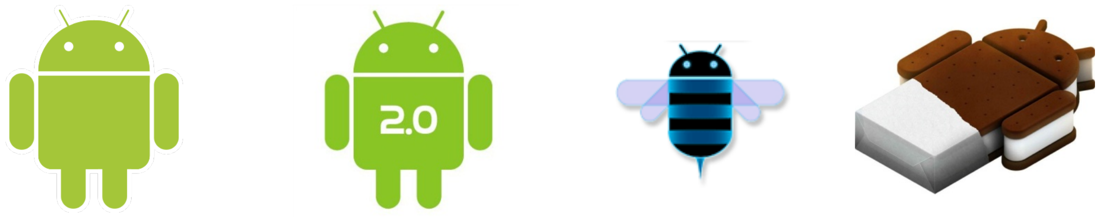


Programming with Android: Widgets and Events

Luca Bedogni **Marco Di Felice**

Dipartimento di Scienze dell'Informazione

Università di Bologna

Outline

Android: Where are we now …

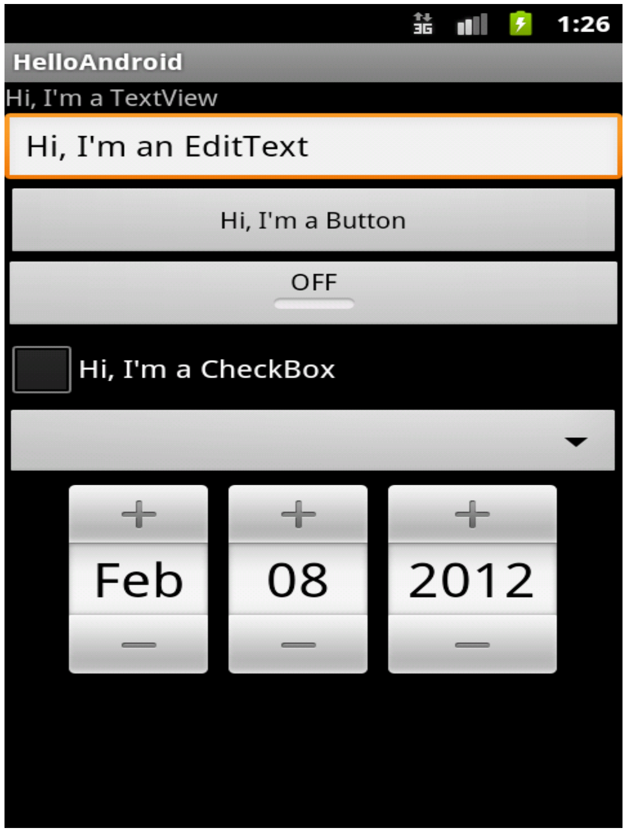
Android Applications' anatomy:

- **Activities** → Application Components (screens)
- **Intents** → Communication between components
- **Layouts** → Placement of the elements on the screen …
- **Views** → … Elements to be placed!

Widget → *Pre-defined, common-used View objects …*

Widgets: some examples …

- TextView
- EditText
- Button
- ToggleButton
- CheckBox
- Spinner
- DatePicker
- Custom Views



Widgets: Java and XML code

- **Widgets** can be created in **Java**
- Widgets can be created in **XML** and accessed through **Java**

