



Lezione 9



Programmazione Android



- Ancora sulla UI
 - ListView e data adapter
 - Altri usi di adapter
 - RecyclerView



ListView e Data Adapter

ListView



- Uno dei componenti più comunemente usati in una GUI su Android è la lista scrollabile
- Ogni elemento è a sua volta una View
 - Quindi la ListView è un ViewGroup pur non essendo un layout
 - Ci sono molti casi del genere
 - Gallery, CalendarView, DatePicker...





ListView statiche e dinamiche



- Se le view contenute nella ListView sono statiche, abbiamo già tutti gli ingredienti necessari
 - Si definisce un array di risorse in res/values
 - Si imposta l'attributo **android:entries** del tag <ListView> con un riferimento alla risorsa array
- Approccio con vantaggi e svantaggi
 - Comodo quando i valori vanno configurati
 - Per lingua, nazione, carrier, ecc.
 - Limitato sui dati visualizzabili
 - Solo statici, solo testi semplici

ListView statiche – esempio



- Res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<ListView xmlns:android="http://schemas.android.com/apk/res/android"  
  android:layout_width="fill_parent"  
  android:layout_height="fill_parent"  
  android:entries="@array/lully"/>
```

Riferimento agli item

- Activity (solita solfa)

```
public class ListViewTestActivity extends Activity {
```

```
  @Override
```

```
  public void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.main);
```

```
  }
```

```
}
```

Monta il layout

ListView statiche – esempio



- Res/values/arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>
```

Array riferito

```
<string-array name="lully">  
  <item>1. Jubilate Deo (29 agosto 1660)</item>  
  <item>2. Miserere (23? marzo 1663)</item>  
  <item>3. Benedictus (1663 o 1664)</item>  
  <item>4. O lachrymae (1664?)</item>  
  <item>5. Plaude laetare Gallia (24 marzo 1668)</item>  
  <item>6. Te Deum (9 settembre 1677)</item>  
  <item>7. De profundis (maggio 1683)</item>  
  <item>8. Dies irae (1 settembre 1683)</item>  
  <item>9. Quare fremuerunt (19 aprile 1685)</item>  
  <item>10. Domine salvum fac regem (1685?)</item>  
  <item>11. Notus in Judea (1685 o 1686)</item>  
  <item>12. Exaudiat Te Domine (1687)</item>  
</string-array>
```

```
</resources>
```





ListView dinamiche



- Più spesso, i dati da visualizzare in una ListView sono dinamici
 - Generati dal programma
 - Estratti da un database
 - Ottenuti da un servizio web
 - ecc.
- In questi casi, si accoppia una ListView a un **Adapter**
 - **ArrayAdapter, CursorAdapter, ListAdapter, ...**



Responsabilità di un Adapter



- Un Adapter ha diversi compiti
 - Ottenere i dati “grezzi” per una entry
 - Costruire una View che rappresenti graficamente i dati “grezzi”
 - Fornire la View al ViewGroup a cui l'Adapter è associato
 - Notificare gli **Observer** quando i dati cambiano
 - Alcuni altri compiti “amministrativi”
- È sempre possibile scrivere propri Adapter custom



ArrayAdapter

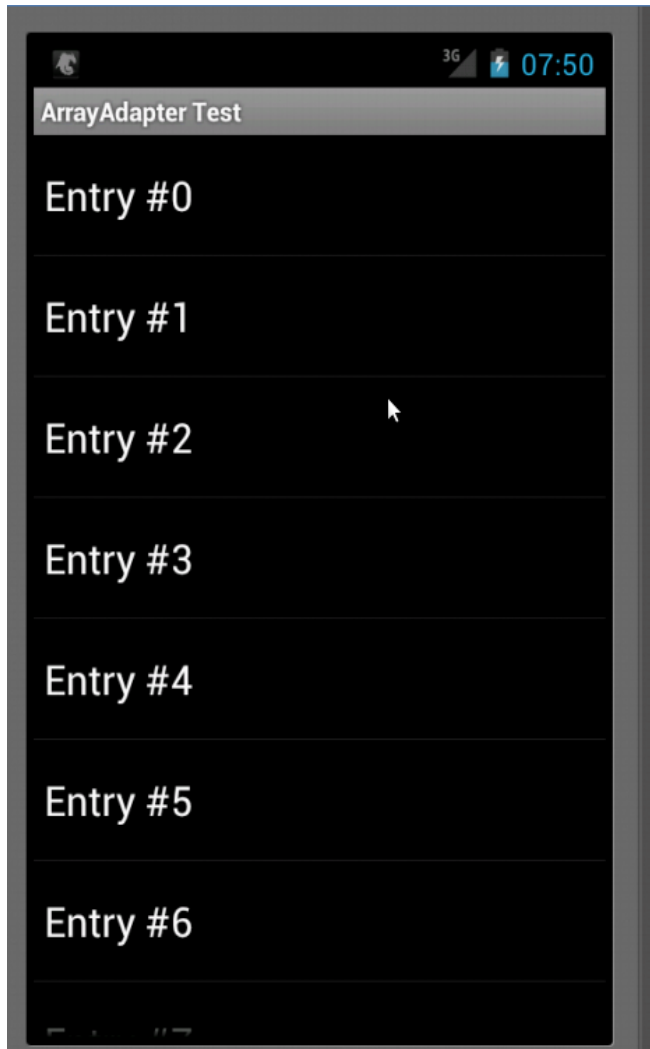
- Le diverse sottoclassi di Adapter traducono diversi formati di dati “grezzi”
- **ArrayAdapter<T>**: un array (Java) di elementi di tipo T
- Vari costruttori, con parametri:
 - Context (per accedere alle risorse)
 - ID del layout (XML) da utilizzare
 - ID della TextView dentro il layout da popolare con i dati
 - T[] o List<T> contenente i dati “grezzi”

