Lezioni 7 e 8

Programmazione Android

- Ancora sulla UI
  - Scrivere proprie View
  - ListView e data adapter
  - Notifiche all'utente
    - Toast
    - Dialog
    - Notification bar
  - Action Bar
  - Stili e temi
- Laboratorio

Scrivere una propria View

Scrivere una propria View

- Gli elementi dell'interfaccia utente sono tutti sottoclassi di **View**
  - View, ViewGroup, Layout, Widget
  - Organizzati in un layout tramite file XML
- Per realizzare Widget custom, è sufficiente estendere View o una delle sue sottoclassi
  - **onMeasure()** – chiamato per negoziare le dimensioni
  - **onDraw()** – chiamato per disegnare l'effettiva UI
  - OnKeyDown(), onTouchEvent() & co. – chiamati per gestire l'input

Negoziare le dimensioni

- onMeasure(int widthSpec, int heightSpec)
  - I parametri passati sono i **requisiti** che il contenitore vuole investigare
  - onMeasure() **deve** fornire la risposta chiamando il metodo setMeasuredDimension(int width, int height)
    - In questo modo, si evita ogni allocazione di oggetti!
- Tipicamente, si chiama super.onMeasure() per lasciar fare alle superclasse, e poi si “aggiusta” il risultato in base alle proprie necessità
  - La superclasse chiamerà setMeasuredDimension()

Specifiche di dimensione

- Gli argomenti interi passati a onMeasure() codificano un **modo** e una **dimensione**
  - Nell'implementazione attuale, il modo è codificato dai due bit alti dell'intero, la dimensione dai rimanenti
  - Meglio però usare getMode() e getSize() di View.MeasureSpec per estrarre i valori
- Modo (costanti definite in View.MeasureSpec)
  - **UNSPECIFIED** – il contenitore non impone restrizioni
  - **EXACTLY** – il contenitore impone esattamente la dimensione data
  - **AT_MOST** – il contenitore impone un massimo

Specifiche di dimensione

- La nostra sottoclasse dovrà rispondere all'invocazione di onMeasure() specificando la sua dimensione “preferita”, nel rispetto dei vincoli
  - In qualche caso, possiamo anche ignorare i vincoli – in genere il contenitore effettuerà clipping
  - Di solito, la nostra view vorrà anche rispettare la dimensione minima indicata dai metodi
    - getSuggestedMinimumHeight() e
    - getSuggestedMinimumWidth()

Specifiche di dimensione

- L'implementazione di default (in View)

```
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec)
{
  setMeasuredDimension(
    getDefaultSize(getSuggestedMinimumWidth(), widthMeasureSpec),
    getDefaultSize(getSuggestedMinimumHeight(), heightMeasureSpec) );
}
```

- In pratica: per default è 100x100, ma se lo spazio è “elastico”, si allarga fino a occuparlo tutto

```
public static int getDefaultSize(int size, int measureSpec) {
    int result = size;
    int specMode = MeasureSpec.getMode(measureSpec);
    int specSize =  MeasureSpec.getSize(measureSpec);

    switch (specMode) {
    case MeasureSpec.UNSPECIFIED:
        result = size;
        break;
    case MeasureSpec.AT_MOST:
    case MeasureSpec.EXACTLY:
        result = specSize;
        break;
    }
    return result;
}
```

