Chapter 6. Widgets and Events

OUTLIN

- What is a **Widget**?
- **Widget:** TextView and EditText
- **Widget**: Button and CompoundButton
- **Widget**: ImageView
- **Widget**: CheckedTextView
- **Event** Management: Event **Handlers**
- **Event** Management: Event **Listeners**

Android: Where are we now …

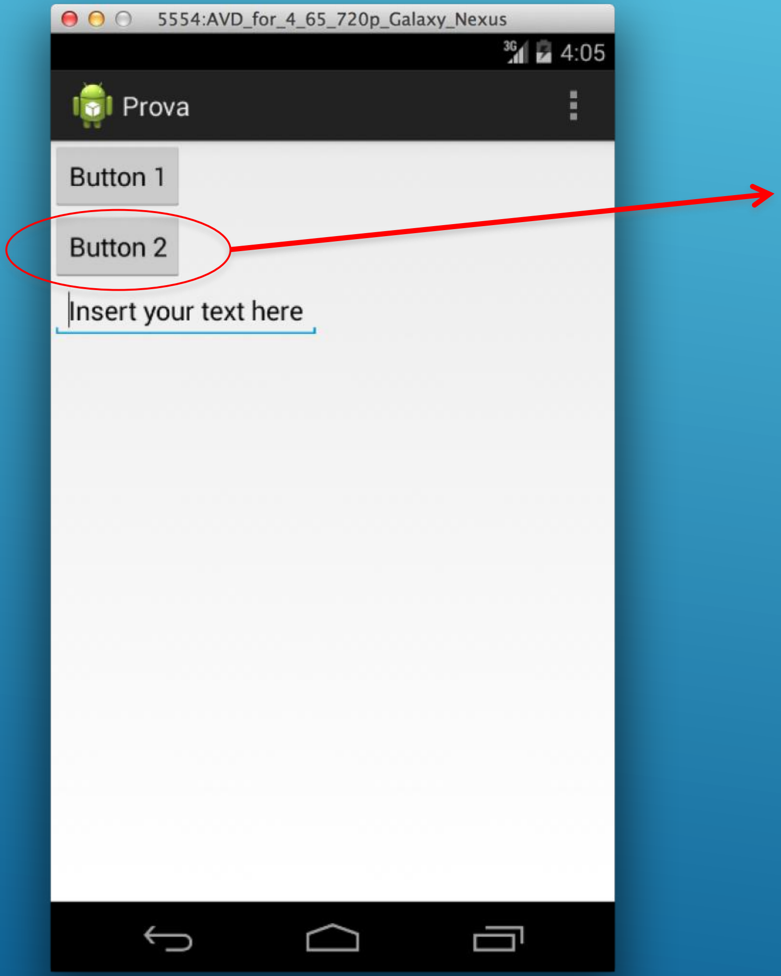
Android Applications' anatomy:

- **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 …*

Android: Views objects

Views → basic building blocks for user interface components



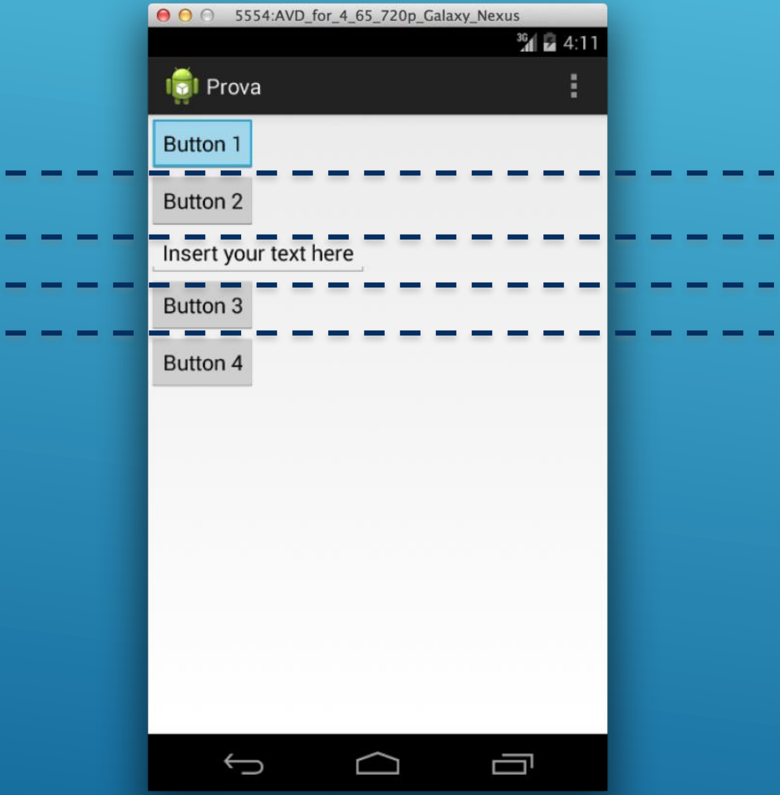
✧ Rectangular area of the screen
✧ Responsible for **drawing**
✧ Responsible for **event handling**

