RecyclerView Avançat

Gestionant el RecyclerView

- Afegir/Modificar/Eliminar elements dinàmicament al RecyclerView
- Fent scroll cap els nous ítems
- Optimització de l'scrolling
- Establint un clickListener a un RecyclerView
- Efecte *selected* quan premem un element
- Articles per llegir i aprendre més

2n DAM

Programació Multimèdia i Dispositius Mòbils

Afegir/Modificar/Eliminar elements dinàmicament al RecyclerView

És molt habitual, que l'usuari puga afegir, modificar o eliminar elements d'una llista. Fins ara havíem vist com utilitzar una llista de manera estàtica, per a mostrar elements. Ara volem interactuar amb el contingut de la llista.

Podem aprofitar el Float Action Button, que per defecte ens crea en qualsevol activity, per donar-li la funció d'afegir un element més.

De manera similar (i per no extendre més l'exemple) podríem afegir-li a cada item de la llista un longClickListener, per a que quan l'usuari faça un click llarg sobre un ítem, ens aparega un diàleg preguntant-nos si el volem eliminar o no (Figura 1).

Després ens centrarem amb els events. De moment, volem vore com podem interactuar amb la llista per afegir-li, modificar i eliminar ítems.

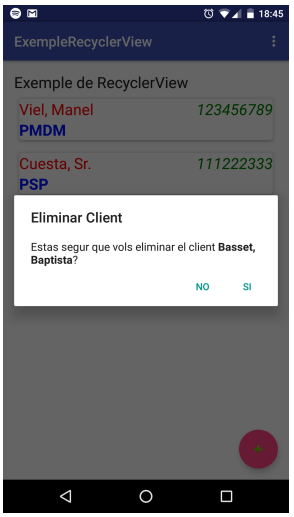


Figura 1. -Diàleg quan fem un click llarg sobre un element de la llista

Interactuant amb el conjunt de dades:

L'única manera que tenim d'interactuar amb la llista, és **interactuar** directament **sobre el conjunt de dades i informant**, després, **a l'adaptador** dels canvis que s'han produït.

Així, tenint en compte que la llista que estàvem utilitzant com a conjunt de dades era un objecte List anomenat *llistaClients*:

```
public class AdaptadorRecyclerViewClient extends
                    RecyclerView.Adapter<AdaptadorRecyclerViewClient.ClientViewHolder> {
    private List<Client> llistaClients; // Llista amb els clients que volem mostrar
    …
```

I que a l'Activity principal, l'havíem convertit en un ArrayList:

```
List clients;
clients = new ArrayList(); //Conjunt de dades a utilitzar
```

Utilitzarem els mètodes add(), remove(), o qualsevol que necessitem per a treballar amb el conjunt de dades de l'ArrayList. D'aquesta manera aconseguim interactuar *directament* contra el conjunt de dades. Ara només ens queda *informar* a l'Adaptador de que les dades del conjunt de dades han canviat.