Disegnare il proprio aspetto

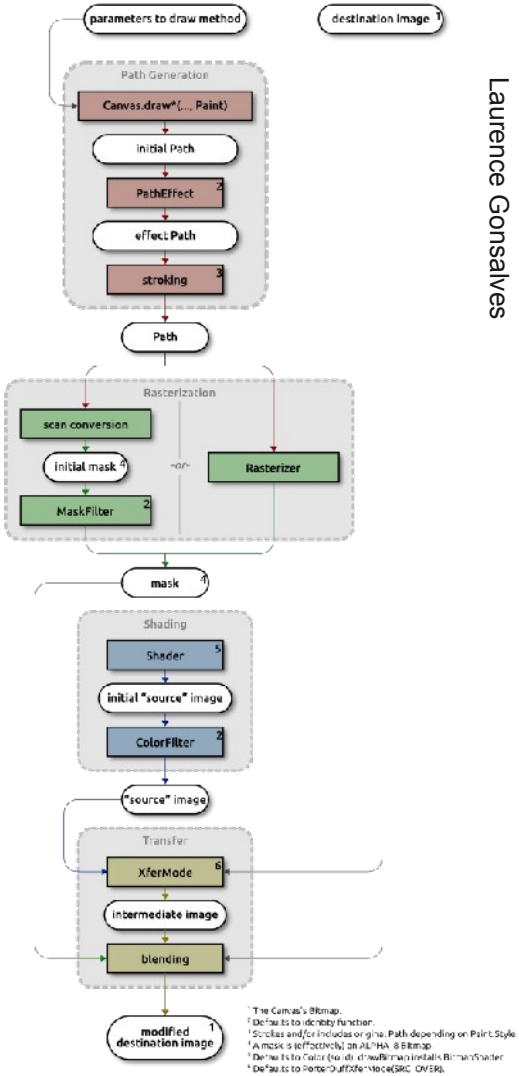
- Il metodo onDraw() viene invocato quando la view deve disegnarsi (nello spazio che è stato negoziato dalla onMeasure())
- A onDraw() viene passato un **Canvas**
  - Una superficie di disegno (eventualmente bufferizzata)
- Il nostro codice può disegnare sul canvas usando una o più **Paint**
  - "vernici" che specificano colore, trasparenza, tratteggi, font, ecc.

Canvas

- Un Canvas è una superficie di disegno virtuale
  - Non direttamente una bitmap!
- Supporta
  - Primitive di disegno varie
    - Geometriche
    - Immagini
    - Testo
  - Clipping
  - Matrici di trasformazione (rotazione, scalatura, ecc.)

La pipeline di rendering 2D

- Un Canvas è un contenitore per chiamate grafiche
- In realtà, compie molte altre operazioni prima che il disegno finisca su una Bitmap
  - **Rendering** di path, trasformazione secondo un PathEffect (arrotondamento degli angoli, tratteggio, ecc.)
  - **Rasterizzazione** (alpha-channel, anti-aliasing, filling, blurring, ecc.)
  - **Shading** (trasformazione di colori, gamma, ecc.)
  - **Trasferimento** (il disegno parziale viene combinato con i precedenti contenuti della Bitmap di destinazione, con vari operatori)



Metodi di Canvas

- **Clipping**
  - clipPath(), clipRect(), clipRegion()
- **Trasformazioni affini**
  - rotate(), scale(), translate(), skew(), concat(), setMatrix(), save(), restore()
- **Primitive di disegno**
  - drawBitmap/Picture()
  - drawArc/Circle/Oval()
  - drawColor/Paint/RGB/ARGB()
  - drawLine/Lines/Path/Vertices()
  - drawRect/RoundRect()
  - drawText/PosText/TextOnPath()
- **Informative**
  - getWidth/Height()
  - GetMaximumBitmapWidth/Height()
  - isHardwareAccelerated()
  - isOpaque()
  - getDensity()
  - getClipBounds(), getMatrix()

Varianti

- Di ogni metodo sono disponibili molte varianti, spesso con overloading e utilità varie
- Esempi:
  - void **drawRect**(float left, float top, float right, float bottom, Paint paint)
  - void **drawRect**(RectF rect, Paint paint)
  - void **drawRect**(Rect r, Paint paint)
  - void **drawRoundRect**(RectF rect, float rx, float ry, Paint paint)

Paint

- Un oggetto di classe Paint rappresenta una specifica completa del modo con cui una primitiva grafica (di Canvas) deve essere disegnata
- Proprietà più comuni
  - Colore, alpha
  - Dithering, antialiasing
  - Font, hinting, stili
  - Dimensione delle linee, tratteggio, cap, join
- Fornisce inoltre metodi per gestire la metrica del testo
  - Ascent, descent, misure in font proporzionali, ecc.

