User Interfaces

Views, Controls, and Layouts.

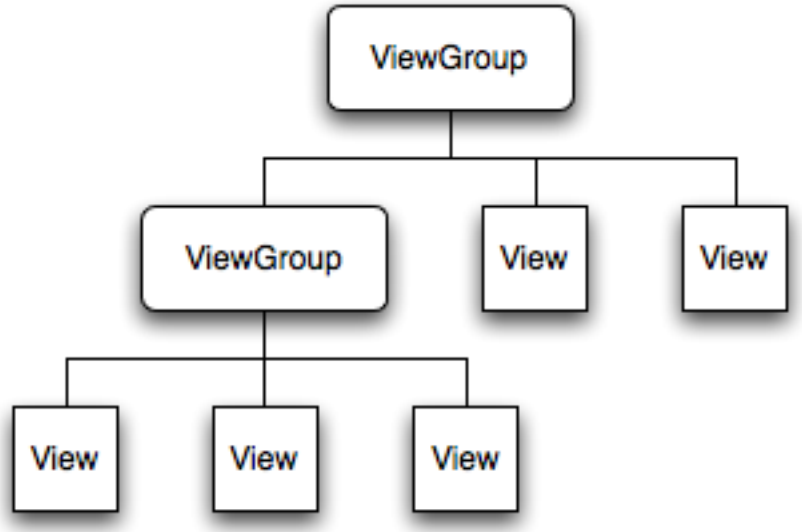
Preview

This is the content we'll see

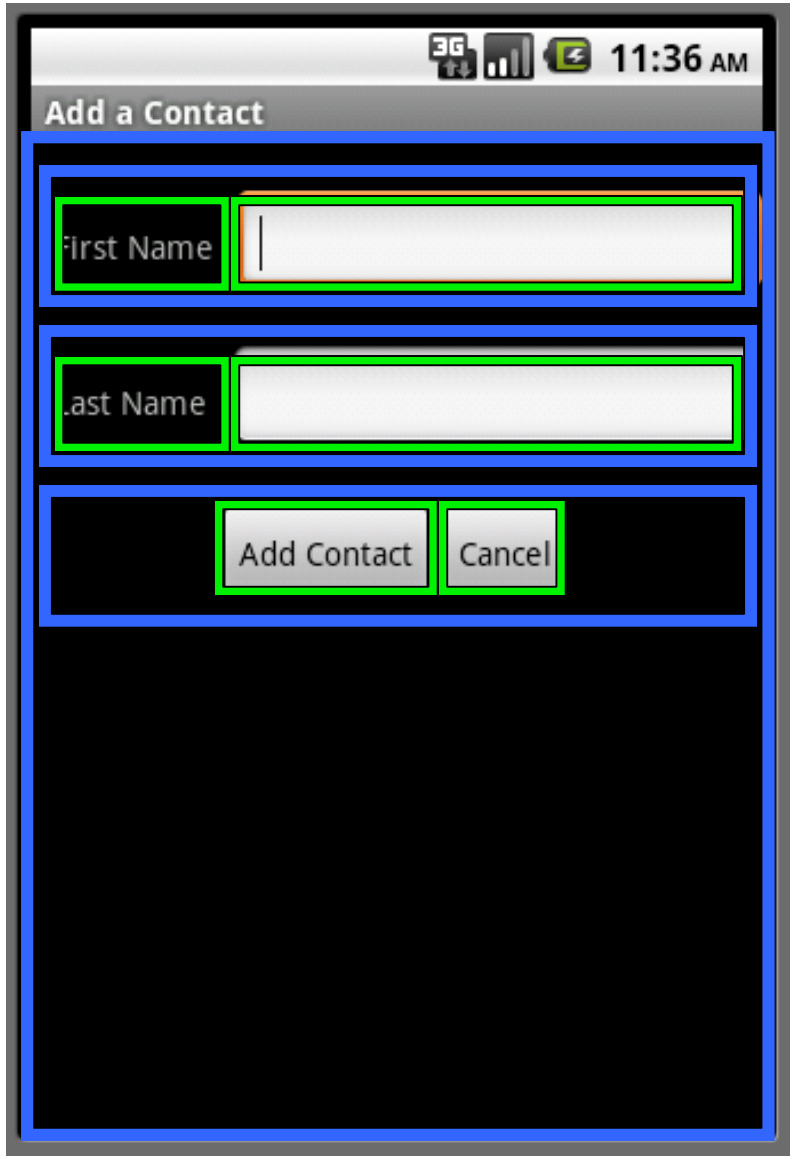
- Presentation
- XML Definition VS Java Definition
- ID Attributes
- Layouts
- Views
- List Adapters
- Events