ArrayAdapter – esempio

```
public class ArrayAdapterTest extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        //NON usiamo setContentView(R.layout.main);  
        String[] a = new String[20];  
        for (int i=0; i<20; i++) a[i]="Entry #" +i;  
        ListView lv = new ListView(this);  
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(  
            ArrayAdapterTest.this,  
            android.R.layout.simple_list_item_1,  
            a);  
        lv.setAdapter(adapter);  
        setContentView(lv);  
    }  
}
```

Per cambiare: questa volta creiamo l'intera Activity in maniera dinamica, tutto a codice, senza usare alcun file XML.

ArrayAdapter – esempio



- Android definisce fra le risorse di sistema alcuni layout comuni per le view interne di una lista
- `android.R.layout. ...`
 - `simple_list_item_1`
 - `simple_list_item_2`
 - `simple_list_item_checked`
 - ...
- Corrispondono a layout XML

Altri Adapter

- Vedremo altre forme di Adapter più avanti
 - CursorAdapter
 - Adatta i risultati di una query SQL
 - ResourceCursorAdapter
 - Adatta un array di risorse XML come se fosse un database
 - SimpleCursorAdapter
 - Usa i risultati di una query SQL come identificatori per accedere a testi e immagini fra le risorse XML
 - SimpleAdapter
 - Usa una `ArrayList<Map>`, una riga per entry, una chiave nella Map per ogni campo della riga (stringhe, booleani, immagini)

Gestione dell'input

- Oltre a visualizzare dati (con scroll), le ListView sono spesso usate per consentire all'utente delle scelte
 - Attivare un elemento da una lista (azione → button)
 - Scegliere un elemento da una lista (opzione → radio)
 - Selezionare zero o più elementi da una lista (opzione → check)
 - Espandere o collassare sezioni di una lista gerarchica (navigazione → tree)



Gestione dell'input



- Per riconoscere il click su un elemento;
 - Si implementa l'interfaccia `OnItemClickListener`
 - Lo si associa alla lista con `setOnItemClickListener()`
 - Si aspetta che venga chiamato `onItemClick()`
- Vale quanto detto a suo tempo su `onClick!`
 - Efficienza, evitare le **new**, esecuzione nel thread UI

Gestione dell'input – esempio



```
public class ArrayAdapterTest extends Activity implements OnItemClickListener  
{
```

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);
```

Facciamo implementare il listener all'Activity

```
    ...  
    lv.setAdapter(adapter);  
    lv.setOnItemClickListener(this);
```

Impostiamo noi stessi come listener alla ListView

```
    ...  
}
```

```
public void onItemClick(AdapterView<?> parent, View view, int pos, long id) {
```

```
    CharSequence s=((TextView)view).getText();  
    Log.d("AAT",s.toString());
```

Chiamato al click dell'utente

- **parent** è la ListView
- **view** è l'item selezionato
- **pos** è la posizione (in ordine)
- **id** è l'ID dell'item selezionato



Gestione dell'input - multi



- Possibilità di selezionare zero, uno o più elementi
- Layout di un elemento
 - Il sistema fornisce `android.R.layout.simple_list_item_multiple_choice`, ma è sempre possibile definire il proprio layout
- Opzioni della ListView
 - `<ListView ... android:choiceMode="multipleChoice" ... />`
- Recupero delle selezioni
 - SparseBooleanArray `getCheckedItemPositions()`
 - Long[] `getCheckedItemIds()`
 - ecc.

Esempio di lista con multiselezione