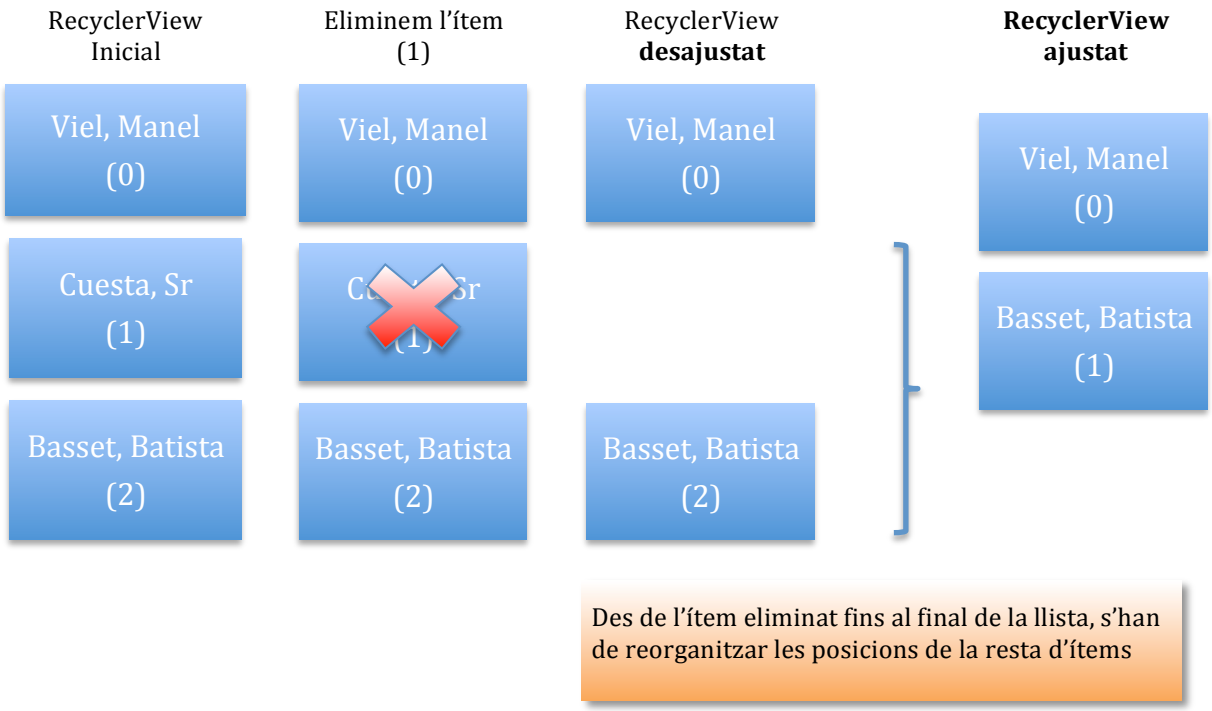
Informant a l'Adaptador

Depenent dels canvis produïts al conjunt de dades, necessitarem informar a l'adaptador d'una manera o d'altra. A la taula 1 podeu trobar les distintes maneres que tenim d'informar a l'adaptador.

Mètode	Descripció
notifyItemChanged(int pos)	Informa que l'ítem de la posició *pos* ha canviat
notifyItemInserted(int pos)	Informa que l´ítem que ocupa la posició *pos*, ha sigut afegit
notifyItemRemoved(int pos)	Informa que l'ítem que ocupava la posició *pos* ha sigut eliminat
notifyDataSetChanged()	Informa que el conjunt de dades ha canviat. Utilitzeu-lo només com a últim recurs.

Taula 1. - Mètodes per informar a l'Adaptador

Atenció: quan eliminem algun element de la llista, es produeix una reestructuració dels índex del ítems.



Per a reorganitzar els ítems des de la posició eliminada fins al final de la llista, cridarem al mètode:

```
notifyItemRangeChanged(posicio, getItemCount());
```

Així, seguin amb el nostre exemple, per eliminar un ítem de la llista de clients, farem:

```
llistaClients.remove(posicio);   //eliminem del conjunt de dades l'element que ocupa la posició posicio
notifyItemRemoved(posicio);   // informem a l'Adaptador que s'ha eliminat l'ítem que ocupa la posició posicio
notifyItemRangeChanged(posicio, getItemCount());  //reorganitzem els ítems de la llista des de posicio fins el final
```

Si no fem aquesta reorganització, l'aplicació ens fallarà de manera indeterminada per culpa dels índex de la llista.

Fent scroll cap els nou ítems

Quan afegim dinàmicament ítems a una llista, ens pot interessar fer un scroll cap a eixe ítem.

- Si afegim elements per la part superior de la llista, farem scroll cap a la posició 0.
- Si afegim elements per la part inferior de la llista, farem scroll cap a la posició *numero_d'items_de_la_llista -1*

```
objecteRecyclerView.scrollToPosition(int posicio);
```

Al nostre exemple, afegirem elements pel final de la llista, per tant farem:

```
rvLM.scrollToPosition(clients.size()-1);
```

Optimització de l'scrolling

Només en el cas que tots els component del RecyclerView tinguen la mateixa amplària i alçada, podem afegir a l'objecte RecyclerView:

```
objecteRecyclerView.setHasFixedSize(true);
```

Amb açò aconseguirem tindre un scrolling més fluid.

Establint un clickListener a un RecyclerView

A diferència del ListView, el RecyclerView no té un Listener tipus *setOnItemClickListener()*, que ens permetria d'una manera molt senzilla capturar els clicks produïts en un element del RecyclerView.

Si cerqueu per Internet, voreu que hi han diverses maneres d'aconseguir aquest objectiu. Jo vaig a explicar ací una manera de fer-ho, però tingueu clar que es pot fer d'altres maneres, i totes són correctes.

Per al nostre exemple, volíem que quan férem un "long click" ens mostrara un diàleg per a confirmar si volíem eliminar eixe ítem de la llista o no.