Presentation

- A user interface is a set of graphical components like :
  - Button
  - Text
  - Form field
  - Component composed of other components…
- These components are called **Views**
- The last one is a special view called **ViewGroup**



Presentation

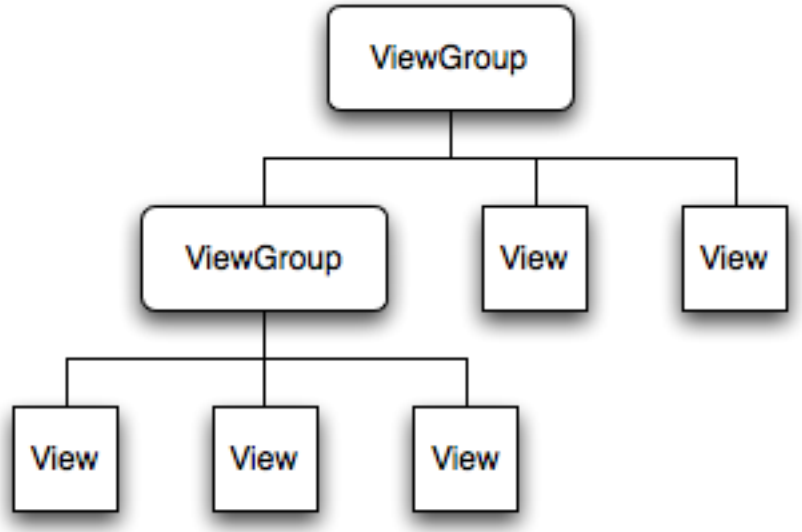


Views

ViewGroups

Presentation

- User interfaces can be defined :
  - In XML, inside a layout resource file
  - Directly in the Activity code
- We’re going to see both



XML Definition VS Java Definition

- Use of XML layout to define user interfaces is advised :
  - Separate interface structure and interface logic
  - Easier to maintain
  - Useful for **static** components
- But java definition can also be useful :
  - Adding components **dynamically**

User Interface

XML Definition : Example

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/first_name"
    />
  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/first_name"
    />

</LinearLayout>
```

Views

GroupView

Java Definition : Example

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

    LinearLayout layout = new LinearLayout(this);
    layout.setOrientation(LinearLayout.VERTICAL);
    layout.setLayoutParams(
                            new LayoutParams(

       LayoutParams.FILL_PARENT,

       LayoutParams.FILL_PARENT));

    TextView textView = new TextView(this);
    textView.setText(R.string.first_name);

    EditText editText = new EditText(this);

    layout.addView(textView);
    layout.addView(editText);

    setContentView(layout);
```

ID Attribute

- Ids are typically assigned in the layout XML files, and are used to retrieve specific views inside the Activity code.
- You can ask the ADT to generate a new ID using this special syntax :

Instead of `"@+id/resource_identifier"`

`"@id/resource_identifier"`

ID Attribute

- Example :