EXAMPLEs of **VIEWS** objects:

- GoogleMap
- WebView
- **Widgets** → topic of the day
- …
- User-defined Views

Android: Views objects

ViewGroup → Container of other views, base class for **layouts**



LINEAR LAYOUT

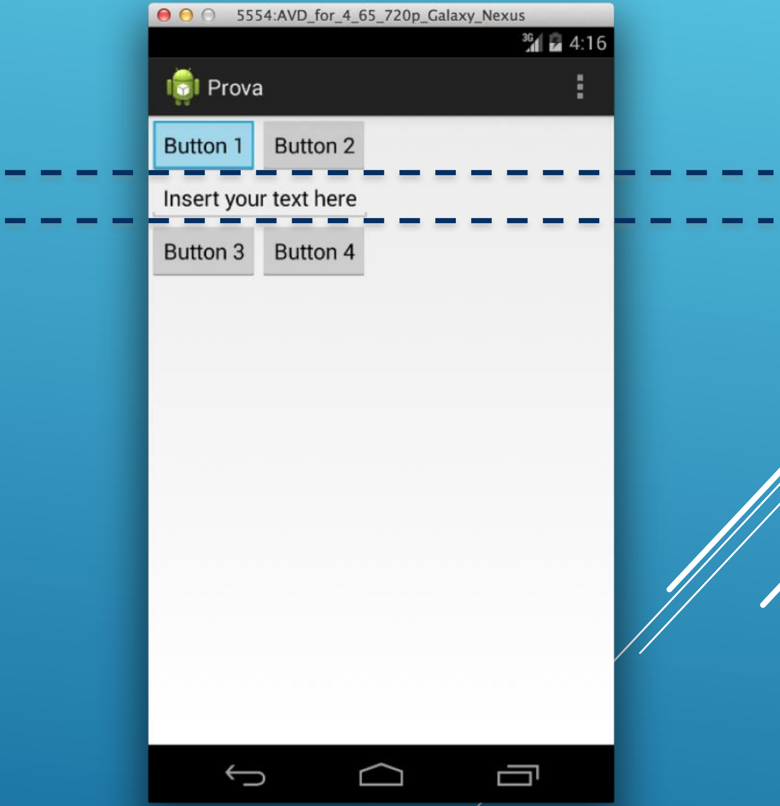
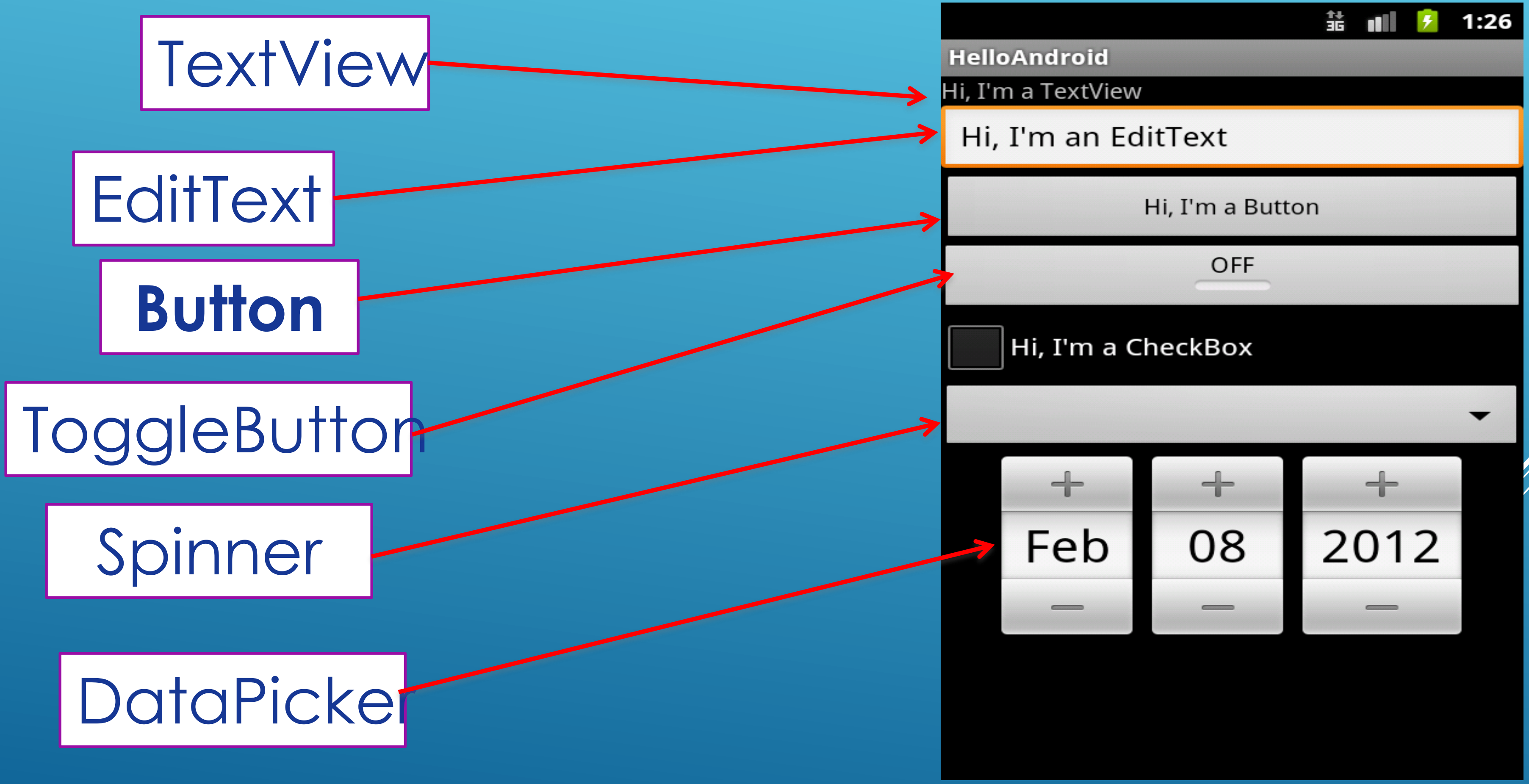


