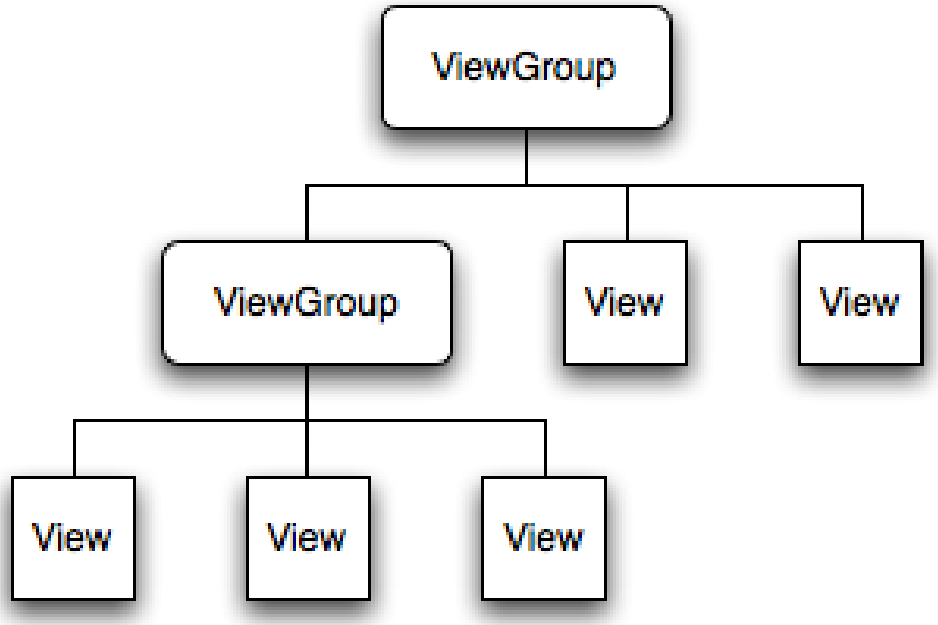
Lezione 6

Programmazione Android

- Definire la UI di un'Activity
  - Layout & View
  - Interazione (con richiami sui Listener)
  - Menu, opzioni, ActionBar

Layout & View

Layout & View

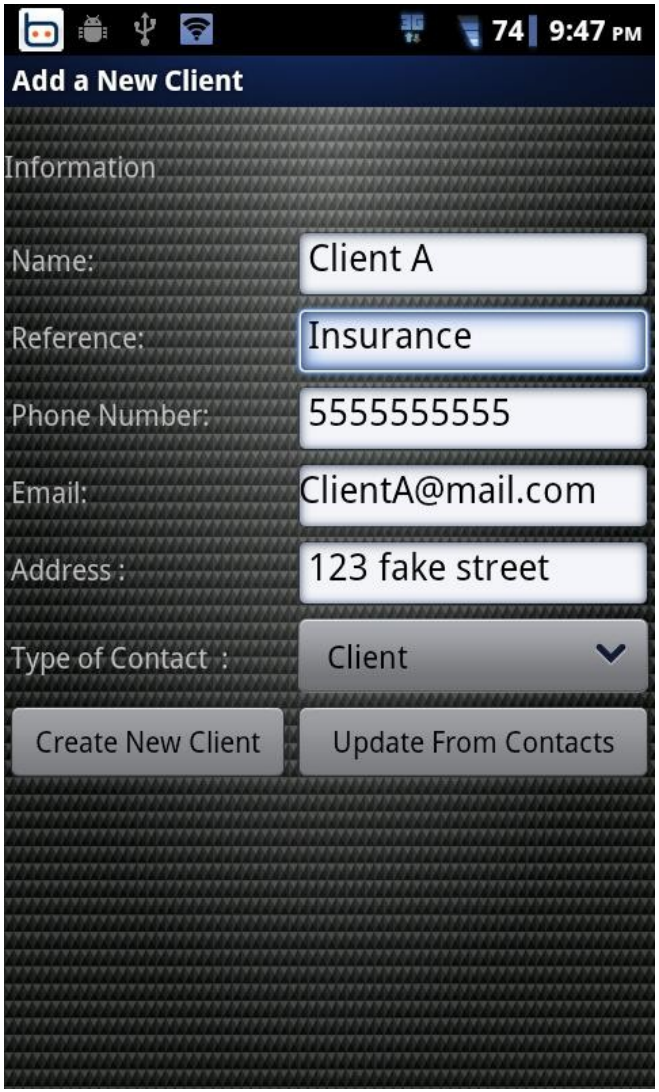


- Una UI Android è un albero con foglie di classe **View** e nodi intermedi di classe **ViewGroup**
  - Come già visto, tipicamente definito in XML
- Ogni View è una classe Java con nome uguale al tag XML relativo
- La disposizione visuale delle view è regolata da un **Layout**

Layout & View

- Un ViewGroup può contenere un numero qualunque di View
- ViewGroup è una sottoclasse di View
  - → Tipica gerarchia ad albero
- Una View che gestisce input è detta **Widget**
- Un contenitore di Widget è detto **Widget Host** (es.: Home)

Layout & View



- Un ViewGroup che determina il posizionamento dei figli è detto **LayoutManager**
- Il posizionamento si basa una una **negoziazione** fra esigenze del contenitore ed esigenze dei contenuti
- Esistono molte strategie possibili
  - E di conseguenza, molti layout manager diversi

Layout Manager comuni

- **AbsoluteLayout**
  - Coordinate assolute (x,y) per ogni componente – ugh!
- **LinearLayout**
  - Serie verticale o orizzontale di componenti
- **RelativeLayout**
  - Posizione di ogni componente relativa agli altri o al contenitore
- **GridLayout**
  - Griglia di celle di dimensione variabile (ma allineate); componenti a cavallo di più celle

