CFGS: DESENVOLUPAMENT D'APLICACIONS MULTIPLATAFORMA

Mòdul: Programació multimèdia i dispositius mòbils

TEMA 4: INTERFÍCIES D'USUARI. CONTROLS DE SELECCIÓ

Germán Gascón Grau
ggascon@gmail.com

Introducció

- Una vegada repassats els controls bàsics que podem utilitzar en les nostres aplicacions Android, anem a descriure els diferents controls de selecció disponibles a la plataforma.
- Igual que en altres frameworks, Android disposa de diversos controls que ens permeten seleccionar una opció dins d'una llista de possibilitats.
- Així, podrem utilitzar per exemple llistes desplegables (**Spinner**), llistes fixes (**ListView**), taules (**GridView**) o crear al nostre gust (**RecyclerView**) el nostre propi control de selecció.
- Però abans de ficar-nos de ple a veure els diferents components, cal descriure un element comú a tots ells, els **adaptadors**.

Adaptadors

- Un adaptador representa una mena d'**interfície comuna al model de dades** que hi ha per darrere de tots els controls de selecció que hem comentat. Dit d'una altra manera, tots els controls de selecció accediran a les dades que contenen a través d'un adaptador.
- A més de proveir de dades als controls visuals, l'adaptador també serà **responsable de generar** a partir d'aquestes dades, **les vistes específiques** que es mostraran **dins el control de selecció**.
- Per exemple, si cada element d'una llista estigués format per una imatge i diverses etiquetes, el responsable de generar i establir el contingut de tots aquests "sub-elements" a partir de les dades serà el propi adaptador.

Tipus d'adaptadors

- Android proporciona de sèrie diversos tipus d'adaptadors senzills, encara que podem estendre la seva funcionalitat fàcilment per adaptar-los a les nostres necessitats.
- Els més comuns són els següents:
  - **ArrayAdapter**: Proveeix de dades a un control de selecció a partir d'un **array d'objectes** de qualsevol tipus.
  - **SimpleAdapter**. S'utilitza per **mapejar dades** sobre els diferents controls definits en un **fitxer XML** de layout.
  - **SimpleCursorAdapter**. S'utilitza per **mapejar les columnes** d'un cursor obert en una **base de dades** sobre els diferents elements visuals continguts en el control de selecció.

Exemple ArrayAdapter I

- Vegem com crear un adaptador de tipus **ArrayAdapter** per treballar amb un array genèric de Java:

```
final String[] datos =
        new String[]{"Elem1","Elem2","Elem3","Elem4","Elem5"};

ArrayAdapter<String> adaptador =
        new ArrayAdapter<String>(this,
            android.R.layout.simple_spinner_item, datos);
```

- A la primera línia creem un array de Strings i l'inicialitzem amb 5 elements.
- Després creem l'**ArrayAdapter** passant-li **3 paràmetres** al constructor:
  - El **primer paràmetre**, el **context** que normalment serà una referència a l'Activity on es crea l'adaptador.
  - El **segon paràmetre**, el **id del layout** sobre el qual es mostraran les dades. Podem passar algun ja predefinit, com en l'exemple (R.layout.simple_spinner_item), o bé crear un layout personalitzat.
  - El **tercer paràmetre**, l'**array** que conté les **dades** a mostrar.

Exemple ArrayAdapter II

- Una alternativa a tindre en compte si les dades a mostrar en el control són **estàtiques**, és definir una llista de possibles valors com un recurs de tipus **string-array**.
- Per a això, primer crearíem un fitxer **XML** (valores_array.xml) a la carpeta **/res/values** i inclouríem en ell els valors seleccionables de la següent manera:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="valores_array">
        <item>Elem1</item>
        <item>Elem2</item>
        <item>Elem3</item>
        <item>Elem4</item>
        <item>Elem5</item>
    </string-array>
</resources>
```

- Per a crear l'adaptador utilitzaríem el mètode **createFromResource()** per fer referència a aquest array XML.

```
ArrayAdapter<CharSequence> adapter =
    ArrayAdapter.createFromResource(this,
        R.array.valores_array,
        android.R.layout.simple_spinner_item);
```

Spinner (I)

- Els **spinners** o llistes desplegables funcionen de manera similar a qualsevol control d'aquest tipus, l'usuari prem el control i es mostra una espècie de **llista emergent** a l'usuari amb **totes** les **opcions disponibles** i al seleccionar-se una d'elles aquesta queda fixada en el control.
- Per afegir una llista d'aquest tipus:

```
<Spinner android:id="@+id/CmbOpciones"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

- Les opcions per personalitzar l'aspecte visual del control (fons, color i mida de font, ...) són les mateixes que les comentades pels controls bàsics.