TABLE LAYOUT

Android: Views objects

Widget→ Pre-defined interactive UI components (android.view.widg



Widgets: Java and XML code

- Widgets can be created in the **XML layout files**

```
< TextView
        android:id="@+id/textLabel”
        android:width=“100dp”
        android:height=“100dp”
        android:layout_width="match_parent"
        android:layout_height="wrap_content”
        android:visibility="visible”
        android:enabled="true”
        android:scrollbars="vertical”
        ….
/>
```

Widgets: Java and XML code

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

XML

```
< TextView
      android:id="@+id/name1” />
```

JAVA

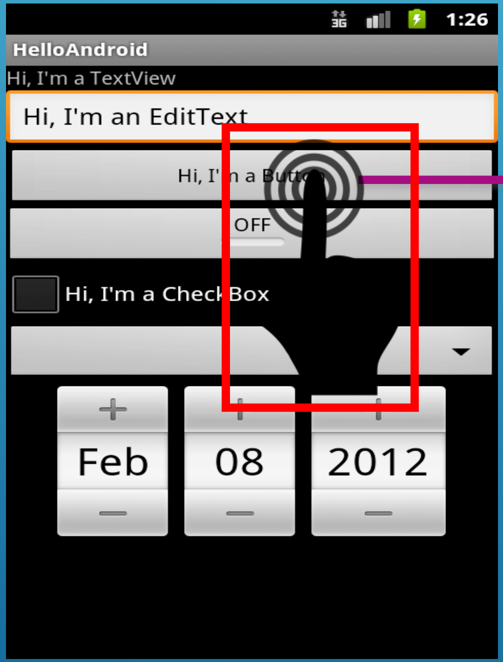
```
public TextView text;
text=(TextView)findViewById(R.id.name1);
```

CAST REQUIRED

```
public TextView text;
text=new TextView();
```

Widgets: Java and XML code

- Each Widget can generate events, that can be captured by **Listeners** that define the appropriate actions to be performed in response to each event.