XML

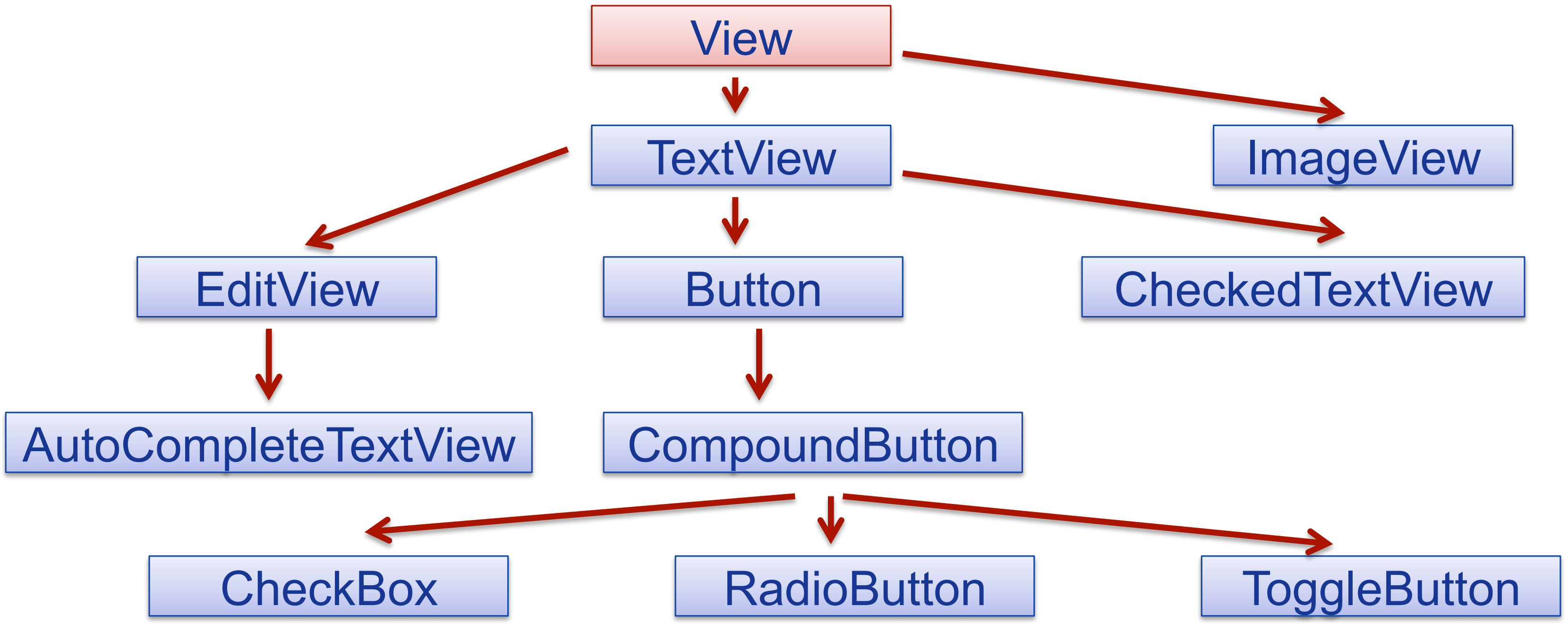
```
< TextView
      android:id="@+id/name1"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"  />
```

JAVA

```
public TextView text;
text=(TextView)findViewById(R.id.name1);
```

Widgets: Hierarchy of the classes …

- **Widgets** are organized on a *hierarchy* of classes …



Widgets: TextView

- **XML** tags: **<TextView> </TextView>**

- Could be filled with **strings**, HTML **markups**
- Should specify which type of text is displayed
  - Not directly editable by users
- Usually used to display **static** informations

- Some methods:
  - void **setText**(CharSequence text)
  - CharSequence **getText**();

Widgets: Linkify elements

- Simple **strings** that could be **linkified** automatically.
- How? Pick a normal string, and use **Linkify.addLinks()** to declare what kind of link should be created.
- Could manage: *Web addresses*, *Emails*, *phone numbers*, *Maps*

```
TextView textView=(TextView) findViewById(R.id.output);
Linkify.addLinks(textView, Linkify.WEB_URLS |
                           Linkify.WEB_ADDRESSES |
                           Linkify.PHONE_NUMBERS );
Linkify.addLinks(textView, Linkify.ALL);
```

- It is possible to define **custom** Linkify objects. ..

Widgets: TextView methods

- **Methods** to place the text inside a TextView …
  - public void **setSingleLine**(boolean singleLine)
  - public void **setHorizontallyScrolling**(boolean wether)
  - public void **setLines**(int lines)
  - public void **setEllipsize**(TextUtils.TruncateAt where)
  - public void **setHints**(CharSequence hint)

  - TextUtils.TruncateAt.**END**
  - TextUtils.TruncateAt.**MARQUEE**
  - TextUtils.TruncateAt.**MIDDLE**
  - TextUtils.TruncateAt.**START**

Widgets: EditText

- Similar to a TextView, but **editable** by the users
  - Used to get information by the users.