Spinner (II)

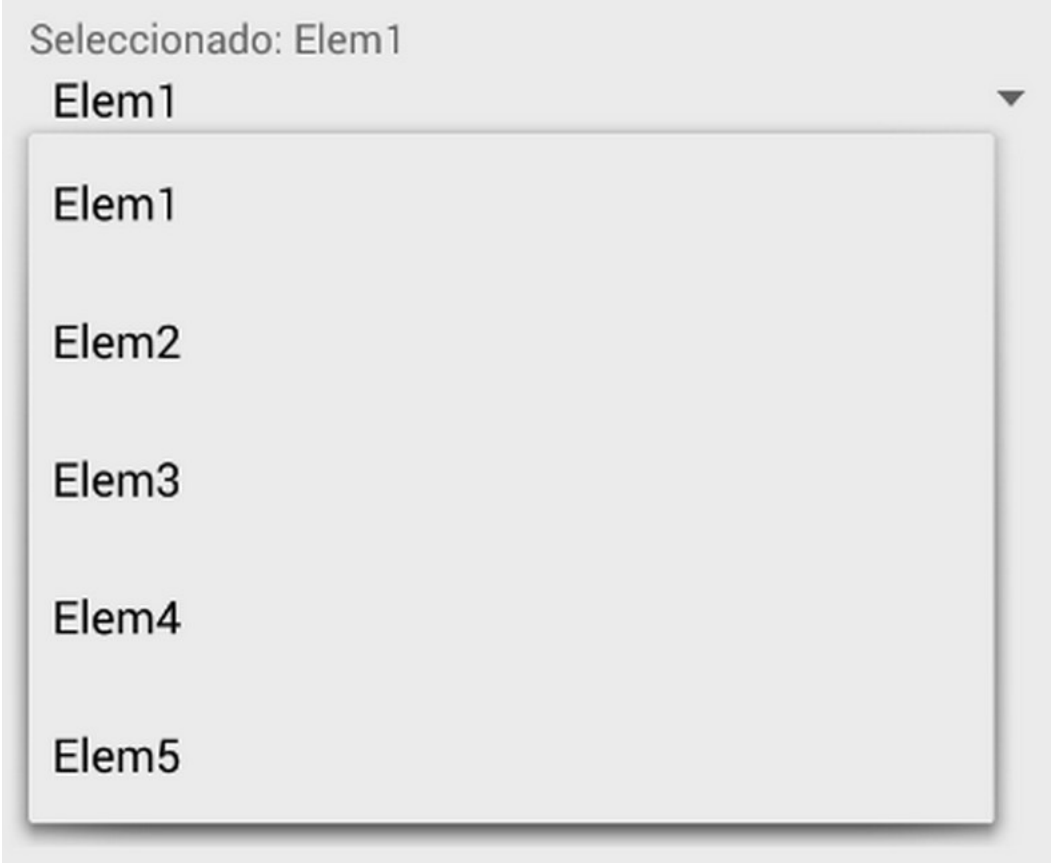
- Per **enllaçar** l'adaptador (i per tant les nostres dades) a aquest control utilitzarem el següent codi java:

```
private Spinner cmbOpciones;

//...

cmbOpciones = (Spinner)findViewById(R.id.CmbOpciones);

adaptador.setDropDownViewResource(
        android.R.layout.simple_spinner_dropdown_item);

cmbOpciones.setAdapter(adaptador);
```

- La segona línia és per personalitzar la llista amb tots els elements. Quan varem indicar en l'apartat anterior com construir un adaptador, vegerem com un dels paràmetres que li passàvem era l'**id** del layout que utilitzaríem per visualitzar els elements del control, però aquest layout només s'aplicarà a l'element seleccionat de la llista, és a dir quan el control no està desplegat. Per tant, mitjançant el mètode **setDropDownViewResource()** podem **personalitzar** el layout que es mostra quan la **llista** està **desplegada**. És aquest cas s'utilitza un layout ja predefinit en Android **R.layout_simple_spinner_dropdown_item**.

Spinner (III)

- L'exemple anterior quedaria de la següent manera:



Spinner (IV)

- Quant als esdeveniments llançats pel control **Spinner**, un dels més utilitzats serà el generat al seleccionar una opció de la llista desplegable, **onItemSelected**.
- Per capturar aquest tipus d'esdeveniments utilitzarem el mètode **setOnItemSelectedListener()** i li passarem com a paràmetre un objecte **AdapterView.OnItemSelectedListener()**

```
cmbOpciones.setOnItemSelectedListener(
        new AdapterView.OnItemSelectedListener() {
            public void onItemSelected(AdapterView<?> parent,
                                       android.view.View v, int position, long id)
                lblMensaje.setText("Seleccionado: " +
                        parent.getItemAtPosition(position));
            }

            public void onNothingSelected(AdapterView<?> parent) {
                lblMensaje.setText("");
            }
        });
```

ListView (I)

- Un control **ListView** mostra a l'usuari una **llista d'opcions seleccionables** directament sobre el propi control, **sense llistes emergents** com en el cas del control Spinner.
- En cas d'existir més opcions de les que es poden mostrar sobre el control es podrà fer **scroll** sobre la llista per accedir a la resta d'elements.
- Vegem com afegir un control **ListView** a la nostra interfície d'usuari:

```
<ListView android:id="@+id/LstOpciones"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
```

ListView (II)

- Podem modificar l'aspecte del control utilitzant les propietats de font i color ja comentades en articles anteriors.
- Per enllaçar les dades amb el control utilitzarem el mateix codi que el que hem vist en el control Spinner.

```
final String[] datos =
    new String[]{"Elem1","Elem2","Elem3","Elem4","Elem5"};

ArrayAdapter<String> adaptador =
    new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1, datos);

lstOpciones = (ListView)findViewById(R.id.LstOpciones);

lstOpciones.setAdapter(adaptador);
```

ListView (III)

- Per mostrar les dades de cada element hem utilitzat el **layout genèric** d'Android **android.R.layout.simple_list_item_1** per als controls de tipus **ListView**, format únicament per un **TextView** amb unes dimensions determinades.