Costruttori della nostra view

- L'ultimo elemento mancante sono i **costruttori**
  - View(Context c)
    - Costruttore di base, associa la view al suo **context** (tramite il quale accedere, per esempio, alle risorse)
  - View(Context c, AttributeSet attr)
    - Costruttore chiamato quando la view viene creata a partire dalla specifica XML in un file di layout
    - Da attr si possono ottenere i valori degli **attributi**, tramite metodi getter come getAttribute<*tipo*>Value(String namespace, String attributo, <*tipo*> valoreDefault)
  - View(Context c, AttributeSet attr, int stile)
    - Costruttore che in aggiunta applica uno **stile** (identificato dal suo ID di risorsa)

Costruttori della nostra view

- L'ultimo elemento mancante sono i **costruttori**
  - View(Context c)
    - Costruttore di base, ass
      accedere, per esempio,
  - View(Context c, Attribut
    - Costruttore chiamato qu
      specifica XML in un file di layout
    - Da attr si possono ottenere i valori degli **attributi**, tramite metodi getter come getAttribute<*tipo*>Value(String namespace, String attributo, <*tipo*> valoreDefault)
  - View(Context c, AttributeSet attr, int stile)
    - Costruttore che in aggiunta applica uno **stile** (identificato dal suo ID di risorsa)

Manca il costruttore vuoto View(): non si può avere una view senza il suo contesto

Esempio: CartaQuadretti

- Vogliamo realizzare una variante custom di EditText (il widget di sistema per l'editing di testi) che abbia come sfondo una carta a quadretti
- Dovremo:
  - Estendere EditText
  - Implementare i costruttori
  - Fare override di onDraw()
- E, nel contempo, cercare di far fare più lavoro possibile alla superclasse!

CartaQuadretti – intestazione

```
package it.unipi.di.sam.customviewtest;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.widget.EditText;

public class CartaQuadretti extends EditText {

    int dimquad;
    Paint righini = new Paint();
```

Estendiamo la classe di sistema

Il tag da usare in layout.xml sarà <it.unipi.di.sam.customviewtest.CartaQuadretti>

Paint con cui disegneremo i righini

CartaQuadretti – costruttore

```
public CartaQuadretti(Context context) {
    super(context);
    init();
}

public CartaQuadretti(Context context, AttributeSet attrs, int defStyle) {
    super(context, attrs, defStyle);
    init();
}

public CartaQuadretti(Context context, AttributeSet attrs) {
    super(context, attrs);
    init();
}
```

Un metodo privato che si occupa di inizializzare la CartaQuadretti. Potrebbe prendere context, attrs e defStyle come argomenti, ma noi lo teniamo semplice.

CartaQuadretti – init

```
private void init() {
    righini.setARGB(20, 0, 0, 140);
    dimquad = 20;
}
```

Qui impostiamo righini a un colore fisso (blu semi-trasparente). Potremmo però rendere la nostra view configurabile, in vari modi:

- Potremmo definire un nome simbolico per il colore nelle risorse, farci passare il Context, e accedere al colore con

```
Color c = context.getResources().getColor(R.color.righini);
righini.setColor(c);
```

- Oppure, potremmo definire nuove risorse Stylable e includerle fra gli attributi del nostro tag XML (o anche eriditarle da un tema) – vedremo in futuro come fare.