```

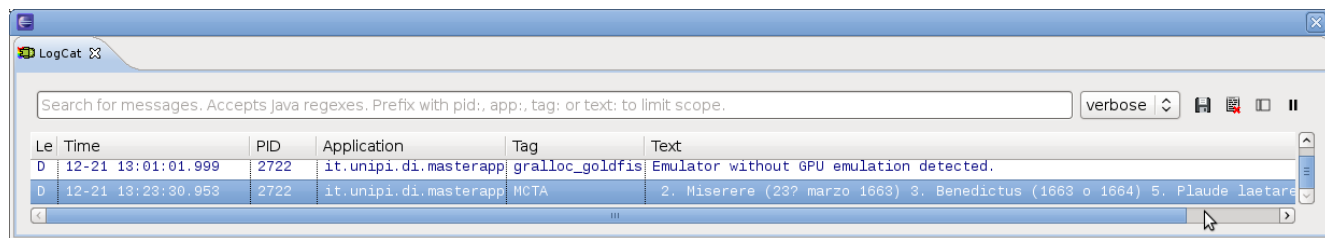
public class MultiCheckTestActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.multi);
        final ListView lv = (ListView) findViewById(R.id.multilist);
        lv.setAdapter(new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_multiple_choice,
            getResources().getStringArray(R.array.lully)));

        Button b = (Button) findViewById(R.id.button1);
        b.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                SparseBooleanArray checked = lv.getCheckedItemPositions();
                int n = checked.size();
                StringBuffer sb = new StringBuffer();
                for (int i = 0; i < n; i++) {
                    sb.append(" "); sb.append(lv.getItemAtPosition(checked.keyAt(i)));
                }
                Log.d("MCTA", sb.toString());
            }
        });
    }
}

```



Esempio di lista con multiselezione



- È anche possibile...
 - Leggere o impostare lo stato di una singola entry
 - Usare come layout `simple_list_item_single_choice` (per radio button)
 - Innestare header o footer alla lista

ListActivity

- Android fornisce una sottoclasse di Activity specializzata per contenere ListView
 - Il layout di default contiene due view:
 - La ListView, con id “@android:id/list” (= “list”)
 - Opzionalmente, una view per il caso di lista vuota, con id “@android:id/empty”
 - È anche possibile usare setContentView() per sostituire un proprio layout a quello di default
 - Il proprio layout deve però contenere una ListView “list” e opzionalmente la view “empty”



ListActivity – esempio



```
public class ListActivityTest extends ListActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState){  
        super.onCreate(savedInstanceState);  
  
        Cursor cur = this.getContentResolver().query(People.CONTENT_URI,  
                                                    null, null, null, null);  
  
        startManagingCursor(cur);  
  
        ListAdapter adapter = new SimpleCursorAdapter(  
            this,  
            android.R.layout.two_line_list_item,  
            cur,  
            new String[] {People.NAME, People.PRIMARY_EMAIL_ID},  
            new int[] {android.R.id.text1, android.R.id.text2});  
  