Layout Manager comuni

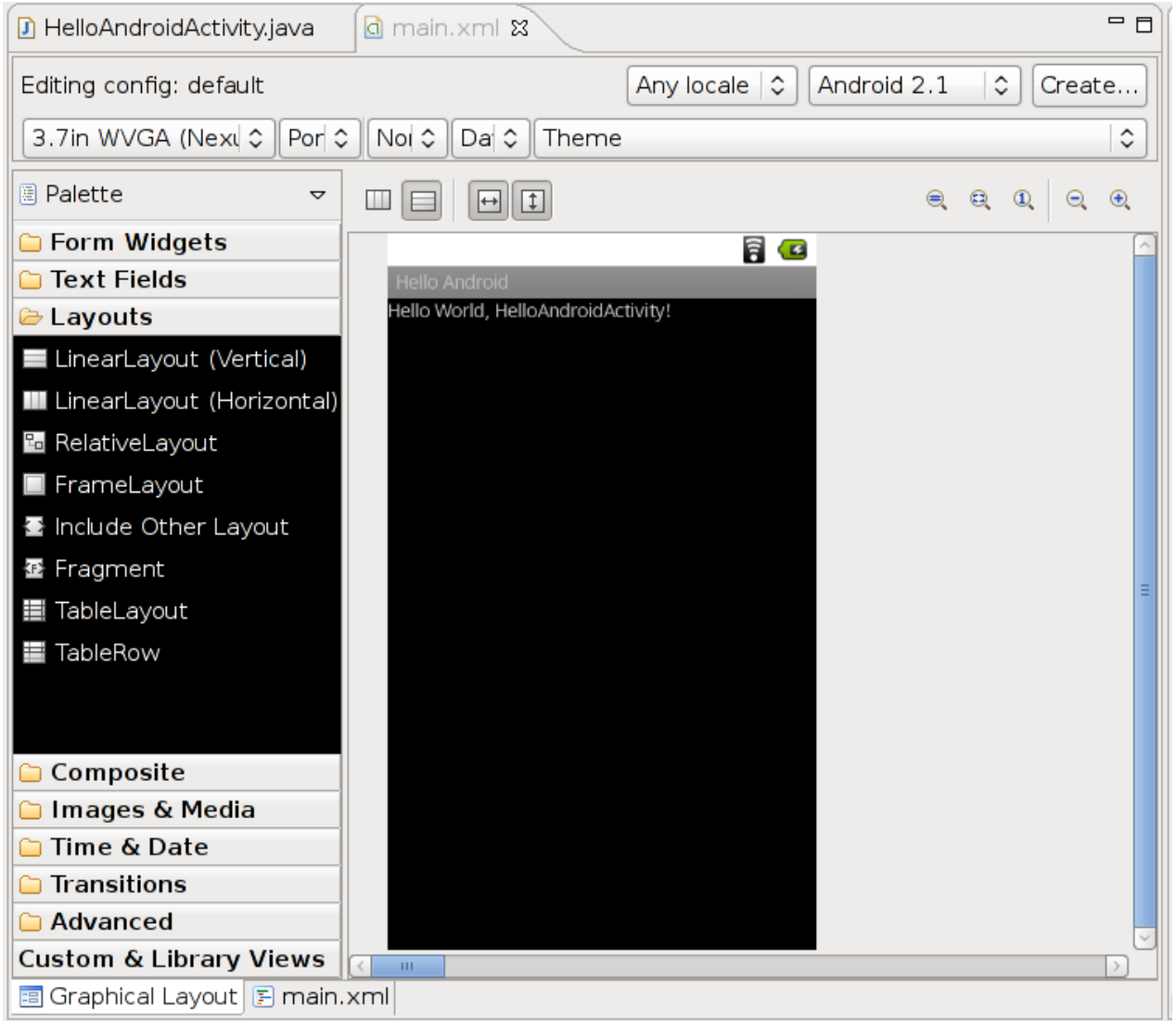
- **FrameLayout**
  - Componenti uno sull'altro (l'ultimo aggiunto sta in cima) – solitamente usato con un solo componente
- **TableLayout**
  - Versione più “antica” di GridLayout (con alcune limitazioni) – raramente usata
  - Ha come figli singole View (che coprono una riga intera) o oggetti **TableRow** (i quali hann ocome figli le View che occupano le singole celle)
- È sempre possibile comporre layout o scrivere i propri Layout Manager

Layout manager meno comuni

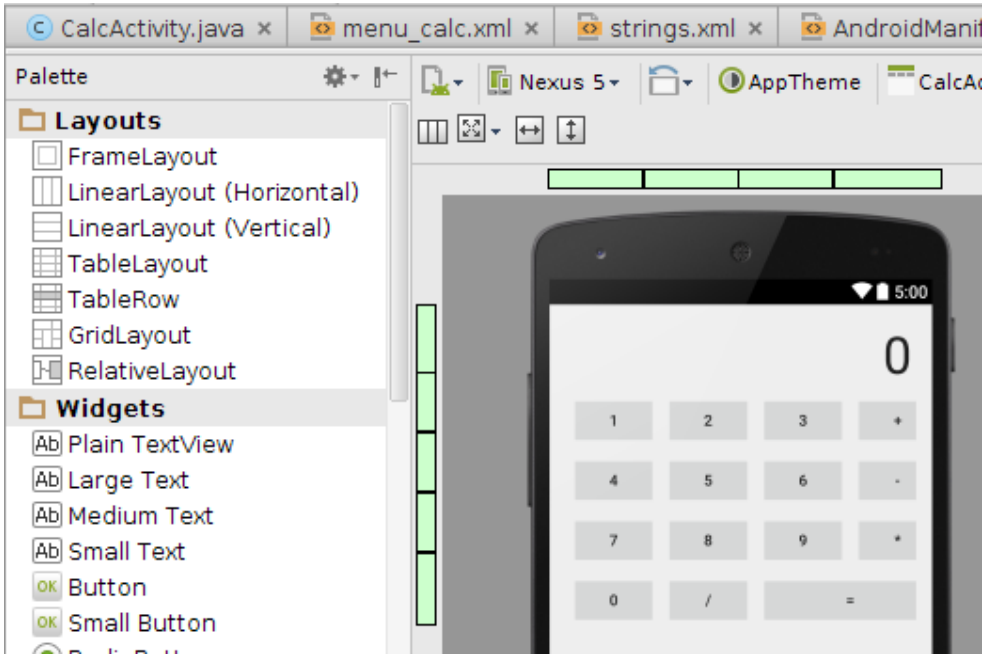
- Esiste un certo numero di LayoutManager più specializzati
  - Solitamente usati internamente da componenti più complessi
  - **DrawerLayout** – per i menu a scorrimento laterali
  - **SwipeRefreshLayout** – supporta lo swipe-to-refresh
- Numerose altre View complesse sono *anche* layout
  - **Toolbar** – classica barra di strumenti
  - **TvView** – mostra programmi TV (!)
  - **WebView** – browser web
  - **CalendarView** – pagina di calendario
  - **Gallery** – classica gallery fotografica
  - ecc. (circa una quarantina di classi)

Layout Manager in Eclipse

- Eclipse fornisce un **editor grafico** per i file di layout (XML)
- Fra l'altro, è possibile scegliere il layout per ogni gruppo



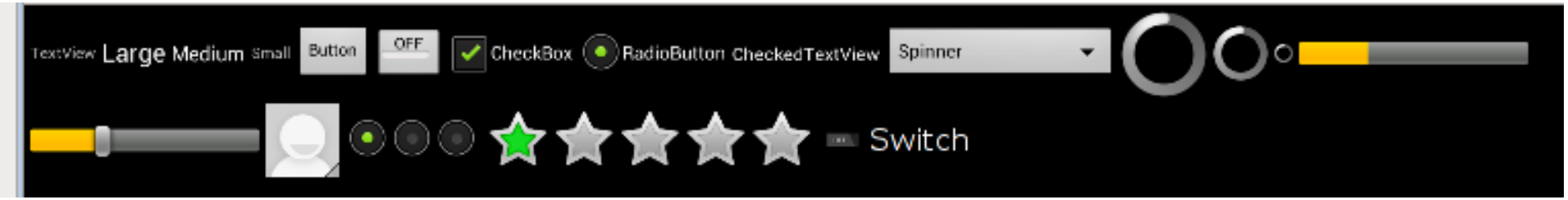
Layout Manager in Android Studio



- Vengono offerti i Layout più comuni
- È possibile impostare i vincoli in maniera grafica (GUI) o testuale (property sheet)
- Sempre possibile editare l'XML