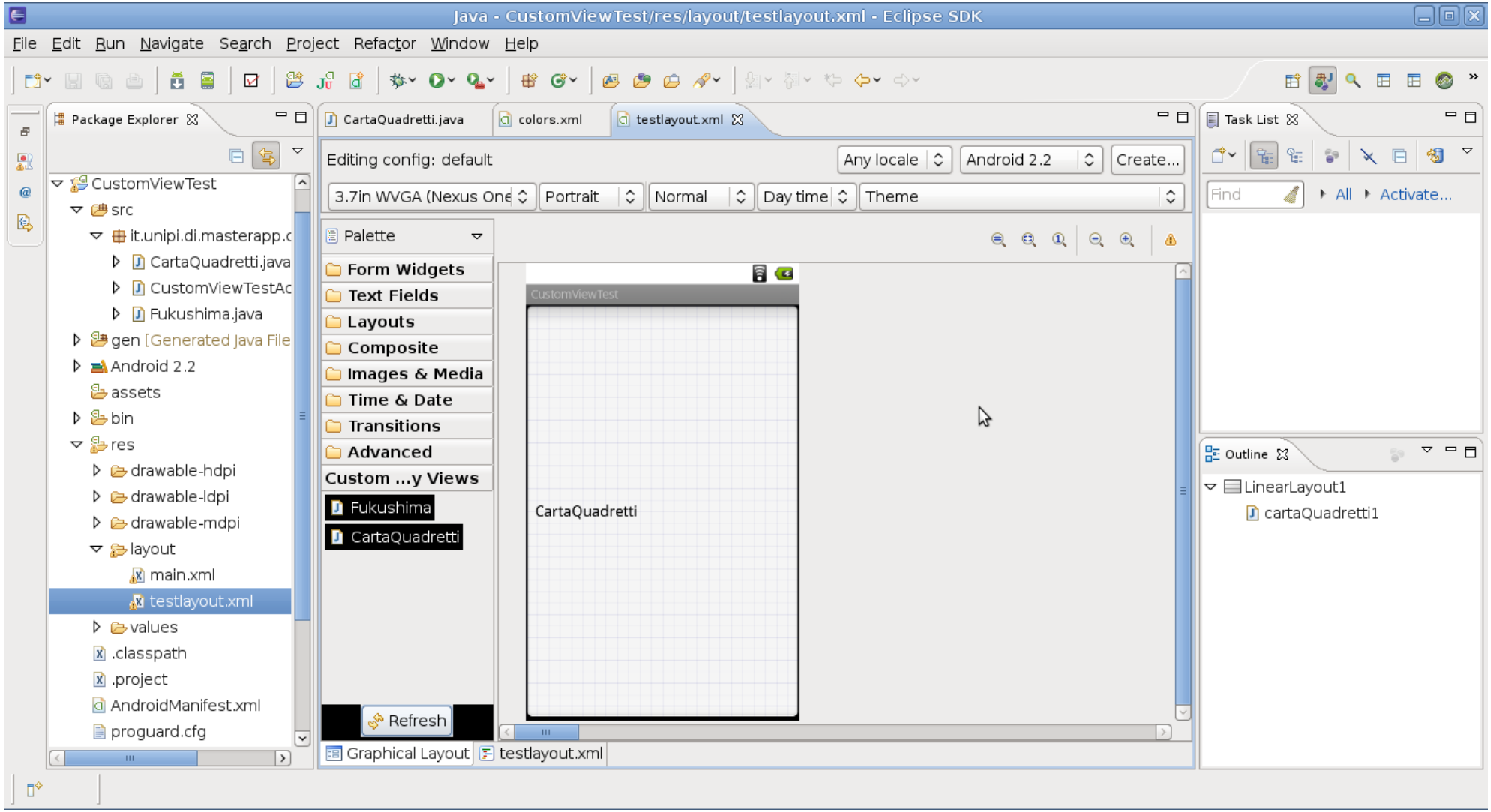
Stessa cosa per dimquad!

CartaQuadretti – disegno

```
protected void onDraw(Canvas canvas) {
    int w=getWidth(), h=getHeight();
    for (int x=0;x<w;x+=dimquad)
        canvas.drawLine(x, 0, x, h, righini);
    for (int y=0;y<h;y+=dimquad)
        canvas.drawLine(0, y, w, y, righini);
    super.onDraw(canvas);
}
```

Qui disegnamo tutto il resto (incluso il testo!)

CartaQuadretti – risultato



Ridisegno

- Una vista ha spesso delle proprietà
  - Grafica (es.: colore e dimensione dei quadretti)
  - Contenuto (es.: testo visualizzato)
- In questi casi, occorre un mezzo per ottenere il rinfresco della vista quando cambia la proprietà

```
public void setDimQuad(int n) {
    if (dimquad>0) {
        dimquad=n;
        invalidate();
    }
}
```

Invalida la view: il sistema chiamerà la onDraw() per ridisegnare la CartaQuadretti con il nuovo dimquad.