ListView (IV)

- Quan siga necessari mostrar dades més complexes en la llista, per exemple que cada element de la llista estiga formada per diversos elements, haurem de **crear el nostre propi adaptador**.
- Suposant que tenim la següent classe:

```
public class Titular
{
    private String titulo;
    private String subtitulo;

    public Titular(String tit, String sub){
        titulo = tit;
        subtitulo = sub;
    }

    public String getTitulo(){
        return titulo;
    }

    public String getSubtitulo(){
        return subtitulo;
    }
}
```

ListView (V)

- Si en cada element de la llista volem mostrar les dues dades, haurem de crear un layout XML amb l'estructura que desitgem que tinga.
- En aquest cas anem a mostrar el títol i el subtítol mitjançant **TextView**, el primer en negreta i amb una mida una mica més gran. Anomenarem a aquest layout **listitem_titular.xml**

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:orientation="vertical">

<TextView android:id="@+id/LblTitulo"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textStyle="bold"
    android:textSize="20sp" />

<TextView android:id="@+id/LblSubTitulo"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textStyle="normal"
    android:textSize="12sp" />

</LinearLayout>
```

ListView (VI)

- El següent pas serà crear **nostre propi adaptador estenent** de la classe **ArrayAdapter**.

```
public class AdaptadorTitulares extends ArrayAdapter<Titular> {
    private Titular[] datos;

    public AdaptadorTitulares(@NonNull Context context, Titular[] datos) {
        super(context, R.layout.listitem_titular, datos);
        this.datos = datos;
    }

    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
        LayoutInflater inflater = LayoutInflater.from(getContext());
        View item = inflater.inflate(R.layout.listitem_titular,  root: null);
        TextView lblTitulo = (TextView)item.findViewById(R.id.LblTitulo);
        lblTitulo.setText(datos[position].getNombre());
        TextView lblSubtitulo = (TextView)item.findViewById(R.id.LblSubtitulo);
        lblSubtitulo.setText(datos[position].getApellidos());
        return item;
    }
}
```

ListView (VII)

- Analitzant el codi anterior, el primer que trobem és el **constructor**, al qual només li passem el **context** i la matriu de **dades** a mostrar, que en aquest cas és un array d'objectes de tipus **Titular**.
- En aquest constructor simplement cridem al constructor pare, passant-li el Context, l'id del layout que volem utilitzar (listitem_titular) i les dades.
- Posteriorment, **redefinim** el mètode **getView()** encarregat de generar i omplir amb les nostres dades tots els controls necessaris de la interfície gràfica de cada element de la llista.
- El mètode **getView()** s'invocarà cada vegada que vaja a mostrar-se un element de la llista. El primer que ha de fer és "inflar" el layout XML que hem creat. Això consisteix en consultar l'XML del nostre layout, crear i inicialitzar l'estructura d'objectes java equivalent. Per a això, crearem un nou objecte **LayoutInflater** i generarem l'estructura d'objectes mitjançant el seu mètode inflate(id_layout).
- Després d'això, tan sols haurem d'obtenir la referència a cadascuna de les nostres etiquetes i assignar el seu text corresponent segons les dades del nostre array i la posició de l'element actual (paràmetre **position** del mètode getView()).

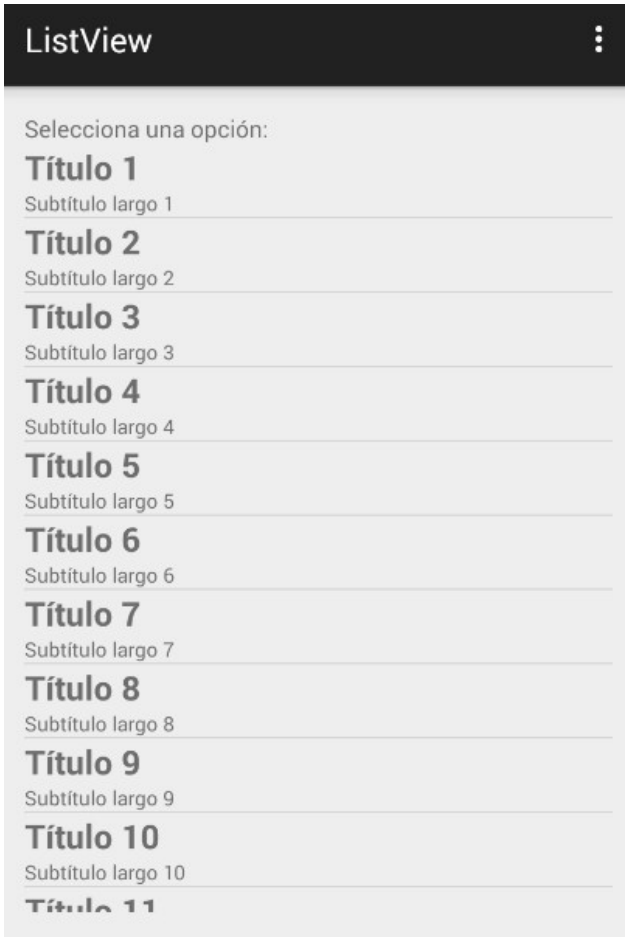
ListView (VIII)

- Una vegada hem definit el comportament del nostre adaptador, només ens queda definir l'**array de dades** en l'Activity, crear l'adaptador i assignar-lo al **ListView** mitjançant el mètode **setAdapter()**.