View

- Una View “foglia” è un widget
- Le librerie di sistema forniscono una vasta scelta di widget standard...
  - … ma è sempre possibile scrivere i propri widget
  - Basta creare sottoclassi di View (o del widget che meglio approssima quello che ci serve)
  - Vedremo più avanti come scrivere un proprio widget

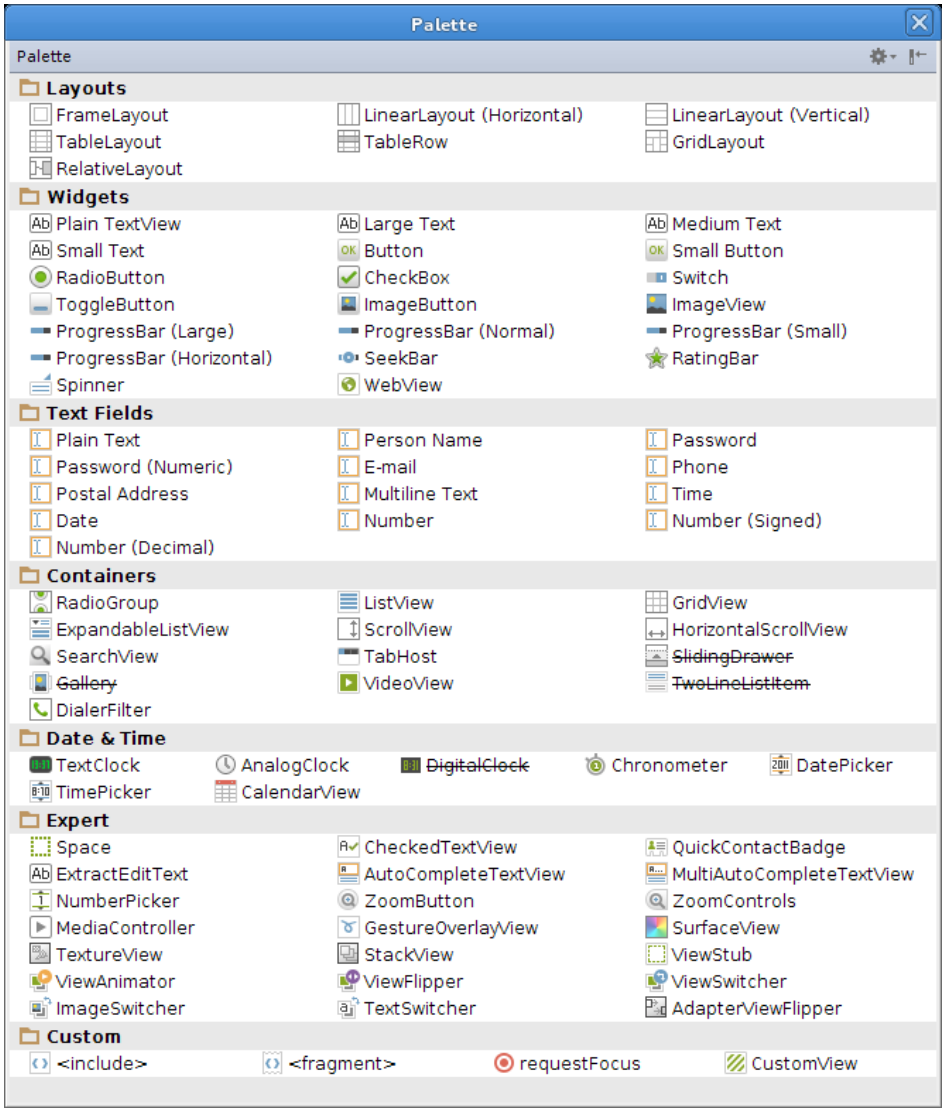


Widget in Eclipse

- In Eclipse, i widget sono raggruppati in categorie
  - **Form widgets** – classici (pulsanti, checkbox, ...)
  - **Text fields** – campi di testo con varie regole di validazione dell'input (nomi, numeri, password, ...)
  - **Composite** – widgets che ne includono altri (tab multiple, listview, webview, ...)
  - **Images & Media** – widget per i multimedia (player video, galleria di foto, …)
  - **Time & Date** – gestione del tempo (calendari, orologi, ecc.)
  - **Advanced** – widget specializzati (zoom, OpenGL, …)

Widget in Android Studio

- Analogamente in Android Studio abbiamo:
  - Layouts
  - Widgets
  - Text fields
  - Containers
  - Date & Time
  - Expert
  - Custom



View: XML vs Java

- <TextView
  - android:gravity=...
  - android:width=...
  - android:height=...
  - android:scrollhorizontally=...
  - android:shadowcolor=...
  - android:shadowDx=...
  - android:shadowDy=...
  - android:shadowRadius=...
- />

- public class TextView {
  - setGravity(...)
  - setWidth(...)
  - setHeight(...)
  - setHorizontallyScrolling(...)
  - SetShadlowLayer(.....)

  }

C'è una corrispondenza (non perfetta) fra attributi XML e metodi Java

View: XML vs Java

- C'è invece una corrispondenza perfetta tra nome del tag XML e nome della classe Java
- Tag non qualificati ↔ Classi del package **`android.widget`**
  - Es: **`<TextView>`** ↔ **`android.widget.TextView`**
- Tag qualificati ↔ Classi custom
  - Es: **`<it.unipi.di.masterapp.MioWidget>`**
  - Le classi custom che implementano widget **devono** ereditare da View!

Interazione

Gestione dell'input

- A run-time, esiste un albero di oggetti Java che creato a partire dall'albero XML del layout
- Gli oggetti possono ricevere input dall'utente (si interfacciano col sistema touch)
- Quando si verifica un **evento** significativo, viene chiamato un *handler*
  - La vostra Activity può registrare propri handler
  - In Java, sono *inner interfaces* dentro la classe View
  - Ogni interfaccia definisce un metodo **on**...**Listener()**

Esempio di Listener

```
private OnClickListener listener = new OnClickListener()
{
    public void onClick(View v) {
      // reazione: per esempio, lanciamo una Activity
    }
};

protected void onCreate(Bundle stato) {
    ...
    // prendi un riferimento al pulsante di nome "b"
    Button b = (Button)findViewById(R.id.b);
    // registra il listener per il click di b
    b.setOnClickListener(listener);
    ...
}
```

Già visto!

Esempio di Listener

```
private OnClickListener listener = new OnClickListener()
{
    public void onClick(View v) {
      // reazione: per esempio, lan                    ty
    }
};

protected void onCreate(Bundle stat
    ...
    // prendi un riferimento al pulsante di nome "b"
    Button b = (Button)findViewById(R.id.b);
    // registra il listener per il click di b
    b.setOnClickListener(listener);
    ...
}
```

Non s'era detto di evitare la new?