- It is possible to declare in the layout file which type of text will be contained … (NORMAL, EDITABLE, SPANNABLE)

- An appropriate **keyboard** and **display** will be used.

- Text selection methods:

  public void **setSelection**(int index)

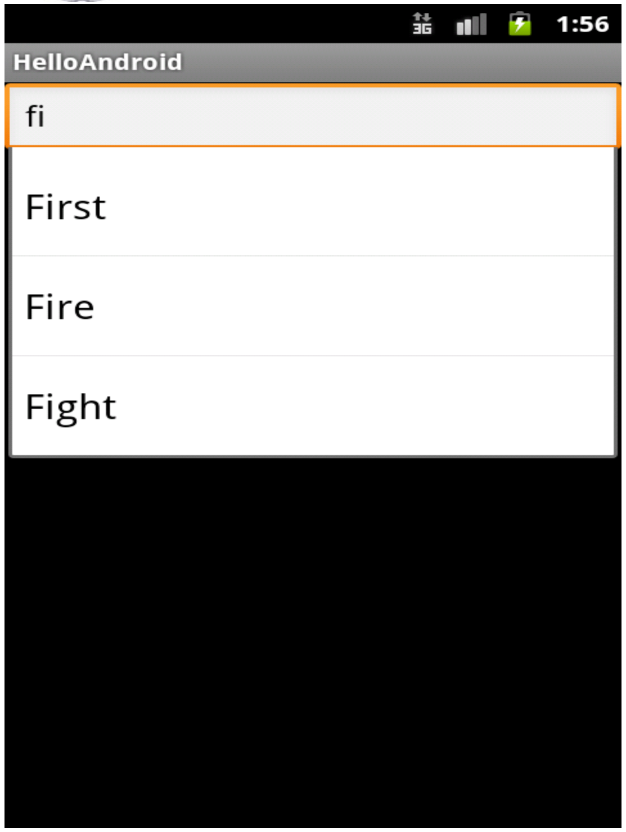
  public void **setSelection**(int start, int end)