```
private Titular[] datos =
    new Titular[]{
        new Titular("Título 1", "Subtítulo largo 1"),
        new Titular("Título 2", "Subtítulo largo 2"),
        new Titular("Título 3", "Subtítulo largo 3"),
        new Titular("Título 4", "Subtítulo largo 4"),
        //...
        new Titular("Título 15", "Subtítulo largo 15")};

//...

AdaptadorTitulares adaptador =
        new AdaptadorTitulares(this, datos);

lstOpciones = (ListView)findViewById(R.id.LstOpciones);

lstOpciones.setAdapter(adaptador);
```

ListView (IX)

- Si hem seguit correctament els passos, la nova llista hauria de quedar una cosa així:



ListView (X)

- Finalment comentem una mica els **esdeveniments** d'aquest tipus de controls. Si volem fer qualsevol acció al prémer sobre un element de la llista hem d'implementar l'esdeveniment **onItemClick**. Vegem com amb un exemple:

```
lstOpciones.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> a, View v, int position, long id) {

        //Alternativa 1:
        String opcionSeleccionada =
                ((Titular)a.getItemAtPosition(position)).getTitulo();

        //Alternativa 2:
        //String opcionSeleccionada =
        //       ((TextView)v.findViewById(R.id.LblTitulo))
        //           .getText().toString();

        lblEtiqueta.setText("Opción seleccionada: " + opcionSeleccionada);
    }
});
```

ListView (XI)

- L'esdeveniment **onItemClick** rep 4 paràmetres:
  - Referència al control **llista** que ha rebut el clic (**AdapterView a**)
  - Referència a l'objecte **View** corresponent al **ítem premut** (**View v**)
  - **Posició** l'element premut dins de l'adaptador de la llista (int **position**)
  - **Id** de la fila de l'element premut (long **id**)
- Si vullguerem mostrar el títol de l'opció polsada en l'etiqueta de text superior (lblEtiqueta) tindríem dues possibilitats:
  - Accedir a la vista associada a l'adaptador i a partir d'aquesta, obtenir mitjançant **getItemAtPosition()** l'element que tinga la posició *position*. Això ens tornaria un objecte de tipus **Titular**, de manera que obtindríem el títol invocant al seu mètode **getTitulo()**.
  - Accedir directament a la vista que s'ha premut, que tindria l'estructura definida en el nostre layout personalitzat listitem_titular.xml, i obtenir mitjançant **findViewById()** i **gettext()** el text del control que alberga el camp títol.

ViewHolder (I)

- Podem fer un millor ús de les llistes utilitzant el patró de disseny **viewHolder**, de manera que la **resposta** de la nostra aplicació siga més **àgil** i **reduïsca** el **consum** de **bateria**.
- Prendrem com a base el codi escrit en els apartats on varem explicar el **ListView**, més concretament, la classe **AdaptadorTitulares**.
- Si ens fixem en la definició del mètode **getView()** podem observar que per a cadascún dels elements de la llista que s'ha de mostrar, hem d'inflar el layout XML, independentment de si ja ho havíem mostrat abans o no, ja que Android no "guarda" els elements de la llista que desapareixen de la pantalla en fer scroll.
- Depenent de la mida de la llista i de la complexitat del layout, això pot suposar la creació i destrucció de quantitats ingents d'objectes, el que augmentarà l'ús de la CPU, de la memòria i com a conseqüència de la bateria.

ViewHolder (II)

- Per alleugerir aquest problema, Android ens proposa un mètode que permet **reutilitzar** algun **layout** que ja hem "inflat" amb anterioritat i que ja no ens calga per algun motiu, per exemple perquè l'element corresponent de la llista ha desaparegut de la pantalla en fer scroll.
- D'aquesta manera evitem tota la feina de crear i estructurar tots els objectes associats al layout, de manera que tan sols ens quedaria obtenir la referència a ells mitjançant **findViewById()** i modificar les seves propietats.
- Per a això, farem el següent: sempre que hi haja algun layout que puga ser reutilitzat, aquest el rebrem a través del paràmetre **convertView** del mètode **getView()**. En els casos en què aquest paràmetre no siga null podrem obviar el treball d'inflar el layout.

ViewHolder (III)

- Vegem com quedaria el codi després d'aquesta optimització:

```
public View getView(int position, View convertView, ViewGroup parent)
{
    View item = convertView;

    if(item == null)
    {
        LayoutInflater inflater = context.getLayoutInflater();
        item = inflater.inflate(R.layout.listitem_titular, null);
    }

    TextView lblTitulo = (TextView)item.findViewById(R.id.LblTitulo);
    lblTitulo.setText(datos[position].getTitulo());