Già visto!

Esempio di Listener

```
public class act extends Activity
                    implements OnClickListener {

    protected void onCreate(Bundle stato) {
        ...
        Button b = (Button)findViewById(R.id.b);
        button.setOnClickListener(this);
    }

    public void onClick(View v) {
      // reazione: per esempio, lanciamo una Activity
    }
    ...
}
```

Già visto!

Alcune interfacce *Listener

- **OnAttachStateChangeListener**
  - Le View possono essere inserite o rimossa da un albero dinamicamente (a run-time)
  - onViewAttachedToWindow(View v)
  - onViewDetachedFromWindow(View v)
- **OnClickListener**
  - Chiamato quando c'è un click (logico!) sulla View
  - onClick(View v)

Alcune interfacce *Listener

- **OnDragListener**
  - Le View possono essere drag-droppate una sull'altra
  - onDrag(View v, DragEvent e)
  - Vengono chiamati gli onDrag(), secondo un protocollo noto
    - della vista draggata
    - di quella da cui viene draggata
    - di quelle su cui passa
    - di quella su cui viene droppata

Alcune interfacce *Listener

- **OnGenericMotionListener**
  - Quando l'utente poggia o sposta il dito sullo schermo
  - onGenericMotion(View v, MotionEvent e)
  - Inviato *prima* di “interpretare” l'evento (es.: come click)
- **OnKeyListener**
  - Quando l'utente preme un tasto
  - onKey(View v, int keycode, KeyEvent ke)
- **OnLongClickListener**
  - onLongClick(View v) – simile a onClick(), ma più lungo!

Quali eventi?

- Ogni sottoclasse di View (anche custom) è libera di definire i propri eventi (e i listener associati)
- Non esiste quindi una tabella completa: occorre consultare la documentazione della classe
- Es.: MediaPlayer definisce 8 eventi “propri” (più quelli ereditati)

Summary

Nested Classes		
interface	MediaPlayer.OnBufferingUpdateListener	Interface definition of a callback to be invoked indicating buffering status of a media resource being streamed over the network.
interface	MediaPlayer.OnCompletionListener	Interface definition for a callback to be invoked when playback of a media source has completed.
interface	MediaPlayer.OnErrorListener	Interface definition of a callback to be invoked when there has been an error during an asynchronous operation (other errors will throw exceptions at method call time).
interface	MediaPlayer.OnInfoListener	Interface definition of a callback to be invoked to communicate some info and/or warning about the media or its playback.
interface	MediaPlayer.OnPreparedListener	Interface definition for a callback to be invoked when the media source is ready for playback.
interface	MediaPlayer.OnSeekCompleteListener	Interface definition of a callback to be invoked indicating the completion of a seek operation.
interface	MediaPlayer.OnTimedTextListener	Interface definition of a callback to be invoked when a timed text is available for display.
interface	MediaPlayer.OnVideoSizeChangedListener	Interface definition of a callback to be invoked when the video size is first known or updated

Un metodo alternativo

- Per i casi più semplici, la classe View offre anche la possibilità di definire **nel layout XML** il **nome** del metodo da chiamare in risposta a un evento
- Deve essere un metodo con un parametro di tipo View e tipo di ritorno void
- Deve essere un metodo definito nel “contesto” (ovvero: nell'Activity a cui appartiene la view)

Un metodo alternativo

- In **`layout/….xml`**

```
<Button … android:onclick="clicked1" … />
```

Nel file .java dell'activity

```
public class … extends Activity {
    public void onCreate(Bundle b) {
      …
    }

    public void clicked1(View v) {
      // qui v è il Button che è stato premuto
    }
}
```

Un metodo alternativo

- Quanto è efficiente questo metodo?
- Dal costruttore di **`android.view.View`**:

```
if (handlerName != null) {
  setOnClickListener(new OnClickListener() {
    private Method mHandler;

    public void onClick(View v) {
      if (mHandler == null) {
        try {
          mHandler = getContext().getClass().getMethod(handlerName, View.class);
        } catch (NoSuchMethodException e) { lancia un'eccezione }
      }

      try { mHandler.invoke(getContext(), View.this); }
      catch (IllegalAccessException e) { lancia un'eccezione }
      catch (InvocationTargetException e) { lancia un'eccezione }
    }
  } );
}
```

Uso di Reflection!

Menu, Opzioni e ActionBar

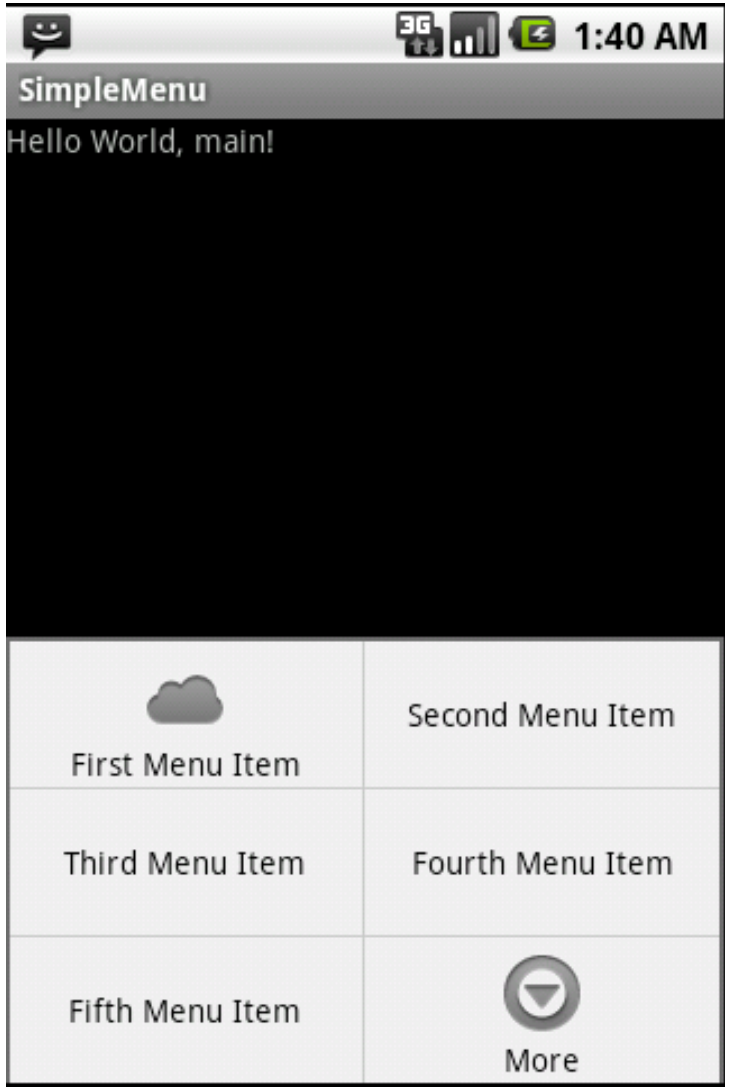
Il sistema dei menu

- Su Android, il menu (o meglio: le opzioni) non è un componente *grafico*, ma uno *logico*
- In altre parole:
  - L'app dichiara quali scelte devono essere disponibili all'utente
  - Il sistema decide autonomamente come presentarle
    - In base alla versione di S.O., allo spazio disponibile su schermo, alla presenza di una tastiera o meno, ecc.
    - In base al fatto che siano *azioni* o *opzioni*, a quanto è lunga la loro etichetta, ecc.
    - In base alla *priorità* specificata dal programmatore