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/first_name"
    />
```

```
EditText txtFirstName =
                    (EditText)
findViewById(R.id.first_name);
```

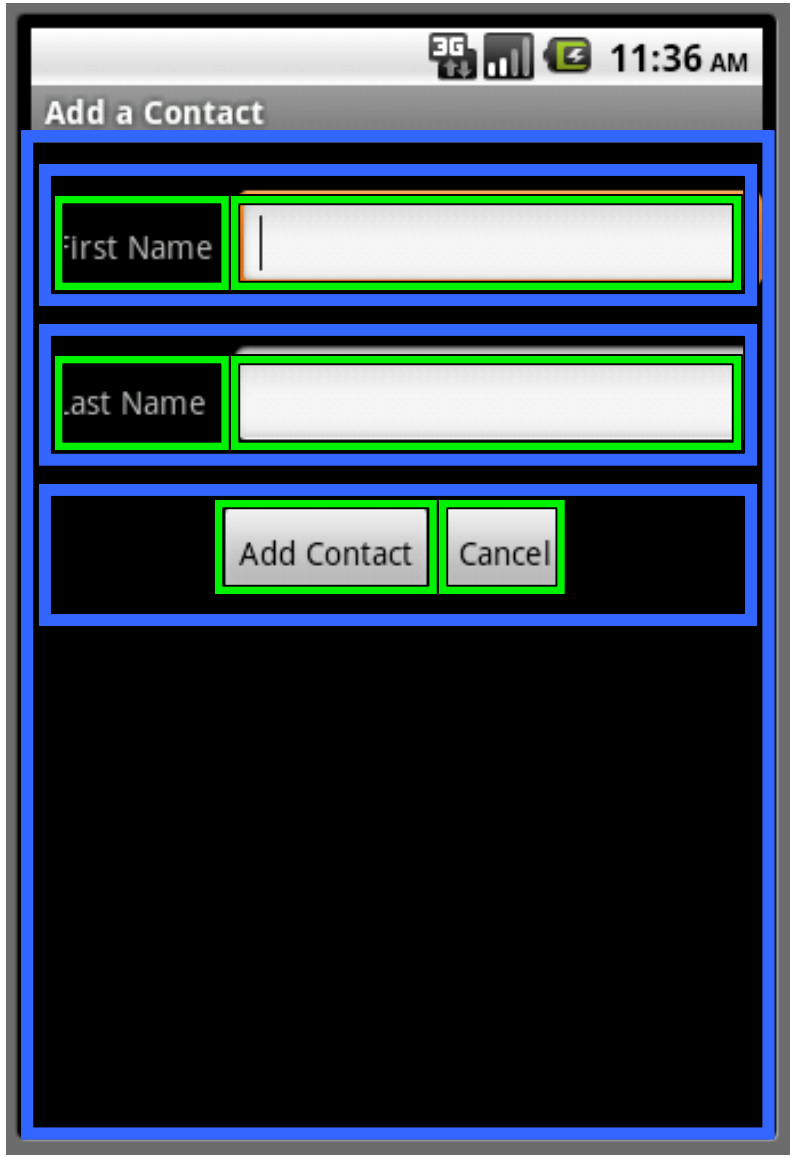
Layouts

- A layout is a ViewGroup used to position the views
- A layout is also a view
- A layout can contain other layouts
- Common layouts provided by the SDK are :
  - **LinearLayout**
  - **RelativeLayout**
  - **FrameLayout**
  - **TableLayout**

- We’re going to see only the first one, next ones later !

LinearLayout

- A Layout that arranges its children in a single column or a single row.
- This layout is the one mostly used in Android development
  - It can almost do everything other layouts can do by nesting layouts !



LinearLayout : Component Size

- Size can be defined :
  - In XML with **layout_width** and **layout_height** at

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="10px"
    />
```

  - In Java with a **LayoutParams** object

  - Value may be a dimension or one of the special symbolic constants :
    FILL_PARENT, **MATCH_PARENT**, **WRAP_CONTENT**

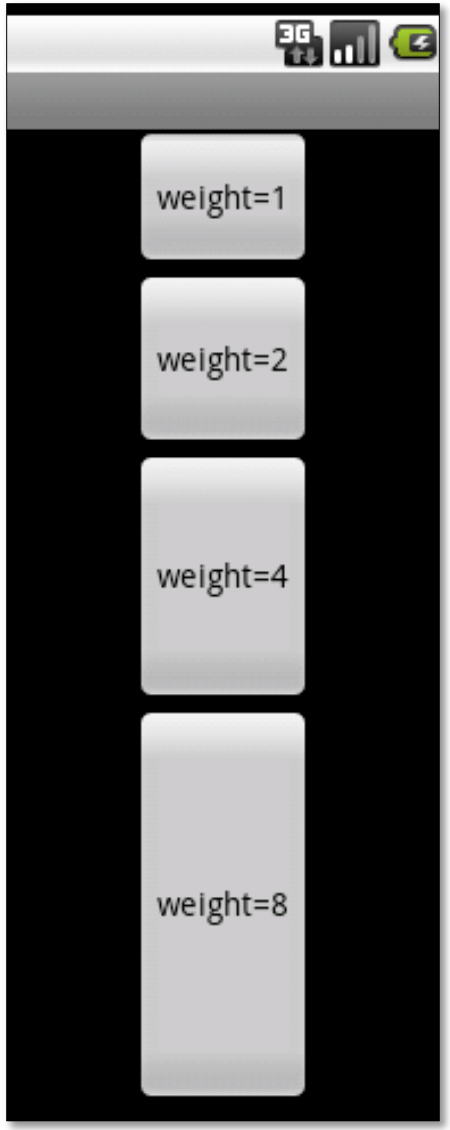
LinearLayout : Component Size

- **FILL_PARENT/MATCH_PARENT**
  View wants to be as big as its parent
- **WRAP_CONTENT**
  View want to be just big enough to hold its content