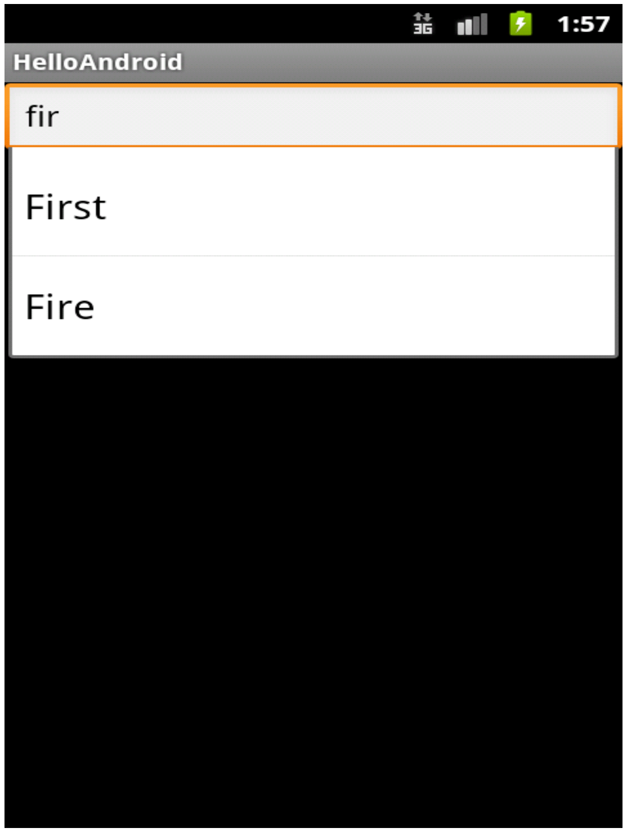
Widgets: AutocompleteTextView

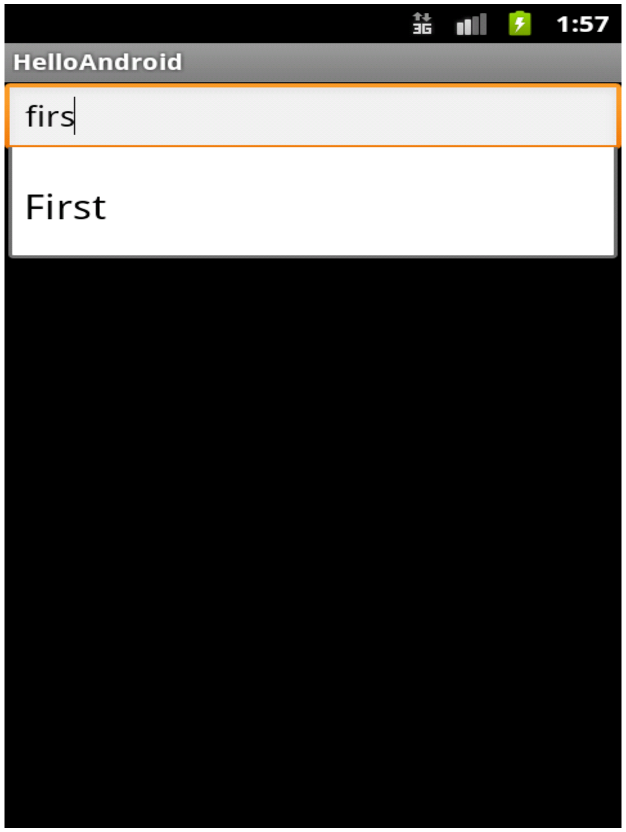
- Defined through tag: **<AutoCompleteTextView>**
- Used to ease the input by the users …
  - As soon as a user starts to type something, hints will be displayed
- A list of hints is given via an *Adapter*

```
 String[] tips=getResources().getStringArray(R.array.nani_array);
 ArrayAdapter<String> adapter=new ArrayAdapter(this,
android.R.layout.simple_dropdown_item_1lines, tips);
 AutoCompleteTextView acTextView=(AutoCompleteTextView) findViewById
(R.id.inputText);
acTextView.setAdapter(adapter);
```