        setListAdapter(adapter);  
    }  
}
```



Altre viste con Adapter

Gallery



- L'idea generale di usare un **Adapter** per decidere quali **view** visualizzare dentro un **ViewGroup** è usata in altri widget
- **Gallery** mostra una “striscia” orizzontale di view
 - Ciascuna view proviene da un Adapter
- Altri: Spinner, Flipper, ...

Gallery – Esempio



- Esempio di **Gallery**

- Mostriamo una striscia di immagini
- Il widget gestisce automaticamente lo scorrimento
- Un doppio tap “seleziona” un elemento

- Esempio di **Adapter custom**

- Recuperiamo le immagini da rete
- Creiamo “al volo” la vista per mostrare ogni immagine



Gallery – Esempio il layout



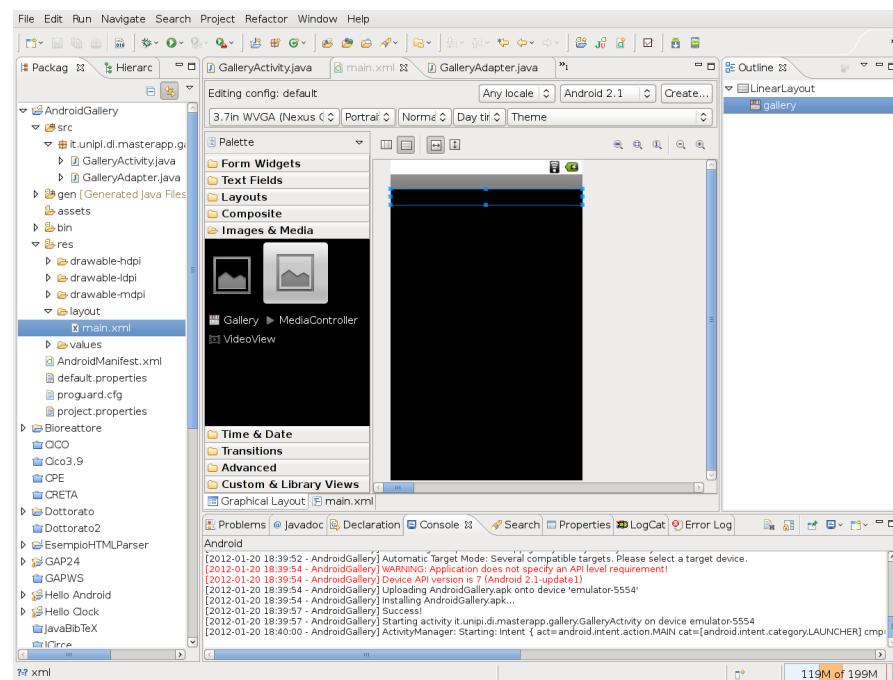
```
<?xml version="1.0" encoding="utf-8"?>
```

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

```
<Gallery
```

```
    android:id="@+id/gallery"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center_vertical"  
    android:gravity="center"  
    android:spacing="8dp" />
```

```
</LinearLayout>
```





Gallery – Esempio

L'Activity



```
public class GalleryActivity extends Activity {  
    private Gallery gallery;  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        gallery=(Gallery) findViewById(R.id.gallery);  
        gallery.setAdapter(new GalleryAdapter(this));  
    }  
    @Override  
    protected void onResume() {  
        super.onResume();  
        gallery.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
            @Override  
            public void onItemClick(AdapterView<?> parent, View view, int pos, long id) {  
                Toast.makeText(getApplicationContext(), "Immagine "+pos, Toast.LENGTH_SHORT ).show();  
            }  
        });  
    }  
}
```

Recuperiamo la Gallery
e la colleghiamo al
nostro Adapter custom

Impostiamo un semplice
OnItemClickListener
(vedremo dopo Toast)



Gallery – Esempio

L'Adapter custom



```
public class GalleryAdapter extends BaseAdapter {
```

```
    int galleryItem;
```

```
    private Context context;
```

```
    public GalleryAdapter(Context context) {  
        super();  
        this.context = context;  
    }
```

Salviamo il Context per usarlo dopo
(attenzione: questo impedisce di fare garbage
collection di **un sacco** di oggetti, finché il nostro
Adapter è in memoria. Non raccomandato!)

```
@Override
```

```
public int getCount() { // Quanti oggetti abbiamo nella nostra lista  
    return 18;  
}
```

Brutale!

```
@Override
```

```
public Object getItem(int position) { // Restituisce l'oggetto alla posizione indicata  
    return urlFor(position);  
}
```

```
@Override
```

```
public long getItemId(int position) { // Restituisce l'ID dell'oggetto alla posizione indicata  
    return position;  
}
```

...



Gallery – Esempio

L'Adapter custom



@Override

```
public View getView(int position, View convertView, ViewGroup parent) {  
    ImageView imageView = new ImageView(context);  
    imageView.setImageDrawable(loadImageFromURL(urlFor(position));  
    imageView.setLayoutParams(new Gallery.LayoutParams(150, 150));  
    imageView.setScaleType(ImageView.ScaleType.CENTER_INSIDE);  
    return imageView;  
}
```

Allochiamo una nuova ImageView per ogni elemento (non consigliabile: vedi dopo)

```
private String urlFor(int position) {  
    return "http://masterapp.di.unipi.it/img/slideshow/a" + (position + 1);  
}
```

```
private Drawable loadImageFromURL(String url) {  
    try {  
        InputStream is = (InputStream) new URL(url).getContent();  
        Drawable d = Drawable.createFromStream(is, "From "+url);  
        return d;  
    } catch (Exception e) {  
        System.out.println(e);  
        return null;  
    }  
}
```

Privati, di utilità

getView()