LinearLayout : Weight

- Defines how views share the layout size (ratio)
- Useful when you want several views to share all the screen
  - Example :

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="2"
    android:layout_text="weight=2"
    />
```



LinearLayout : Gravity

- Useful when the view is within a larger container
- Must be one or more (separated by '|') of the **Gravity** class constant values :
  - LEFT / RIGHT
  - TOP / BOTTOM
  - CENTER
  - And much more…look at class !

```
<TextView
    android:layout_width=”fill_parent"
    android:layout_height="fill_parent”
    android:gravity=”top|right"
    />
```

LinearLayout : Padding

- By default, components are tied to each other
- You can define space between them thanks to padding !
- Padding is defined as space between the edges of the view and the view's content
- Value in pixels
- Five padding attributes exist :
  - **padding**
  - **paddingLeft**
  - **paddingRight**
  - **paddingTop**
  - **paddingBottom**

LinearLayout : Padding

- XML example :

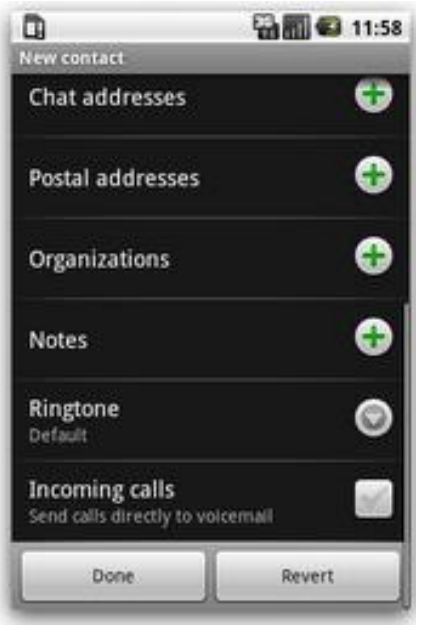
```
<TextView
    android:layout_width=”fill_parent"
    android:layout_height="fill_parent”
    android:padding=”20px"
    />
```

- Java

```
EditText txtFirstName = ... ;
// left, top, right, bottom
txtFirstName.setPadding(20, 30, 10, 20);
```

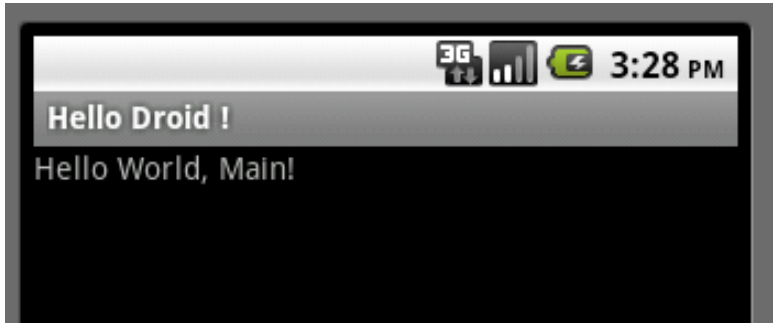
Views

- Android SDK offers many common components :
  - TextView
  - EditText
  - AutoCompleteTextView
  - RadioButton
  - CheckBox
  - Spinner
  - RatingBar
  - Button
  - …
- We're going to see them.



TextView

- Displays text to the user
- Can be editable
  - But disabled by default




```
<TextView
    android:layout_width=”fill_parent"
    android:layout_height="fill_parent”
    android:gravity=”top|right"
    />
```

EditText

- EditText is a subclass of TextView
  - Editable by default !

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/first_name"
    />
```

User Interface

CheckBox

- A check box is a two-states button that can be either checked or unchecked

```
<CheckBox
    android:id="@+id/checkbox”
    android:layout_width="wrap_content”
    android:layout_height="wrap_content”
    android:text="check it out"
    />
```