Widgets: AutocompleteTextView



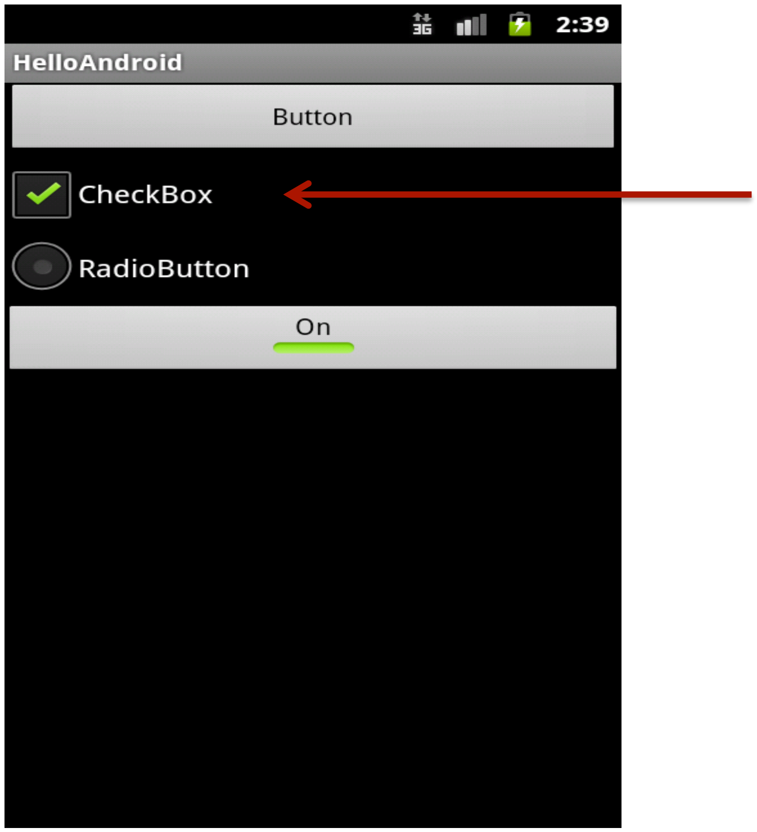




Widgets: Button and CompoundButton

- Not really different to manage than a **Textview**!
- Has events related to clicks, long clicks and so on
- Cannot be directly editable by users
- **CompoundButton**: Button + *state* (checked/unchecked)
  - Subclasses: **CheckBox**, **RadioButton**, **ToggleButton**
  - Methods: public void **setChecked**(boolean checked)
    public void **toggle**()

Widgets: Button and CompoundButton

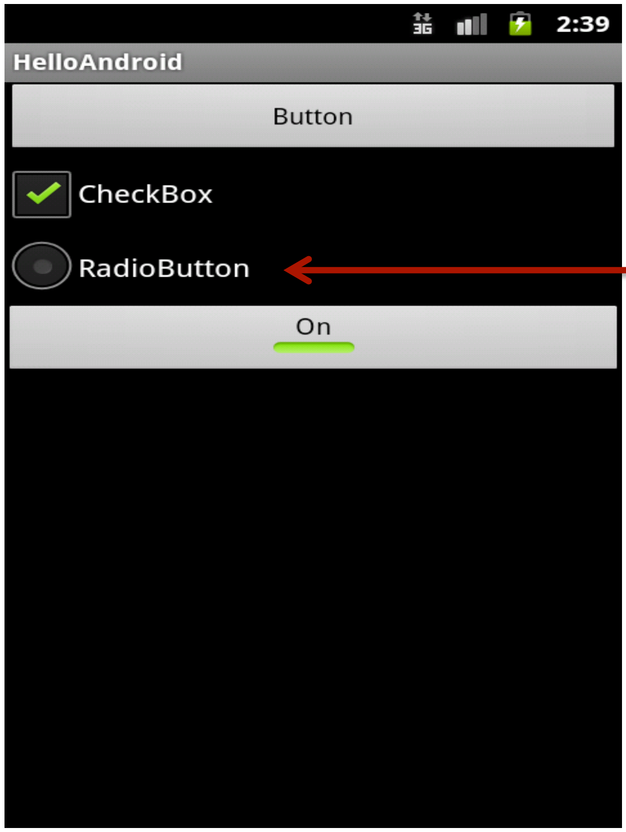


checkBox CompoundButton

public boolean **isChecked**(): return true if the button is checked, false otherwise.

Listener: onCheckedChangeListener

Widgets: Button and CompoundButton



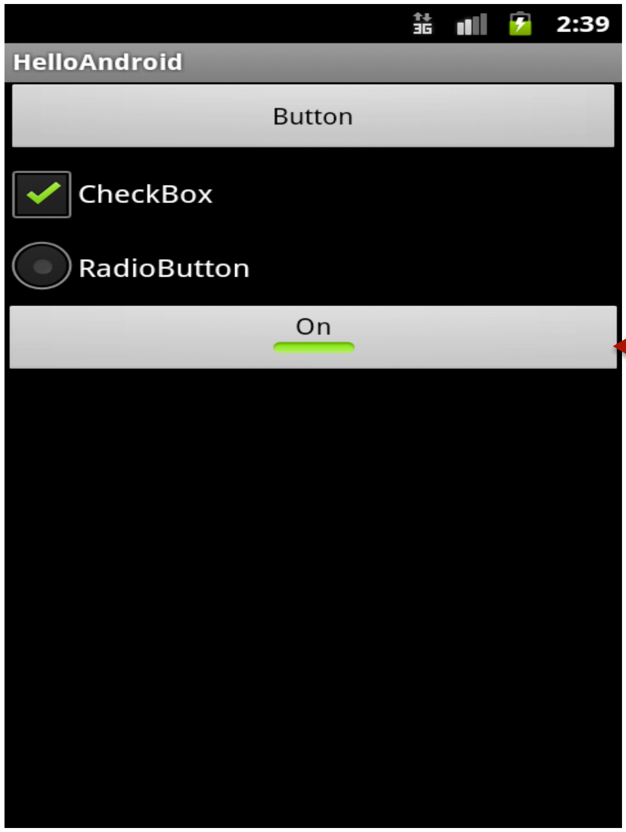
radioButton CompoundButton

Define multiple (mutual-exclusive) options through a <RadioGroup> </RadioGroup> tag.