    TextView lblSubtitulo = (TextView)item.findViewById(R.id.LblSubTitulo);
    lblSubtitulo.setText(datos[position].getSubtitulo());

    return(item);
}
```

- Si executem ara l'aplicació podrem comprovar que al fer scroll sobre la llista tot segueix funcionant amb normalitat, amb la diferència que li estem estalviant una gran feina a la CPU.

ViewHolder (IV)

- Però podem fer un pas més en el procés d'optimització. Encara hi ha dues invocacions relavitament **costoses** que es segueixen executant en totes les cridades, les **dues invocacions al mètode findViewById()**.
- La **recerca per ID** d'un control determinat dins de l'arbre d'objectes d'un layout també pot ser costosa depenent de la complexitat del propi layout.
- Anem a aprofitar que estem guardant un layout anterior per guardar també la referència als controls que el formen, de manera que no haguem de tornar a buscar-los.
- Aquesta tècnica de guardar el layout i la referència als controls és precisament el que en arquitectura del programari es diu **patró ViewHolder**.
- Per fer-ho anem a **crear una classe** que només va a contenir una referència a cadascun dels controls que hem de manipular del nostre layout, en aquest cas les dues etiquetes de text.

ViewHolder (V)

- Vegem com quedaria la classe:

```
static class ViewHolder {
    TextView titulo;
    TextView subtitulo;
}
```

- La idea serà crear i inicialitzar l'objecte **ViewHolder** la primera vegada que "inflem" un element de la llista i associar-lo a aquest element de manera que puguem recuperar fàcilment després.
- Per a això, en Android tots els controls tenen una propietat anomenada **Tag** que podem assignar-la recuperar-la mitjançant els mètodes **setTag()** i **getTag()** respectivament.
- Quan el paràmetre **convertView** no siga null sabrem que també tindrem disponibles les referències als seus controls a través de la propietat **Tag**.

ViewHolder (VI)

```
public View getView(int position, View convertView, ViewGroup parent)
{
    View item = convertView;
    ViewHolder holder;

    if(item == null)
    {
        LayoutInflater inflater = context.getLayoutInflater();
        item = inflater.inflate(R.layout.listitem_titular, null);

        holder = new ViewHolder();
        holder.titulo = (TextView)item.findViewById(R.id.LblTitulo);
        holder.subtitulo = (TextView)item.findViewById(R.id.LblSubTitulo);

        item.setTag(holder);
    }
    else
    {
        holder = (ViewHolder)item.getTag();
    }

    holder.titulo.setText(datos[position].getTitulo());
    holder.subtitulo.setText(datos[position].getSubtitulo());

    return(item);
}
```

GridView (I)

- El control **GridView** d'Android presenta a l'usuari un conjunt d'opcions seleccionables distribuïdes de forma tabular, o dit d'una altra manera, dividides en **files** i **columnes**.
- Les propietats més importants són:
  - android: numColumns, indica el **nombre de columnes** de la taula o "auto_fit" si volem que siga calculat pel propi sistema operatiu a partir de les següents propietats.
  - android: columnWidth, indica l'**amplada de les columnes** de la taula.
  - android: horizontalSpacing, indica l'**espai horitzontal entre cel·les**.
  - android: verticalSpacing, indica l'**espai vertical entre cel·les**.
  - android: stretchMode, indica **què fer amb l'espai horitzontal sobrant**. Si s'estableix al valor "columnWidth" aquest espai serà absorbit a parts iguals per les columnes de la taula. Si per contra s'estableix a "spacingWidth" serà absorbit a parts iguals pels espais entre cel·les.

GridView (II)

- Per definir un **GridView**:

```
<GridView android:id="@+id/GridOpciones"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:numColumns="auto_fit"
    android:columnWidth="80px"
    android:horizontalSpacing="5dp"
    android:verticalSpacing="10dp"
    android:stretchMode="columnWidth" />
```

- La forma d'assignar dades des de codi Java és igual a la comentada per les llistes desplegables. Creem un array genèric que continga dades de prova, declarem un adaptador de tipus **ArrayAdapter** passant-li en aquest cas un layout genèric (simple_list_item_1) i associem l'adaptador al control **GridView** mitjançant el mètode **setAdapter()**.
- Per defecte, les dades de l'array s'afegiran al **GridView** **ordenades per files**.

GridView (III)

```
private String[] datos = new String[50];
//...
for(int i=1; i<=50; i++)
        datos[i-1] = "Dato " + i;

ArrayAdapter<String> adaptador =
        new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, datos)

grdOpciones = (GridView)findViewById(R.id.GridOpciones);

grdOpciones.setAdapter(adaptador);
```

GridView (IV)

- Quant als esdeveniments disponibles, el més interessant és al seleccionar una cel·la: **onItemClick**.
- Podem capturar de la mateixa manera que ho fèiem amb els controls **Spinner** i **ListView**.