RadioButton

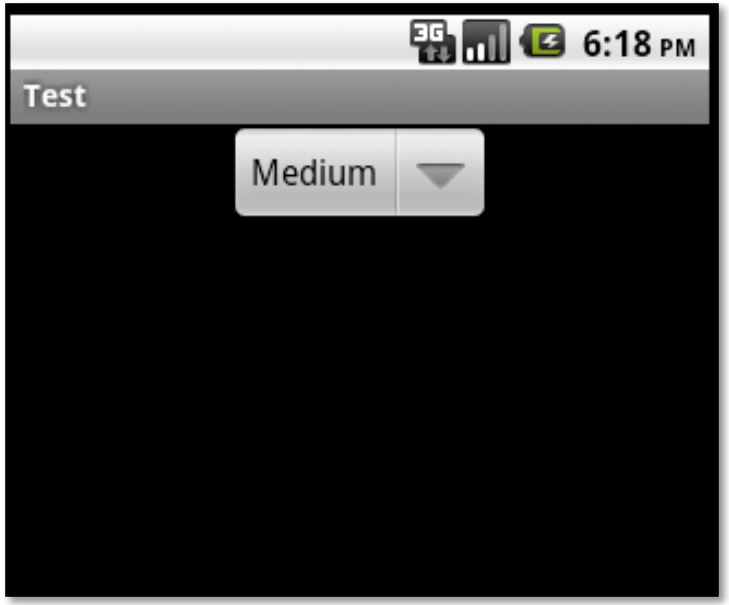
- A radio button is a two-states button that can be either checked or unchecked
- Contrary to a checkbox, only one button can be checked

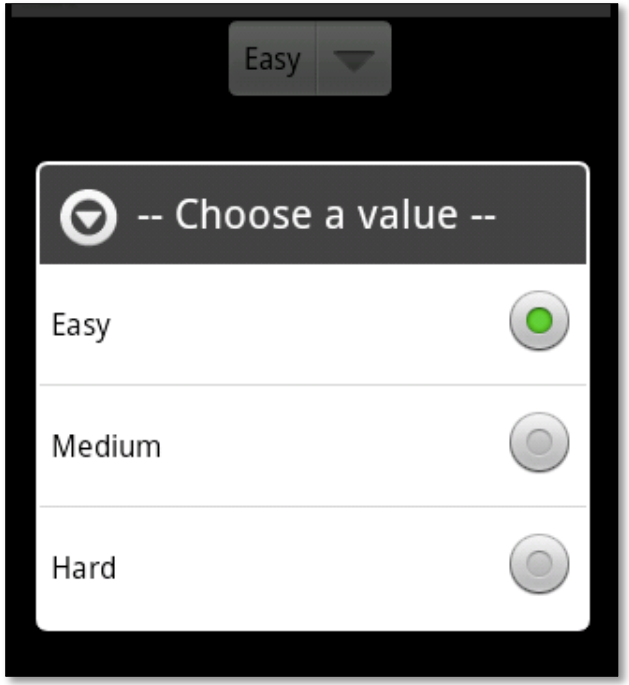
```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:id="@+id/radio_group"
    >

    <RadioButton
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Easy"
    />

    ...

</RadioGroup>
```

Spinner

- A spinner is the Android version of the combo box

```
<Spinner
    android:id="@+id/spinner"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:prompt="@string/planet_prompt"
    />
```





Spinner : Adapter

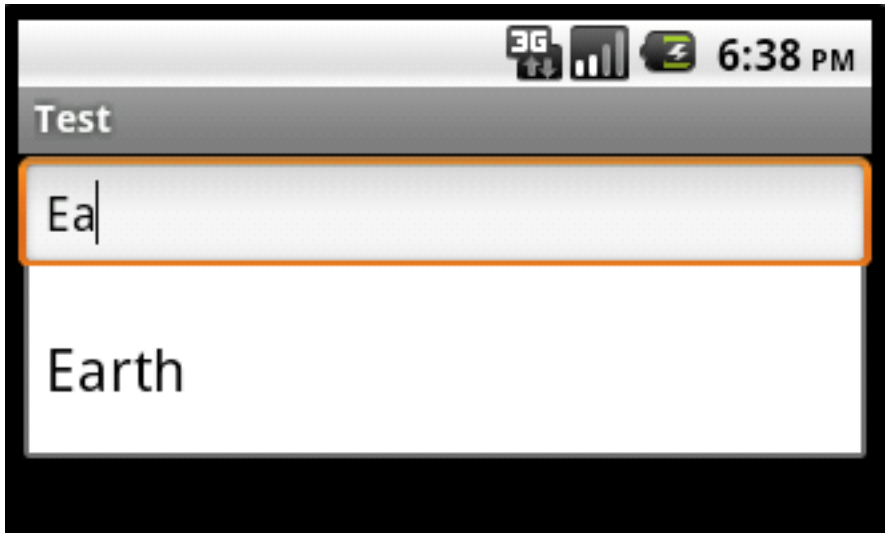
- To set spinner options, you need to use a **ListAdapter** object

```
String[] values = { "Easy", "Medium", "Hard" };

ListAdapter adapter =
             new ArrayAdapter<String>(this,

      android.R.layout.simple_spinner_item,
                                  values);

adapter.setDropDownViewResource

      (android.R.layout.simple_spinner_dropdown_item);

Spinner spinner = (Spinner) findViewById(R.id.spinner);
spinner.setAdapter(adapter);
```