Only one button can be checked within the same RadioGroup.

Listener: OnCheckedChangeListener

Widgets: Button and CompoundButton



toggleButton CompoundButton

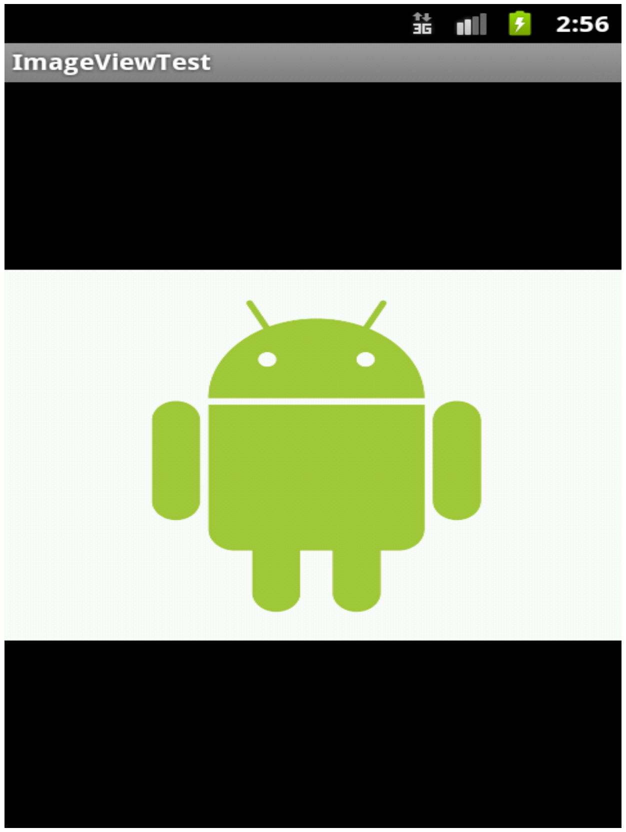
It can assume only 2 states: *checked/unchecked*

Different labels for the states with: **android:textOn** and **android:textOff** XML attributes.

Listener: OnCheckedChangeListener

Widgets: ImageView

- **ImageView** is a subclass of the View object. XML Tag: **<ImageView>**
- Images inside res/drawable (or obtained with other methods)
- Some methods to manipulate it:
  void **setScaleType**(enum scaleType)
  void **setAlpha**(double alpha)
  void **setColorFilter**(ColorFilter color)



Widgets: CheckedTextView

- **Checkable** version of a TextView

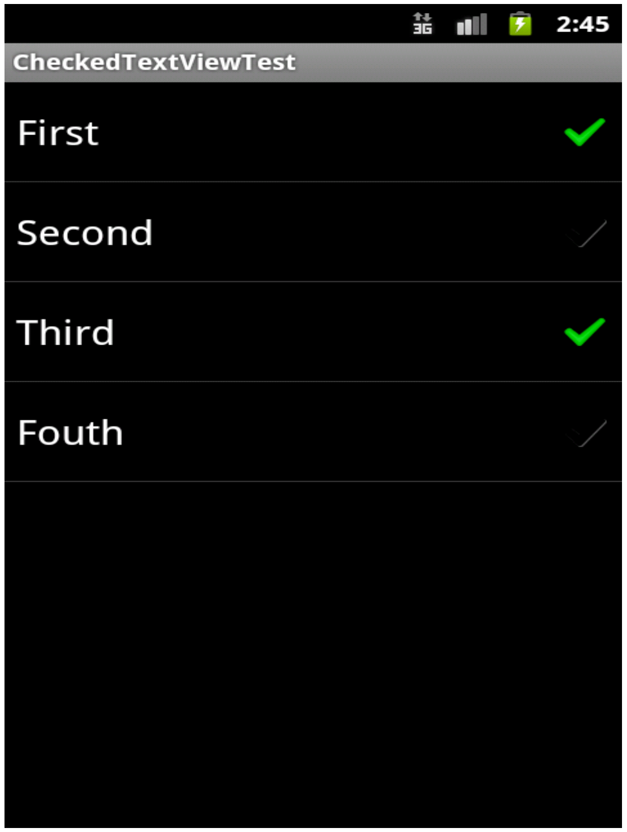
- Usable with a **ListView Adapter**
  - *Multiple* or *single* selection of items
    (CHOICE_MODE_SINGLE, CHOICE_MODE_MULTIPLE)

