, pero NO dar cuerpo al POJO aún). Borramos los imports del paquete dummy. No cambiar el dummy de la interfaz al final del Fragment, puesto que esta interfaz se borrará después para crear la nuestra en una clase aparte.
      2. **Adapter:** También debemos sustituir los tipos DummyItem por nuestro POJO. Para facilitar, se puede utilizar Type Migration. Eliminar imports de paquete dummy y el comentario Javadoc de la clase donde aparece DummyItem con un @link.
   2. **INTERFAZ.**
      1. Creamos una interfaz para gestionar los eventos click realizados en los elementos de la lista, y se implementará en el Activity. Se llamará ‘IPojoClickLIstener’, donde sustituímos la palabra ‘Pojo’ por el nombre de nuestra clase Pojo.  
         Añadimos la cabecera del método que se debe implementar en el Activity para gestionar los clicks, que por ejemplo podemos ponerle ‘void onPojoClick(Pojo p);’
      2. **Adapter:** Hacer ‘Type Migration’ de ‘OnListFragmentInteractionListener’ para cambiarla por la interfaz ‘IPojoClickListener’ que acabamos de crear. (Puede que en este punto primero haya que importar la interfaz que estaba para poder hacer refactor, si no utilizar CTRL+SHIFT+R).  
           
         Dentro del método onBindViewHolder() -> onClick(), cambiar la llamada de mListener.onListFragmentInteraction por la llamada al método creado en nuestra interfaz.
      3. **Fragment:** Eliminamos la interfaz que está creada al final de la clase, ya que ya tenemos la nuestra creada. Borrar el comentario Javadoc de la clase que contiene un @link a dicha interfaz.  
           
         En el método onAttach cambiar la antigua interfaz por la que hemos creado nosotros.
5. **En el Adapter**, buscar el método onCreateViewHolder() y sustituir el layout.fragment\_item, por el layout que creamos en el asistente para pintar los elementos (el que le pusimos \_item al final).  
     
   En este punto, vamos a diseñar el layout con el que vamos a pintar cada uno de los ítems. Como diseño, aunque no será siempre, por lo general podemos utilizar un CardView como elemento padre, y dentro ponerle un RelativeLayout y empezar a introducir los elementos dentro.
6. Tras diseñar el layout**, vamos al Adapter**. Buscamos la clase interna ViewHolder y añadimos todos los elementos de la UI que vayamos a utilizar (obviar elementos decorativos) como atributos de la clase. Los inicializamos en el constructor, utilizando view.findViewById(…).
7. **Vamos a la clase POJO** que hemos creado y la completamos con los datos que necesitemos.
8. Vamos al método onBindViewHolder() del **Adapter**, que es el lugar donde le asignaremos los datos a cada elemento. Hacemos los sets a cada elemento con los datos que obtenemos del mItem, que será el elemento que vamos a pintar.
9. A nivel de la **clase Adapter** añadimos un nuevo atributo Context ctx. Vamos al constructor del Adapter y le añadimos un nuevo parámetro que recibirá, Context context y lo guardamos en ctx. Esto nos servirá para introducir el contexto del Activity en cualquier lugar de este Adapter si nos hiciese falta.
10. **Vamos al Fragment** y creamos el List<Pojo> como atributo de clase. En el método onCreateView(), tras terminar el if-else de mColumnCount se inicializa el List<Pojo> como ArrayList, se obtienen los datos de la manera que corresponda en cada momento y se cambia el DummyContent.ITEMS por ese List que acabamos de inicializar.  
      
    En el caso del ejercicio de la NASA, se hará en un Asynctask donde convertiremos los datos del xml a objetos.
11. Creamos las clases necesarias fijándonos en el XML. Habrá una clase por cada etiqueta que sea padre en el xml, añadiéndole las anotaciones correspondientes para cada atributo.  
    Las que utilizaremos nosotros serán:
    1. @Root. Se le pone a la clase si es una etiqueta padre.
    2. @ElementList(inline=true). El inline se utiliza SÓLO si el listado de elementos que queremos guardar NO está contenida dentro de otra etiqueta.
    3. @Element. Indica cada una de las etiquetas del xml.
    4. @Attribute. Son los que están como atributos de la etiqueta padre.