```
grdOpciones.setOnItemClickListener(
    new AdapterView.OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent,
            android.view.View v, int position, long id) {
                lblMensaje.setText("Opción seleccionada: "
                    + parent.getItemAtPosition(position));
            }
    });
```

Introducció al RecyclerView

- Desde Android 5.0, Google incorpora al SDK d'Android un nou component que millora els clàssics **ListView** i **GridView**.
- En aplicacions noves es recomana utilitzar **RecyclerView** ja que aporta flexibilitat per a suplir la funcionalitat de tots dos controls i anar fins i tot més enllà.
- Ens permetrà mostrar grans col·leccions de dades, però ho fa de forma diferent.
- **RecyclerView** no farà “quasi res” per si mateix, sinó que es sustentarà sobre altres **components complementaris** per a determinar com accedir a les dades i com mostrar-les.

Components de RecyclerView

- Els **components complementaris** sobre els quals treballa un RecyclerView són:
  - **RecyclerView.Adapter**
  - **RecyclerView.ViewHolder**
  - **LayoutManager**
  - **ItemDecoration**
  - **ItemAnimator**
- D'igual forma que hem fet amb els components anteriors, un RecyclerView es recolzarà també en un adaptador per a treballar amb les nostres dades, en aquest cas un adaptador que herete de la classe **RecyclerView.Adapter**. La peculiaritat en aquesta ocasió és que aquest tipus d'adaptador **ens "obligarà"** en certa mesura **a utilitzar el patró ViewHolder**, i d'ací la necessitat del segon component de la llista anterior, **RecyclerView.ViewHolder**.

LayoutManager

- Quan utilitzem un **ListView** sabem que les dades es representaran de forma lineal amb la possibilitat de fer scroll en un sentit o un altre, i en el cas de triar un **GridView** es representaran de forma tabular.
- Una vista de tipus **RecyclerView** per contra no determina per si sola la forma en què es mostraran en pantalla els elements de la nostra col·lecció, sinó que delegarà eixa tasca a un altre component anomenat **LayoutManager**, que també haurem de crear i associar al **RecyclerView** per al seu correcte funcionament.
- Per sort, el SDK incorpora de sèrie tres **LayoutManager** per a les tres representacions més habituals: **llista** vertical o horitzontal (**LinearLayoutManager**), **taula** tradicional (**GridLayoutManager**) i **taula apilada** o de cel·les no alineades (**StaggeredGridLayoutManager**).
- Sempre que optem per alguna d'aquestes distribucions d'elements no haurem de crear el nostre propi **LayoutManager** personalitzat, encara que per descomptat res ens impedeix fer-ho, i ací un dels punts forts del nou component: la seua **flexibilitat**.

ItemDecoration i ItemAnimator

- **ItemDecoration** i **ItemAnimator** s'encarregaran de definir com es representaran alguns aspectes visuals concrets de la nostra col·lecció de dades (més enllà de la distribució definida pel **LayoutManager**).
- Per exemple marcadors o separadors d'elements, i de com s'animaran els elements en realitzar-se determinades accions sobre la col·lecció, per exemple en afegir o eliminar elements.
- No sempre serà obligatori implementar tots aquests components per a fer ús d'un **RecyclerView**. El més habitual serà implementar l'Adapter i el **ViewHolder**, utilitzar algun dels **LayoutManager predefinits**, i només en cas de necessitat crear els **ItemDecoration** i **ItemAnimator** necessaris per a donar un toc de personalització especial a la nostra aplicació.

Llibreries de suport

- Perquè la nostra aplicació suporte aquest nou component ho farem, com és habitual, mitjançant les **llibreries de suport** d'Android. D'aquesta forma podem donar compatibilitat a versions d'Android que no ho suporten de forma nativa.
- Per a això editem l'arxiu **build.gradle** del nostre mòdul (app) i el modifiquem de la següent forma:

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.0.2'
    implementation 'androidx.recyclerview:recyclerview:1.0.0'
}
```

Crear el Layout

- Una vegada incloses les llibreries de suport per a **RecyclerView**, ja podem afegir al layout de la nostra **Activity** el component:

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/rvListado"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

- El següent pas serà escriure el nostre adaptador que haurà d'estendre a **RecyclerView.Adapter** i sobreescriure els següents mètodes:
  - **OnCreateViewHolder()**. Encarregat de crear els nous objectes **ViewHolder** necessaris per als elements de la col·lecció.
  - **OnBindViewHolder()**. Encarregat d'actualitzar les dades d'un **ViewHolder** ja existent.
  - **getItemCount()**. Indica el nombre d'elements de la col·lecció de dades.

Crear el nostre adaptador

- Per a veure un exemple d'ús complet de **RecyclerView**, utilitzarem la mateixa classe que ja utilitzem per al **ListView** (**Titular**).
- L'adaptador **Recycler.Adapter** ens obliga a fer ús del patró **ViewHolder**. Per tant, hem de definir primer el **ViewHolder** necessari. En aquest cas, el farem definint-lo com una classe interna del nostre adaptador, estenent la classe **RecyclerView.ViewHolder**.
- Inclourem com a atributs les referències als controls del layout d'un element llista (en el nostre cas dos **TextView**) i els inicialitzarem en el constructor.
- Per comoditat afegirem un mètode auxiliar, que anomenarem **bindTitular()**, que s'encarregarà d'assignar els continguts dels dos quadres de text a partir d'un objecte **Titular**.