- Methods:

  void setChoiceMode(int choiceMode)

  long[] getCheckItemIds()

  int getCheckedItemPosition()



Widgets: Other elements …

See the official documentation for the complete list:

- **AnalogClock** Widget
- **DigitalClock** Widget
- **DataPicker** Widget
- …. … …

- As an alternative, it is possible to create **custom View** …
  - Ad hoc components, code reuse …

Views and Events

- The users **interacts** with the Views …
- … Upon certain action, an appropriate **event** will be fired
- Reacting to this events makes the activity interactive
- Events for click, long click, gestures, focus, external events …
- **PROBLEM**: How to **handle** these events?

Views and Events

- **Two** ways to **handle** the View events:

1. Events Handlers.

Some Views have callback methods to handle specific events.

when a **Button** is touched → **onTouchEvent**() called

Es.

```
boolean onKeyDown(int keycode, KeyEvent event)
boolean onKeyUp(int keycode, KeyEvent event)
boolean onKeyMultiple (int keycode, KeyEvent event)
….
```

Views and Events

***1. Events Handlers*.**

Some views have **callback** methods to handle specific events.

when a **Button** is touched → **onTouchEvent**() called

PROBLEM: to intercept an event, you must extend the View class and override the callback method … not very practical!

- In practice: use *Events Handlers* for custom components …
- … use *Events Listeners* for common View/Widget components …

Views and Events

***1. Events Listeners*.**

- A View class contain a collection of nested **interfaces** (**listeners**).
- Each interface handles a single type of events…
- Each interface contains a single **callback** method …
- This method is called in occurrence of the event of the View.

Views and Events: ActionListener

Some ActionListeners:

- **OnClickListener**

  method: *onClick()*
- **OnLongClickListener**

  method: *onLongClick()*
- **OnFocusChangeListener**

  method: *onFocusChange()*
- **OnKeyListener**

  method: *onKey()*

Views and Events: ActionListener

More ActionListener:

- **OnCheckedChangeListener**
  method: *onCheckedChanged()*

- **OnTouchListener**
  method: *onTouch()*

- **OnCreateContextMenuListener**
  method: *onCreateContextMenu()*

Views and Events: ActionListener

To handle events through ActionListener:

1. Implement the **callback** method
2. Define an ActionListener object as an anonymous class
3. Pass an instance of the ActionListener implementation to the View through the View.setOnXXXEventListener() method

```
Button btn = (Button)findViewById(R.id.btn);
btn.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        // Event management
    }
});
```

Views and Events: ActionListener

To handle events through ActionListener:

1. Implement the **callback** method
2. *Implement the nested interface in the Activity*
3. Pass an instance of the ActionListener implementation to the View through the View.setOnXXXEventListener() method

```
public class ExampleActivity extends Activity implements OnClickListener {
 …
   Button button=(Button)findViewById(R.id.buttonNext);
   button.setOnClickListener(this);
   …
  public void onClick(View v) { }
}
```

Views and Events: ActionListener

- Possible to perform some events in the code
- Tipically in the form **performSomething**()
- Used to simulate an event
- The corresponding listener (if set) will be fired …
- Give a result (true or false) depending if the component has a listener or not!