Il sistema dei menu

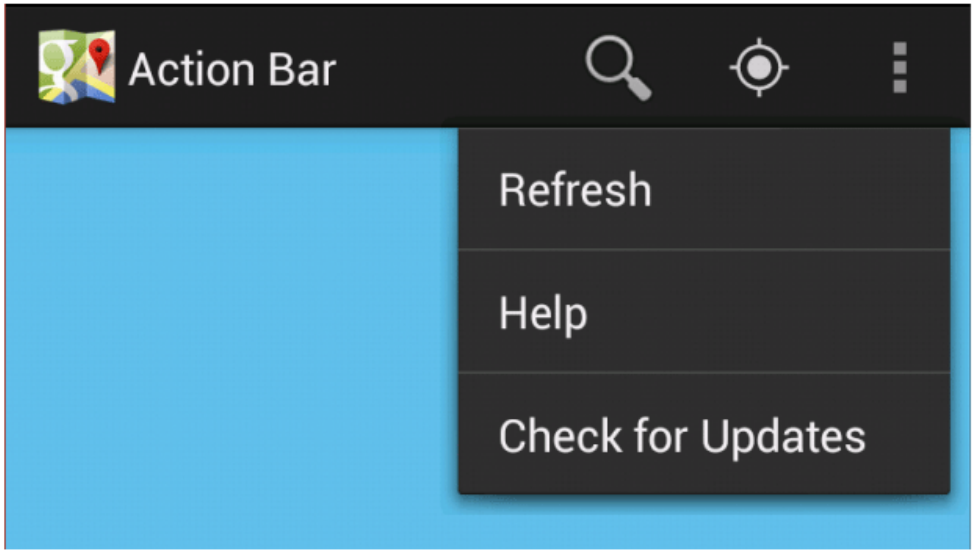
- Android utilizza un sistema non-convenzionale per i menu
  - Niente liste gerarchiche con etichette...
- Tre stadi:
  - Il menu “primario” è composto da (solitamente al più) 6 caselle con icone e opzionalmente testo
  - Una di queste può essere un “Altro...” che mostra una lista di voci più lunga
  - Una voce può aprire un sottomenù (finestra floating)
- Si possono anche avere menù contestuali
  - Richiamabili da un long click su una View

Il sistema dei menu



- Il sistema decide **autonomamente** come, dove e quante voci mostrare nel menu primario
  - Rispetta però l'ordine di importanza definito dal programmatore
  - Aggiunge automaticamente la voce “More” che apre un menu secondario se ci sono altre voci
- Il menu primario mostra preferibilmente **icone+testo**, non checkmark o altro
- Il sistema può spostare delle voci nell'ActionBar (da Honeycomb in poi)

Il sistema dei menu



- Se c'è una ActionBar, il sistema mette le voci più importanti come icone; le altre in un menu “tre puntini”
- Retrocompatibilità
  - app senza ActionBar su dispositivi senza tasto menu fisico → “tre puntini” aggiunto ai softkeys su schermo

Creazione di menu

- Il modo più semplice di creare un menù è di usare (come al solito) un file XML in res/menu/

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

<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu1" android:icon="@drawable/ic_menu1"
          android:title="@string/menu1" />
    <item android:id="@+id/menu2" android:icon="@drawable/ic_menu2"
          android:title="@string/menu2" />
    <item android:id="@+id/sottomenu" android:icon="@drawable/ic_sottomenu"
          android:title="@string/sottomenu" >
        <menu>
            <item android:id="@+id/sotto1" android:title="@string/sotto1" />
            <item android:id="@+id/sotto2" android:title="@string/sotto2" />
        </menu>
    </item>
</menu>
```

Creazione di menu

- **<menu>** definisce un menu
- **<item>** definisce una voce (con eventuale sottomenu)
- **<group>** può essere usato per raggruppare più <item> logicamente correlati, ma in maniera "invisibile"
  - Gli <item> ereditano alcune proprietà da <group>
    - Per esempio, possono essere abilitati/disabilitati in blocco

Creazione di menu

- Gli <item> possono avere altre proprietà

3.0+

```
<item android:id="@[+][package:]id/resource_name"
      android:title="string"
      android:titleCondensed="string"
      android:icon="@[package:]drawable/drawable_resource_name"
      android:onClick="method name"
      android:showAsAction=["ifRoom" | "never" | "withText" |
                            "always" | "collapseActionView"]
      android:actionLayout="@[package:]layout/layout_resource_name"
      android:actionViewClass="class name"
      android:actionProviderClass="class name"
      android:alphabeticShortcut="string"
      android:numericShortcut="string"
      android:checkable=["true" | "false"]
      android:visible=["true" | "false"]
      android:enabled=["true" | "false"]
      android:menuCategory=["container" | "system" | "secondary" |
                                                    "alternative"]
      android:orderInCategory="integer" />
```

Creazione di menu

- Solo alcune di queste possono essere definite per i <group>

```
<group android:id="@[+][package:]id/resource name"
       android:checkableBehavior=["none" | "all" | "single"]
       android:visible=["true" | "false"]
       android:enabled=["true" | "false"]
       android:menuCategory=["container" | "system" | "secondary"
                                                    | "alternative"]
       android:orderInCategory="integer" >
```

Il ciclo di vita di un menu

- Come già abbiamo visto per le Activity, i cicli di vita degli oggetti su Android sono controllati dal sistema (non dall'applicazione)
  - L'intera UI è affidata al sistema!
- La nostra activity dovrà fornire delle callback:
  - **onCreateOptionsMenu()** - crea il menu
  - **onPrepareOptionsMenu()** - sta per visualizzare il menu
  - **onOptionsItemSelected()** - reagisce alle selezioni

Creazione di un menu

- public boolean **onCreateOptionsMenu**(Menu m)
  - **true** → ok, ho un menu, l'ho messo in m
  - **false** → non ho un menu, non visualizzare nulla

```
public boolean onCreateOptionsMenu(Menu m) {
    MenuInflater mi = getMenuInflater();
    mi.inflate(R.menu.miomenu, m);
    return true;
}
```

Creazione di un menu

- È anche possibile creare un menu “a mano” senza definirlo in XML
  - ma... pensateci bene! Non avete una famiglia? Degli affetti? Degli amici?