AdaptadorTitulares.java (I)

```
public class AdaptadorTitulares extends RecyclerView.Adapter<AdaptadorTitulares.TitularesViewHolder>
{
   //…
   public class TitularesViewHolder extends RecyclerView.ViewHolder {
   private TextView tvTitulo;
   private TextView tvSubtitulo;

   public TitularesViewHolder(@NonNull View itemView, ITitularListener listener) {
      super(itemView);
      tvTitulo = itemView.findViewById(R.id.tvTitulo);
      tvSubtitulo = itemView.findViewById(R.id.tvSubtitulo);
   }

   public void bindTitular(Titular titular) {
      tvTitulo.setText(titular.getTitulo());
      tvSubtitulo.setText(titular.getSubtitulo());
   }
   //…
}
```

Sobreescriure mètodes de l'adaptador

- Una vegada acabat el nostre **ViewHolder**, ja podem seguir amb la implementació de l'adaptador sobreescrivint els mètodes esmentats anteriorment.
- En el mètode **onCreateViewHolder()** ens limitarem a unflar una vista a partir del layout corresponent als elements de la llista (listitem_titular), crear i retornar un nou **ViewHolder** cridant al constructor de la classe **TitularesViewHolder** passant-li aquesta vista com a paràmetre.
- En el mètode **onBindViewHolder()** només haurem de recuperar l'objecte **Titular** corresponent a la **posició** rebuda com a paràmetre i assignar les seues dades sobre el **ViewHolder** rebut també com a paràmetre.
- En el mètode **getItemCount()** l'única cosa que hem de fer és retornar **la grandària** de la llista d'objectes **Titular**.

AdaptadorTitulares.java (II)

```
public class AdaptadorTitulares extends
RecyclerView.Adapter<AdaptadorTitulares.TitularesViewHolder> {
   private Titular[] datos;

   public AdaptadorTitulares(Titular[] datos) {
      this.datos = datos;
   }
   @NonNull
   @Override
   public TitularesViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType)
{
      View itemView =
LayoutInflater.from(parent.getContext()).inflate(R.layout.item_titular, parent, false);
      return new TitularesViewHolder(itemView);
   }
   @Override
   public void onBindViewHolder(@NonNull TitularesViewHolder holder, int position) {
      Titular titular = datos[position];
      holder.bindTitular(titular);
   }
   @Override
   public int getItemCount() {
      return datos.length;
   }
   //…
}
```

Assignar el nostre adaptador

- Amb això tindríem finalitzat l'adaptador i només ens quedaria crear-ho en la nostra **Activity** i assignar-ho al **RecyclerView** mitjançant el mètode **setAdapter()**.

```
public class MainActivity extends AppCompatActivity {
   private final int NDATOS = 50;
   private Titular[] datos;
   private RecyclerView rvListado;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      inicializarDatos();
      rvListado = findViewById(R.id.rvListado);
      rvListado.setHasFixedSize(true);
      rvListado.setAdapter(new AdaptadorTitulares(datos));
      //...
   }
   public void inicializarDatos() {
      StringBuilder sbTitulo = new StringBuilder();
      StringBuilder sbSubtitulo = new StringBuilder();
      datos = new Titular[NDATOS];
      for(int i = 0; i < NDATOS; i++) {
         sbTitulo.setLength(0);
         sbSubtitulo.setLength(0);
         sbTitulo.append("Título ").append(i+1);
         sbSubtitulo.append("Subtítulo largo ").append(i+1);
         datos[i] = new Titular(sbTitulo.toString(), sbSubtitulo.toString());
      }
   }
```

Consideracions

- Hem aprofitat el mètode **onCreate()** per a inicialitzar la llista de dades d'exemple amb 50 titulars.
- Després d'obtenir la referència al **RecyclerView** hem inclòs una anomenada al mètode **setHasFixedSize()**. Encara que això no és obligatori, sí que és convenient fer-ho quan tinguem certesa que la grandària del nostre **RecyclerView** no va a canviar, ja que permetrà aplicar diverses **optimitzacions** sobre el control.
- El següent pas serà associar al **RecyclerView** un **LayoutManager** per a determinar la forma en la qual es distribuiran les dades en pantalla.
- Com ja hem comentat, Android proporciona alguns LayoutManager predefinits per a les tres principals formes de mostrar dades, llistes (**LinearLayoutManager**), taules (**GridLayoutManager**) i taules apilades (**StaggeredGridLayoutManager**).