AutoCompleteTextView

- An editable text view that shows completion suggestions automatically while the user is typing

```
<AutoCompleteTextView
    android:id="@+id/autocomplete_planet”
    android:layout_width=”fill_parent”
    android:layout_height="wrap_content”
    />
```



AutoCompleteTextView : Adapter

- To set AutoCompleteTextView options, you need to use a **ListAdapter** object again

```
String[] values = { "Mercury","Venus","Earth","Mars" };

ListAdapter adapter =
            new ArrayAdapter<String>(this,

      android.R.layout.simple_dropdown_item_1line,
                                  values);

AutoCompleteTextView textView = (AutoCompleteTextView)

      findViewById(R.id.autocomplete_planet);
textView.setAdapter(adapter);
```

User Interface

RatingBar

- A RatingBar is a component which represent a rating in stars
- Two special attributes :
  - **numStars** : the number of stars to disp
  - **stepSize** : the number equivalent to one star

In

```
<RatingBar
    android:id="@+id/rating_bar”
    android:layout_width=”wrap_content”
    android:layout_height="wrap_content”
    android:numStars="5”
    android:stepSize="1.0"
    />
```

Button

- Represents a push-button widget
- Push-buttons can be pressed, or clicked, by the user to perform an action

```
<Button
    android:id="@+id/my_button”
    android:layout_width=”wrap_content”
    android:layout_height="wrap_content”
    android:text=“@string/button_text”
    />
```

ImageButton

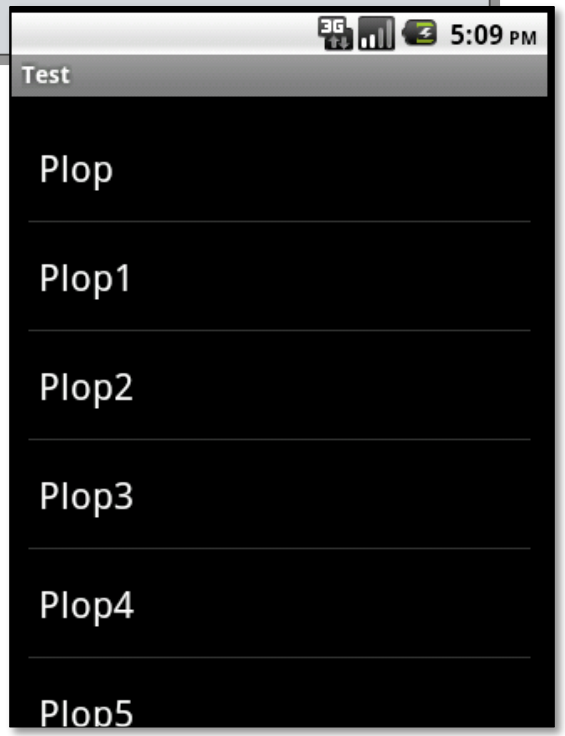
- Represents a push-button widget but with an image instead of text inside

```
<ImageButton
    android:id="@+id/my_button”
    android:layout_width=”wrap_content”
    android:layout_height="wrap_content”
    android:src="@drawable/logo_google"
    />
```

ListView

- A view that shows items in a vertically scrolling list

```
<ListView
    android:id="@+id/my_list_view”
    android:layout_width=”fill_parent”
    android:layout_height=”fill_parent”
    />
```