```
public boolean onCreateOptionsMenu(Menu m){
    m.add(0,1,0,"voce 1");
    m.add(0,2,0,"voce 2");
    m.add(“voce 3”);
    return true;
}
```

Creazione di un menu

- È anche possibile creare un menu "a mano" senza de
  -

add(int groupid, int itemid, int order, CharSequence title)

Ci sono poi altre varianti overloaded che prendono solo il titolo, oppure un Resource ID anziché una stringa

```
public boolean onCreateOptionsMenu(Menu m){
    m.add(0,1,0,"voce 1");
    m.add(0,2,0,"voce 2");
    m.add("voce 3");
    return true;
}
```

Creazione di un menu

- La **onCreateOptionsMenu**() viene chiamata una volta sola
  - Su Android < 3.0, alla prima apertura del menu
  - Su Android ≥ 3.0, alla creazione dell'Activity
    - Perché alcune voci possono finire nell'ActionBar, sempre visibili!
- La **onPrepareOptionsMenu**() viene chiamata
  - Su Android < 3.0, prima di ogni apertura del menu
  - Su Android ≥ 3.0, solo se prima chiamate **invalidateOptionsMenu**()

Modifiche dinamiche a un menu

- In tutti i casi, la **onPrepareOptionsMenu**() è dove si possono apportare modifiche a un menu
  - Abilitare o disabilitare certe voci (ghosting)
  - Aggiungere o rimuovere voci
  - Cambiare testo, icone, checkmark, ...

```
public boolean onPrepareOptionsMenu(Menu m){
   MenuItem mi=m.find(VOCE_1);
   if (…)  mi.setIcon(R.drawable.ic_voce_1a);
   else    mi.setIcon(R.drawable.ic_voce_1b);
   return true;
}
```

Modifiche dinamiche a un menu

- La onPrepareOptionsMenu() deve restituire **true** se il menu è disponibile
  - Oppure **false** per indicare che non c'è nessun menu da mostrare
- Come regola di stile, è opportuno non configurare *troppo* il menu – altrimenti l'utente si perde
  - Semmai, è possibile usare i menu contestuali, oppure altri controlli di UI
- Ricordate: da 3.0 in poi, dopo una chiamata a onPrepareOptionsMenu(), non ce ne sarà un'altra se non viene chiamato prima invalidateOptionsMenu()

Rispondere alla selezione

- I menu notificano la loro selezione in maniera analoga a quanto fanno i widget
- L'attributo **onClick** dell'XML può indicare il nome di un metodo dell'activity da chiamare
  - 3.0+
  - Il metodo deve prendere come unico argomento un MenuItem – che indica chi è stato selezionato
- Oppure, il sistema chiama il callback **onOptionsItemSelected**(MenuItem mi)
  - Differenza con le View: c'è un solo handler per i menu, non occorre usare i Listener

Rispondere alla selezione

uso di onclick

- Nella definizione XML del menu:

```
<item … android:onclick="voce1" … />
```

- Nel codice della activity:

```
public void voce1(MenuItem mi) {
// qui mi è il menu item che è stato premuto
}
```

Rispondere alla selezione

uso di onOptionsItemSelected()

- Nel codice della activity:

```
public boolean onOptionsItemSelected(MenuItem mi) {
    switch (mi.getItemId()) {
    case R.id.menu1:
        …
        return true;
    case R.id.menu2:
        …
        return true;
    case …
    default:
        return super.onOptionsItemSelected(mi);
    }
}
```

Il metodo deve restituire TRUE se ha **consumato** l'evento, FALSE altrimenti.

In questo caso, ci sono altri modi per rispondere alla selezione!

Rispondere alla selezione

- Se la selezione di una voce di menu non è stata gestita con i metodi precedenti, possono accadere ancora due cose:
  - Il menu può essere configurato per lanciare automaticamente un Intent
    - mi.**setIntent**(intent)
    - Si può anche pre-filtrare in base ai potenziali riceventi
  - Il menu può essere configurato per chiamare un Listener (come abbiamo visto per le View)
    - mi.**setOnMenuItemClickListener**(listener)
    - listener.**onOptionsItemSelected**(MenuItem mi)

Rispondere alla selezione

- Se la selezione di una v
  gestita con i metodi prec
  ancora due cose:
  - Il menu può essere configurato per lanciare automaticamente un Intent
    - mi.**setIntent**(intent)
    - Si può anche pre-filtrare in base ai potenziali riceventi
  - Il menu può essere configurato per chiamare un Listener (come abbiamo visto per le View)
    - mi.**setOnMenuItemClickListener**(listener)
    - listener.**onOptionsItemSelected**(MenuItem mi)

Di tutte queste possibillità, usare la **onOptionsItemSelected**() della activity è la più efficiente!

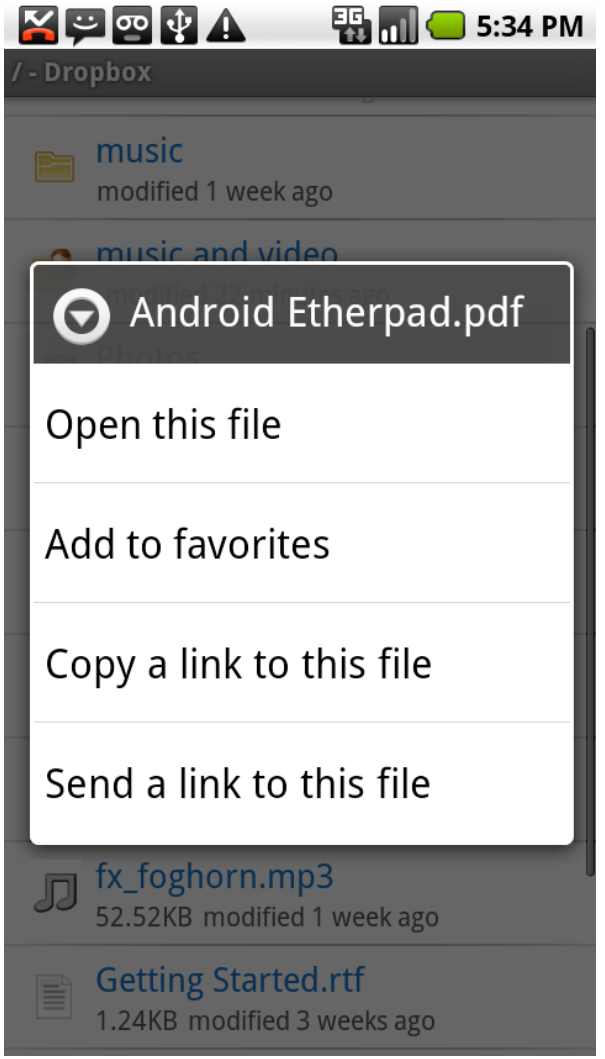
(non si fa nessuna **new**!)

Altre callback del ciclo di vita dei menu

- Il sistema consente di intercettare il funzionamento dei menu in altri punti “critici”
- Molte hanno a che fare con la costruzione dei **panel** (le superfici che ospitano i menu)
  - onMenuOpened()
  - onOptionsMenuClosed()
  - onPanelClosed()
  - onPreparePanel()
  - onMenuItemSelected()

Menu contestuali

- Sono l'equivalente Android del “tasto destro”
- Invocati quando si tiene premuto su una View per un tempo lungo (circa tre secondi)
- Dipendono dalla particolare View su cui sono invocati!
  - Mentre i menu che abbiamo visto finora dipendevano dall'Activity, essendo più “globali”



Creazione di context menu

- La creazione di context menu può seguire due strade
  - Si può creare una sottoclasse della View che ci serve, e fare override del suo metodo **onCreateContextMenu**(ContextMenu cm)
    - Vedremo questo metodo quando parleremo di come creare le nostre View
  - Si può implementare il metodo **onCreateContextMenu**() dell'activity, e registrare le view che devono invocarlo
    - Tutte le view non registrate non avranno context menu

Creazione di context menu

- Supponiamo di avere nella nostra GUI un widget TextEdit con ID *te* a cui vogliamo associare un context menu
- Dovremo registrare *te* alla partenza