```
public View getView(int position, View convertView, ViewGroup  
parent)
```

- Deve restituire una View che rappresenta l'oggetto in posizione *position*
- La View verrà inserita come figlia di *parent*
- **Se possibile**, deve modificare *convertView* in modo che essa rappresenti l'oggetto, e restituirla
- **Altrimenti** (meno efficiente), può allocare e restituire una nuova View



RecyclerView

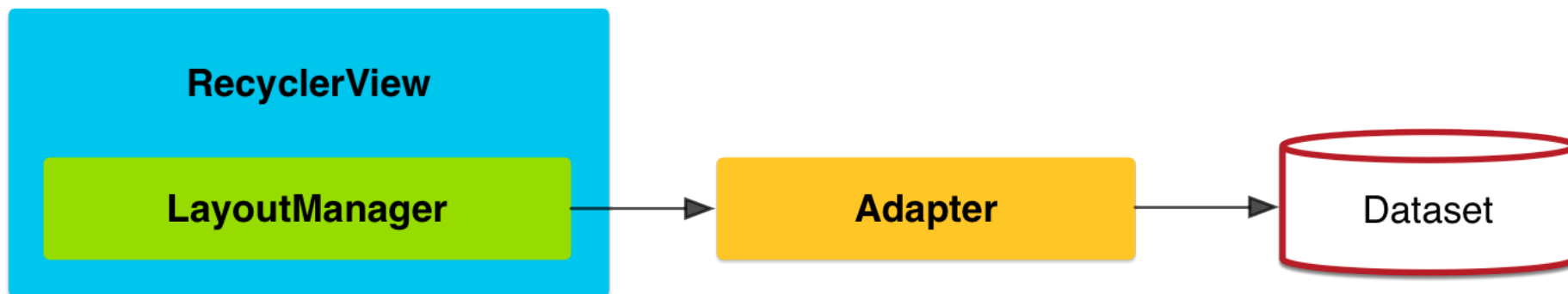


ListView e il riciclo

- **ListView** implementa un comportamento utile in generale
 - Adattamento delle View con dati dinamici
 - Riciclo delle View per evitare **new** e risparmiare memoria
- Però questo comportamento è strettamente accoppiato con la gestione a lista
 - Visualizzazione con `LinearLayout` verticale
 - Scroll up & down
- **RecyclerView** separa queste funzioni

RecyclerView

- RecyclerView fornisce
 - Un **LayoutManager** per decidere come disporre gli elementi
 - Un **Adapter** per recuperare i dati da mostrare
 - Operazioni di **binding** per inserire i dati nelle viste
 - **Decorazioni** per evidenziare una vista
 - **Animazioni** per l'aggiunta e la rimozione di elementi





RecyclerView LayoutManager



- RecyclerView fornisce tre LayoutManager pre-implementati:
 - LinearLayoutManager
 - GridLayoutManager
 - StaggeredGridLayoutManager
- Potete implementare altri LayoutManager scrivendo i vostri
 - Si eredita da RecyclerView.LayoutManager
 - Moltissimi metodi, ma quelli importanti sono
 - onLayoutChildren(), onMeasure(), generateDefaultLayoutParams()



RecyclerView Adapter



- Come nel caso precedente, si estende **RecyclerView.Adapter<VH extends RecyclerView.ViewHolder>**
 - Il **ViewHolder** è un contenitore per le View “cachate”
 - Si può usare l'implementazione di default o darne una propria
- I metodi da implementare sono i soliti
 - getItemCount(), getItemId(), ...
- Ma il binding vero e proprio è fatto con il VH, non si restituisce direttamente una View
 - Quindi, niente getView()



RecyclerView Binding



- Il processo di binding consiste nel modificare una View in modo da mostrare i dati corrispondenti a un dato indice
- Effettuato da due metodi dell'Adapter
 - **onCreateViewHolder(ViewGroup parent, int tipovista)**
 - chiamato quando la RecyclerView ha bisogno di creare un nuovo VH da inserire nel parent dato
 - **onBindViewHolder(VH holder, int indice)**
 - chiamato quando la RecyclerView vuole inserire i dati di indice dato nel VH dato



RecyclerView

Decorazioni e Animazioni



- Opzionalmente, la RecyclerView offre la possibilità di **decorare** una vista
 - Utile per aggiungere bordi o cambiare l'elevazione 3D per indicare una selezione
 - Si implementa un RecyclerView.ItemDecoration
 - Il suo metodo onDraw() può disegnare “sopra” la vista normale
- Sono anche previste **animazioni** per gli effetti di inserimento e cancellazione di elementi dalla lista
 - Non li esploriamo: fanno parte del framework di animazione che vedremo in futuro