ListView : Adapter

- To populate the list, you need to use an **ListAdapter** object again

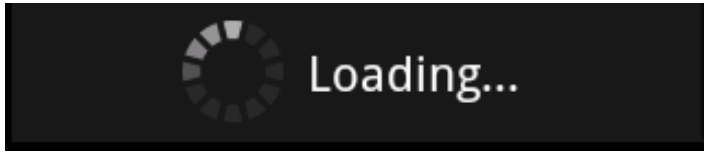
```
ListView listView =
        (ListView)findViewById(R.id.my_list_view);
Cursor cursor = new PersonDao(this).getAllPersons();
ListAdapter adapter =
                    new SimpleCursorAdapter(this,

android.R.layout.simple_list_item_1,
                                        cursor, new String[]
{ "name" },
                                          new int[] {
android.R.id.text1 });
listView.setAdapter(adapter);
```

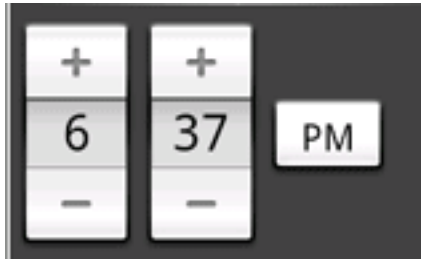
More about ListAdapter…

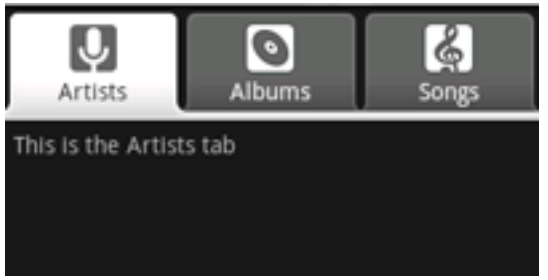
- The bridge between a component and the data that backs the list
- The most used concrete subclasses are :
  - **ArrayAdapter**
    - Adapter to map object arrays or object lists to a view
  - **SimpleCursorAdapter**
    - Adapter to map columns of a cursor to a view
    - We'll see more about cursors later…
- Constructors of these classes take a resource id :
  - The layout to apply to the item of the view
    - You can use one of proposed by the SDK
    - You can define your own layout
- Remember : **android.R ≠ R**

Other views

- Now you understand the principle
- Go to see the Android Documentation for more information
- You will see many more views :
  - ImageView
  - WebView
  - GridView
  - DatePicker
  - DigitalClock
  - ProgressBar
  - ToggleButton
  - VideoView
  - …







Events

- With Android, all user actions are events
  - Click
  - Long click
  - Key pressed
  - Item selected
  - …
- You can link behaviours to these events
- The interception mechanism based on the Listener notion
  - As with Swing !

Click Event

- To add a listener to a click event on a view :
  - **setOnClickListener(View.OnClickListener)**
- OnClickListener is an inner interface of the View class
- You have three possibilities :
  - Make your activity implement it
  - Create a new class implementing it
  - Create an anonymous class

Click Event

- First solution :

```
public class MyActivity extends Activity
                                                implements
View.OnClickListener {

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Button button = (Button) findViewById(R.id.my_button);
    button.setOnClickListener(this);
  }

  public void onClick(View view) {
    // Display a notification popup during 1 second.
    Toast.makeText(this, "Button clicked !", 1000).show();
  }

}
```

Click Event

- Second solution :

```
public class MyActivity extends Activity {

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Button button = (Button) findViewById(R.id.my_button);
    button.setOnClickListener(new ButtonClickListener());
  }
}
```

```
public class ButtonClickListener
                                              implements
View.OnClickListener {

  public void onClick(View view) {
    // Display a notification popup during 1 second.
    Toast.makeText(this, "Button clicked !", 1000).show();
  }
}
```

Click Event

- Third solution :

```
public class MyActivity extends Activity {

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Button button = (Button) findViewById(R.id.my_button);

    button.setOnClickListener(new View.OnClickListener() {

      public void onClick(View view) {
        // Display a notification popup during 1 second.
        Toast.makeText(MyActivity.this, "Clicked!", 1000)
                                .show();
      }

    });
  }

}
```

Other Events

- All events are based on the same principle
- Some have to return if the event has been consumed
  - If true, the event does not fire other listeners

```
EditText editText = (EditText) findViewById(R.id.my_text);

editText.setOnTouchListener(new View.OnTouchListener() {

   public boolean onTouch(View view, MotionEvent e) {
      Toast.makeText(MyActivity.this, "Touch!", 1000)
                              .show();
      return true;
      // True means the listener has consumed the event.
   }

});
```