```
public void onCreate(Bundle b) {
   super.onCreate(b);
   setContentView(R.layout.main);
   ...
   TextEdit te=(TextEdit)findViewById(R.id.te);
   registerForContextMenu(te);
   ...
}
```

Creazione di context menu

- La **registerForContextMenu**() imposta il listener OnCreateContextMenuListener della View alla nostra Activity, che dovrà implementare il metodo **onCreateContextMenu**()

```
public void onCreateContextMenu(ContextMenu m,
                                View v, ContextMenuInfo cmi)
{
   super.onCreateContextMenu(m, v, cmi);
   // o così (raro, avremmo sempre lo stesso menu)
   MenuInflater mi = getMenuInflater();
   mi.inflate(R.menu.miocontextmenu, m);
   // o così (più probabilmente, dipende da v)
   m.setHeaderTitle(...);
   m.add(...);
}
```

Rispondere alla selezione dei context menu

- La selezione è gestita come per i menu globali dell'activity
- Uno dei seguenti metodi:
  - Registrare un intent direttamente nel MenuItem
  - Registrare un MenuItemClickListener nel MenuItem
  - Implementare **onContextItemSelected**(MenuItem mi) nell'Activity
    - Come al solito, si discrimina poi in base all'ID di mi
- Quest'ultimo è il metodo più efficiente

Context actions

- Come alternativa ai context menu, è possibile utilizzare una *action bar contestuale*

3.0+
raccomandato

  - Viene mostrata solo quando richiesto dal programma
  - Si sovrappone visivamente alla Action Bar dell'activity, ma è un oggetto separato
- Si implementa l'interfaccia **ActionMode.Callback**
  - Metodi analoghi a quelli per gli OptionsMenu
- Si chiama **startActionMode()** per aprire l'action bar contestuale
  - Per esempio, dentro la onLongClick() di una view

Context actions

implementare ActionMode.Callback

```
private ActionMode.Callback cab = new ActionMode.Callback() {
    // creazione: viene invocata dopo una startActionMode()
    @Override
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        MenuInflater inflater = mode.getMenuInflater();
        inflater.inflate(R.menu.context_menu, menu);
        return true;
    }
    // preparazione: come per gli OptionsMenu
    @Override
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        return false; // Return false if nothing is done
    }
    // azione: l'utente ha selezionato un item
    @Override
    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_share:
                shareCurrentItem();
                mode.finish(); // l'utente ha selezionato, chiudiamo la CAB
                return true; // evento consumato
            default:
                return false; // evento non consumato
        }
    }
    // distruzione: comodo segnarselo in un flag
    @Override
    public void onDestroyActionMode(ActionMode mode) {
       mycab=null;
    }
};
```

Il parametro **ActionMode** passato ai vari metodi è la CAB su cui si sta operando: la stessa ActionMode.Callback può essere usata per più istanze di CAB.

I parametri **Menu** e **MenuItem** sono gli stessi usati dagli OptionsMenu.

Context actions
aprire e chiudere la CAB

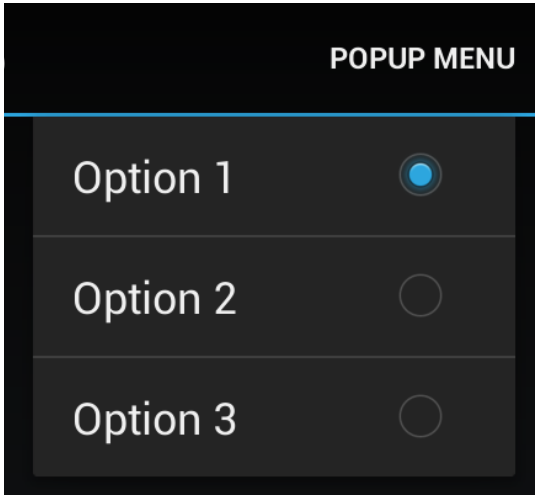
```
myview.setOnLongClickListener(new View.OnLongClickListener() {