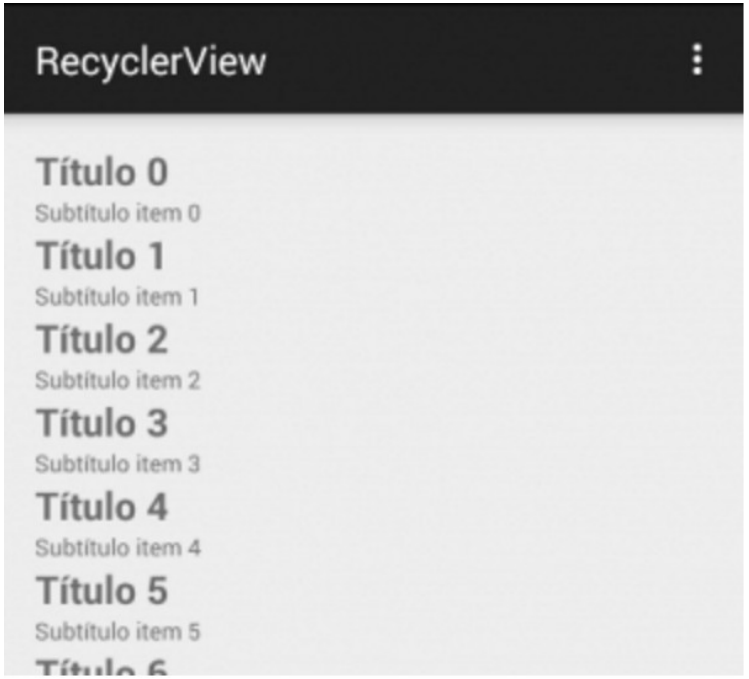
Assignar el LayoutManager

- En aquest exemple mostrarem les dades en forma **de llista** amb desplaçament **vertical**, per tant crearem una instància de tipus **LinearLayoutManager** indicant en el constructor l'orientació (**LinearLayoutManger**.VERTICAL o **LinearLayoutManager**.HORITZONTAL) i ho assignarem al **RecyclerView** mitjançant el mètode **setLayoutManager()**.

```
//...
rvListado.setAdapter(new AdaptadorTitulares(datos, this));
rvListado.setLayoutManager(new LinearLayoutManager(this,
LinearLayoutManager.VERTICAL, false));
//...
```

RecyclerView amb LinearLayoutManager

- Arribats a aquest punt, ja seria possible executar l'aplicació per a veure com queden les dades en pantalla.



RecyclerView amb GridLayoutManager

- Ara si vullguerem canviar la forma en la que es mostren les dades, per exemple com una taula, només hauríem de canviar l'assignació del **LayoutManager** i utilitzar un **GridLayoutManager**, al qual li passaríem com a paràmetre el **nombre de columnes** que volem mostrar.

```
//...
rvListado.setAdapter(new AdaptadorTitulares(datos, this));
rvListado.setLayoutManager(new GridLayoutManager(this, 3));
//...
```

Maneig d'esdeveniments (I)

- La forma de tractar els esdeveniments en **un RecyclerView**, també difereix respecte a **ListView** ja que **no existeixen** esdeveniments del tipus **onItemClick()**.
- Una vegada més, **RecyclerView** **delega** els esdeveniments en altres components, és aquest cas **la pròpia vista** que conforma cada element de la col·lecció.
- Per tant, per a tractar esdeveniments aprofitarem la creació de cada **ViewHolder** per a assignar a la seua vista associada l'esdeveniment **onClick()**.
- A més, per a poder manejar esdeveniments des de fora de l'adaptador, crearem una interfície i inclourem un **listener** membre de dita interfície **com a atribut de l'adaptador** i un mètode per a poder assignar-lo des de fora **setOnClickListener()**.
- D'aquesta forma, en el constructor del **ViewHolder** s'associarà l'esdeveniment a la vista, i finalment implementarà l'esdeveniment **onClick()**, que es limitarà a llançar el mateix esdeveniment sobre el listener extern.

Maneig d'esdeveniments (II)

- En primer lloc creem la interfície **ITitularListener** per a tractar l'esdeveniment:

```
public interface ITitularListener {

    void onTitularSeleccionado(int position);

}
```

Maneig d'esdeveniments (III)

```
public class AdaptadorTitulares extends RecyclerView.Adapter<AdaptadorTitulares.TitularesViewHolder> {
    private Titular[] datos;
    private ITitularListener listener;
    public AdaptadorTitulares(Titular[] datos, ITitularListener listener) {
        this.datos = datos;
        this.listener = listener;
    }
    @NonNull
    @Override
    public TitularesViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View itemView = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_titular, parent, false);
        return new TitularesViewHolder(itemView, listener);
    }
    //...
    public class TitularesViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
        private TextView tvTitulo;
        private TextView tvSubtitulo;
        private ITitularListener listener;
        public TitularesViewHolder(@NonNull View itemView, ITitularListener listener) {
            super(itemView);
            tvTitulo = itemView.findViewById(R.id.tvTitulo);
            tvSubtitulo = itemView.findViewById(R.id.tvSubtitulo);
            this.listener = listener;
            itemView.setOnClickListener(this);
        }
        //...
        @Override
        public void onClick(View v) {
            if(listener != null) {
                listener.onTitularSeleccionado(getAdapterPosition());
            }
        }
    }
}
```

Maneig d'esdeveniments (IV)

- Com hem vist, el nostre **ViewHolder** **implementarà** la **interfície** **OnClickListener**, declararà un **listener** com a atribut (que rebrà com a paràmetre en la construcció del ViewHolder), en el constructor del **ViewHolder** associarà l'esdeveniment a la vista, i finalment implementarà l'esdeveniment **onClick()**, que es limitarà a llançar el mateix esdeveniment sobre el listener extern.
- L'Activity implementará la interfície **ITitularListener** i
- El constructor de l'adaptador rebrà com a paràmetres les dades i el listener (**ITitularListener**) que ens servirà per a associar el listener "real" al nostre adaptador en el moment de crear-lo, així des de la nostra **Activity** implementarem la interfície **ITitularListener** i podrem assignar-lo a l'adaptador.
- Vegem com quedaria l'Activity amb aquest canvi.

Maneig d'esdeveniments (V)

```
public class MainActivity extends AppCompatActivity implements ITitularListener{
//…
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   inicializarDatos();
   rvListado = findViewById(R.id.rvListado);
   rvListado.setAdapter(new AdaptadorTitulares(datos, this));
   rvListado.setLayoutManager(new LinearLayoutManager(this,
LinearLayoutManager.VERTICAL, false));
}
@Override
public void onTitularSeleccionado(int position) {
   Toast.makeText(this, datos[position].getTitulo(), Toast.LENGTH_SHORT).show();
}
```