ListView e DataAdapter

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, si procede come sappiamo già
  - 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>

   <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>
```

Array riferito

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

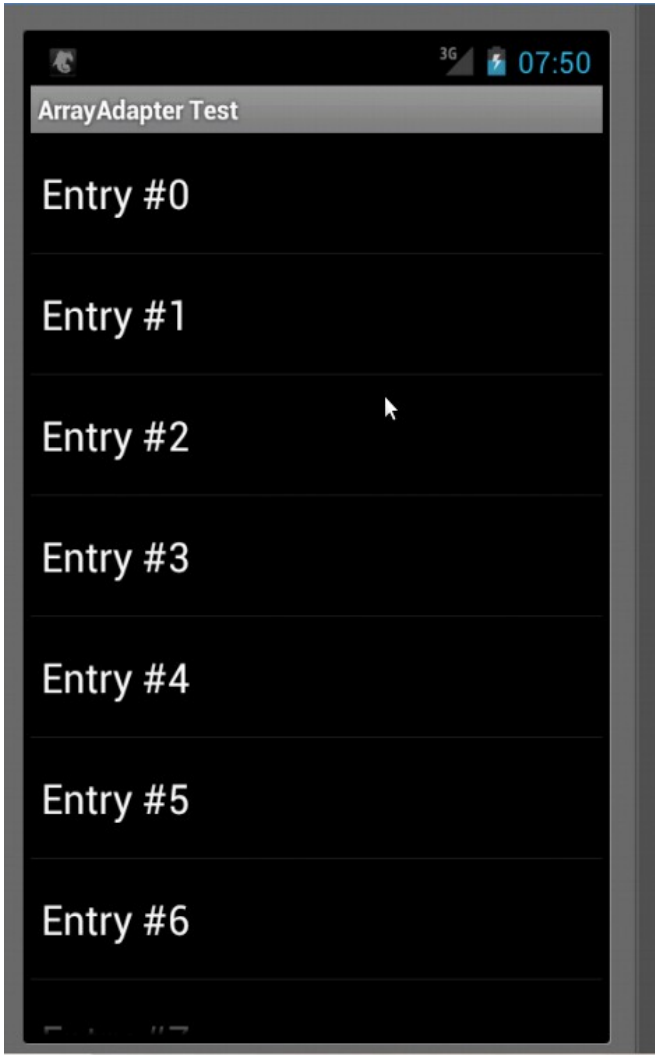
- Le diverse sottoclassi di Adapter traducono diversi formati di dati “grezzi”
- **ArrayAdapter<T>**: un array (Java) di elementi di tipo T
- Vari costruttori, con parametri:
  - Context (per accedere alle risorse)
  - ID del layout (XML) da utilizzare
  - ID della TextView dentro il layout da popolare 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);
          ...
    lv.setAdapter(adapter);
    lv.setOnItemClickListener(this);
          ...
  }

  public void onItemClick(AdapterView<?> parent, View view, int pos, long id) {
    CharSequence s=((TextView)view).getText();
    Log.d("AAT",s.toString());
  }
}
```

Facciamo implementare il listener all'Activity

Impostiamo noi stessi come listener alla ListView

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

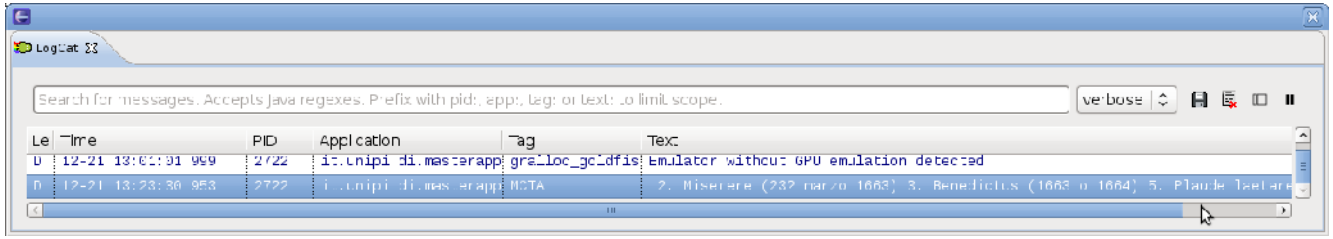
Gestione dell'input - multi

- 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[] getCheckedItemsIds()
  - 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 usate 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);
   }
}
```