Començarem per crear **dins el Holder** una referència a cada view que construirem i afegirem a la llista.

```
public static class ClientViewHolder extends RecyclerView.ViewHolder implements
AdapterView.OnClickListener{
   public View v;  // Referència al view que anem a construir
…
```

I modificarem el constructor del Holder per a guardar la referència d'eixe view.

```
//Constructor de ClientViewHolder
public ClientViewHolder(View v) {
   super(v);
   this.v=v;  // Referenciem l'objecte View
   nom_i_cognoms = (TextView) v.findViewById(R.id.cognoms_i_nom);
   …
```

Després, a l'Adaptador, buscarem el mètode onBindViewHolder(), que és on s'enllacen les dades del view creat amb el conjunt de dades , i li afegirem al view el LongClickListener que necessitem.
En roig he marcat el codi afegit al mètode onBindViewHolder.

```
public void onBindViewHolder(ClientViewHolder holder, int position) {
    final int pos=position;

    holder.telefon.setText(llistaClients.get(position).getTelefon());
    holder.nom_i_cognoms.setText(this.llistaClients.get(position).getCognoms() + ", " +
this.llistaClients.get(position).getNom());
    holder.malnom.setText(this.llistaClients.get(position).getMalnom());

    // Li afegim el OnItemLongClickListener al l'objecte holder, que representa un element de la llista
    holder.v.setOnLongClickListener(new View.OnLongClickListener() {
        @Override
        public boolean onLongClick(View v) {
            dialegConfirmaEliminacio(v, pos); // Creem un dialeg per a confirmar l'acció
            return true; // hem gestionat l'event
        }
    });
}
```

1r. Guardem la posició de l'ítem seleccionat en una constant anomenada *pos*. Acó ho fem per necessitat ja que al utilitzar-lo dins del onLongClick() no podem utilitzar variables que no siguen constant.
2n. El mètode onLongClick torna un boolean indicant si hem gestionat l'event o no. En aquest cas, com si que el gestionem, tornem *true*.

Per a afegir Listeners dins els elements d'un view, tal i com ja hem vist a classe, en el Holder afegirem tants listeners com vulguem.

```
//Constructor de ClientViewHolder
public ClientViewHolder(View v) {
    super(v);
    this.v=v;
    nom_i_cognoms = (TextView) v.findViewById(R.id.cognoms_i_nom);
    nom_i_cognoms.setOnClickListener(this); // Li afegim el Listener al textView nom_i_cognoms
    malnom = (TextView) v.findViewById(R.id.malnom);
    telefon = (TextView) v.findViewById(R.id.telefon);
    telefon.setOnClickListener(this); //Li afegim el listener al textview telèfon
}
```

Per a crear el diàleg de confirmació, encara que no és objecte d'aquest article, us mostre el codi utilitzar per a generar-lo.

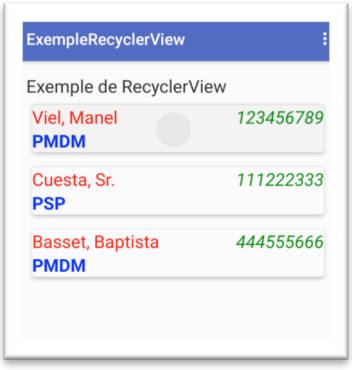
```
/**
 * Mètode que crea un diàleg de confirmació per asegurar-nos que l'usuari realment vol eliminar
 * un registre de la llista
 * @param v View que es vol eliminar
 * @param posicio posicio que ocupa l'element a eliminar (es final ja que accedim des d'una classe interna)
 */
public void dialegConfirmaEliminacio(View v, final int posicio){

    // 1. Instanciem el constructor d'un AlertDialog.Builder
    AlertDialog.Builder builder = new AlertDialog.Builder(v.getContext());
    // 2. Encadenem uns quants mètodes setters per definir el nostre dialeg
    String nomComplet = "<b>"+llistaClients.get(posicio).getCognoms()+",
"+llistaClients.get(posicio).getNom()+"</b>";
    String msg = "Estas segur que vols eliminar el client " + nomComplet+"?";
    builder.setMessage(Html.fromHtml(msg));
    builder.setTitle("Eliminar Client");
    // 3. Li afegim els botons Si i No, amb les accions a realitzar
    builder.setPositiveButton("Si", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            llistaClients.remove(posicio);
            notifyItemRemoved(posicio);
            notifyItemRangeChanged(posicio, getItemCount());
        }
    });
    builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            // Es cancel-la el diàleg
        }
    });
    // 4. Creem  l'AlertDialog
    AlertDialog dialog = builder.create();
    // 5.- Mostrem el dialeg
    dialog.show();
}
```

Efecte *selected* quan premem un element

Per a aconseguir aquest efecte, cal anar al layout on definim ***un element*** del RecyclerView, i afegir-li al layout arrel (**root layout)** l'atribut:

```
android:background="?android:attr/selectableItemBackground"
```



Si estem utilitzant un CardView per a mostrar els elements del RecyclerView, utilitzarem, **dins la definició del CardView**, els següents atributs:

```
android:foreground="?android:attr/selectableItemBackground"
android:clickable="true"
android:focusable="true"
```

Articles per llegir i aprendre més:

http://antonioleiva.com/recyclerview-listener/
https://guides.codepath.com/android/Using-the-RecyclerView
http://nemanjakovacevic.net/blog/english/2016/01/12/recyclerview-swipe-to-delete-no-3rd-party-lib-necessary/