    public boolean onLongClick(View v) {
        if (mycab != null) {
            return false;  // CAB già aperta, non facciamo niente (e non consumiamo l'evento)
        } else {  // avvia la CAB
            mycab = getActivity().startActionMode(cab);
            v.setSelected(true);
            return true;        // consumiamo l'evento
        }
    }
});
```

Context actions

- La CAB (oggetto ActionMode) offre alcuni metodi di ulteriore configurazione
  - setTitle(), setSubTitle(), setTitleOptionalHint(), …
- È poi possibile accedere direttamente alla View che implementa la parte “titolo” della CAB
  - setCustomView(), getCustomView()
  - Utile, per esempio, per sostituire il titolo con un widget di search
- In generale: meglio non manipolare troppo la CAB, e lasciare che il sistema faccia il suo

Popup menu

- L'ultimo tipo di menu su Android riguarda i **pop-up**
  - … che in effetti sono dei pull-down!
  - Lo stesso tipo di menu usato su Android 3.0+ per simulare i menu 2.x
- Consente di collegare un pannello menu a qualunque View
  - Il pannello compare subito sotto la View
    - Se c'è spazio: altrimenti, prova sopra)



Popup menu

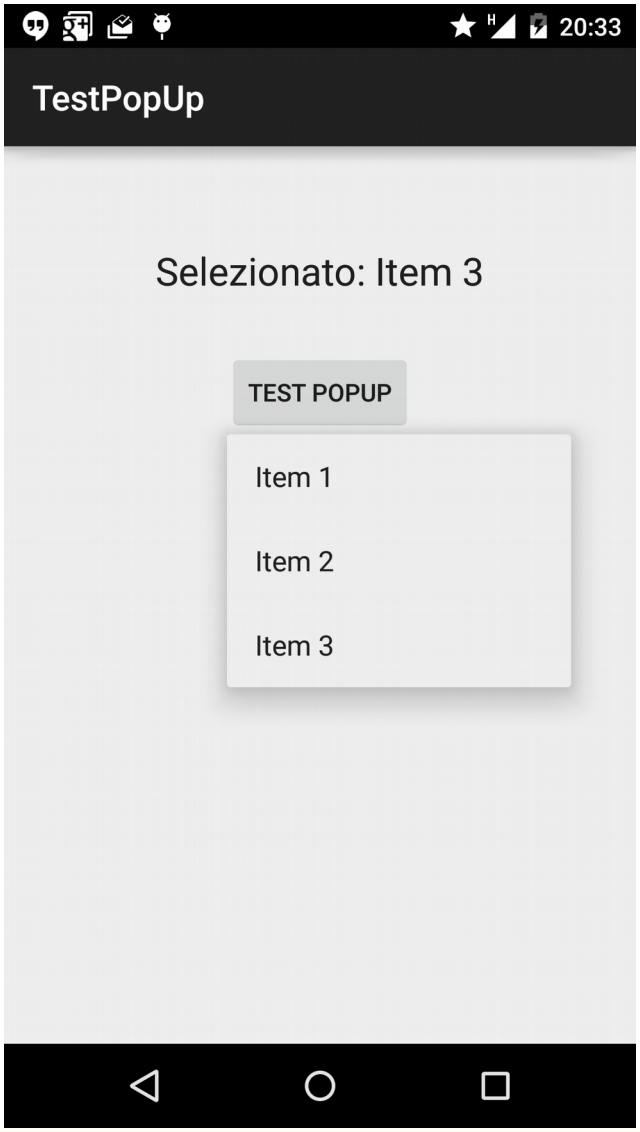
```
public class MainActivity extends ActionBarActivity {
    private TextView text;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        text=(TextView)findViewById(R.id.text);
    }
    public void showPopup(View v) {
        PopupMenu popup = new PopupMenu(this, v);
        MenuInflater inflater = popup.getMenuInflater();
        inflater.inflate(R.menu.popup, popup.getMenu());
        popup.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
            @Override
            public boolean onMenuItemClick(MenuItem item) {
                text.setText("Selezionato: "+item.getTitle());
                return true;
            }
        });
        popup.show();
    }
}
```

Nel layout:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Test popup"
    android:id="@+id/button"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="100dp"
    android:onClick="showPopup"/>
```

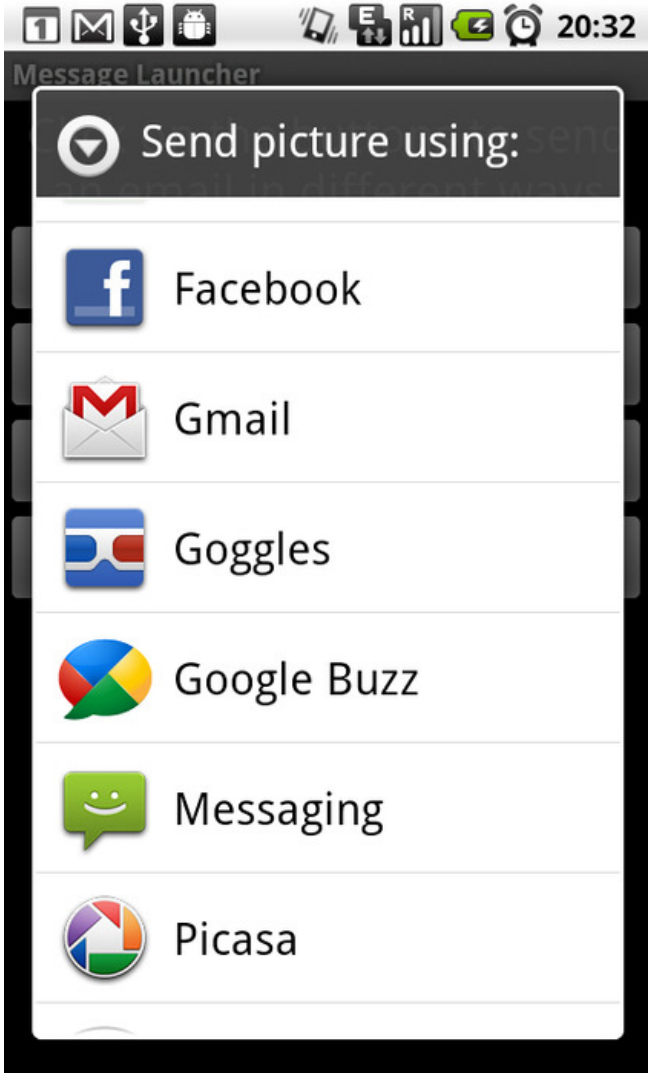
Reagiamo alla selezione con il solito listener

Popup menu


20:33
TestPopUp
Selezionato: Item 3
TEST POPUP
Item 1
Item 2
Item 3

Creazione dinamica di menu in base a Intent

- È possibile chiedere al sistema di riempire un *nostro* menu con tutte le **azioni** offerte da componenti del sistema sui dati che noi specifichiamo
- **Diverso** dal chiedere chi sono i componenti del sistema che possono completare un'azione che noi indichiamo



Creazione dinamica di menu in base a Intent

```
public boolean onCreateOptionsMenu(Menu m){
    super.onCreateOptionsMenu(m);

    // Crea l'Intent-filtro
    Intent in = new Intent(null, uriDati);
    // La categoria è di solito CATEGORY_(SELECTED_)ALTERNATIVE
    in.addCategory(Intent.CATEGORY_ALTERNATIVE);

    // Riempi il menu con le azioni trovate
    m.addIntentOptions(
         Menu.NONE,   // gruppo di menu a cui appartengono i nuovi item
         Menu.NONE,   // ID dell'item
         Menu.NONE,   // ordinamento
         getComponentName(),   // nome dell'attività
         null,   // array di Intent più specifici (gli item corrispondenti
verranno messi per primi nell'ordinamento)
         in, // intent-filtro preparato sopra
         0,      // flag vari (di tutto, di più)
         null);  // MenuItem[] in cui verranno messi, in ordine, gli item
corrispondenti agli intent più specifici (se ci sono!)
    return true;
}
```

Creazione dinamica di menu in base a Intent

- Spesso l'intent-filtro specifica il tipo di dato
  - Vengono inseriti tutti i receiver per quei dati, anche su azioni diverse
- Oppure, l'intent-filtro specifica un'azione
  - Vengono inseriti tutti i receiver in grado di compiere quell'azione, anche su dati diversi
- Oppure, si possono specificare entrambi, o anche in combinazioni
  - In effetti, si usa il processo di Intent resolution generico!

Creazione dinamica di menu in base a Intent

- Il menu costruito avrà tutti gli item associati con l'Intent “giusto” per far partire il receiver corrispondente
- L'icona sarà quella del receiver
- Se si usa un numero di gruppo, tutti gli item pre-esistenti nel menu con quel gruppo vengono cancellati
- Se ci sono più gruppi, viene inserito un divisore
- Se nessuno risponde all'intent-filtro... niente menu