ONCLICK event

Java code that manages the **onClick** event …

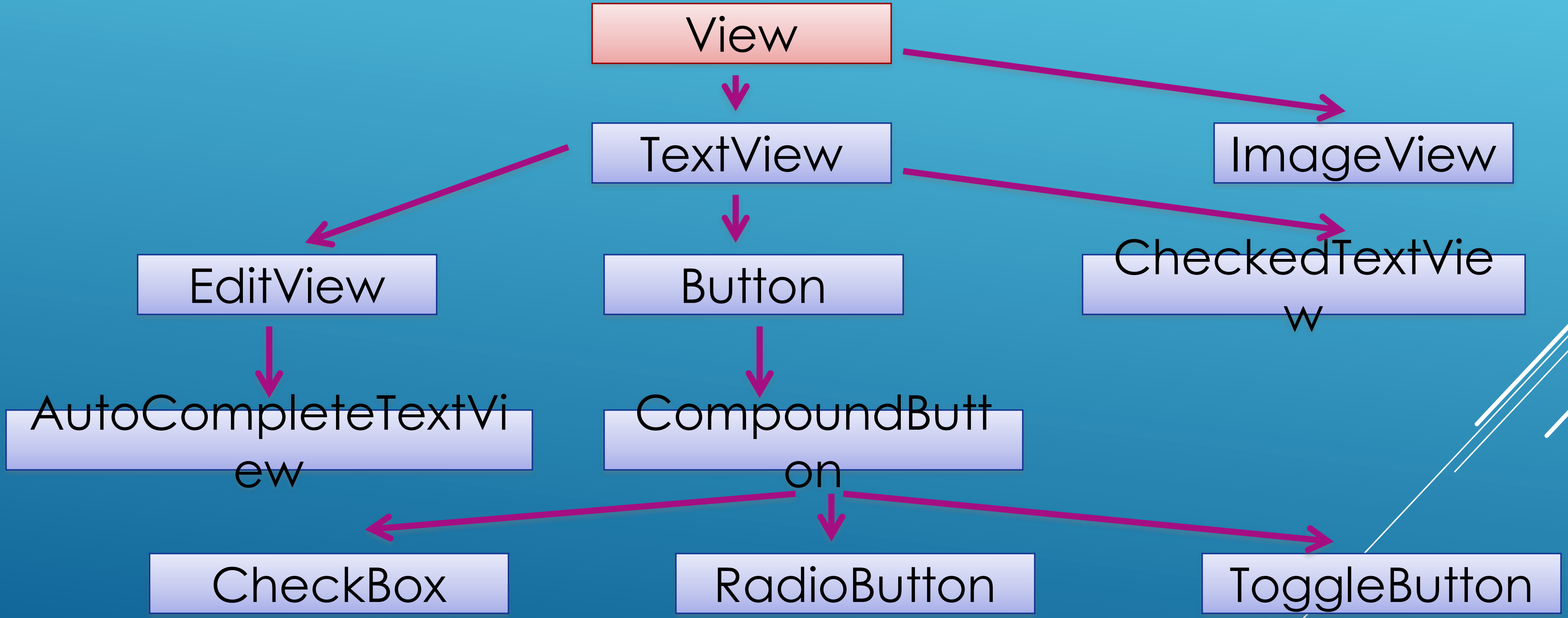
Widgets: Java and XML code

- Each Widget can have a **focus** and a **visibility**, based on the user's interaction.
- The user can force a focus to a specific component through the **requestFocus()** method.
- The user can modify the visibility of a specific component through the **setVisibility(int)** method.

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

Widgets: Hierarchy of the classes …

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



Widgets: TextView

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

- Could be filled with **strings** or HTML **markups**
- Not directly editable by users
- Usually used to display **static** informations

```
 <TextView
        android:text=”@string/textWelcome"
        android:id="@+id/textLabel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content”
/>
```

Widgets: TextView methods

- **Methods** to place some texts inside a TextView …
  - public void **setText**(CharSequence text)
  - public CharSequence **getText**()
  - public void **setSingleLine**(boolean singleLine)
  - public void **setHorizontallyScrolling**(boolean enable)
  - public void **setLines**(int lines)
  - public void **setEllipsize**(TextUtils.TruncateAt where)
  - public void **setHints**(CharSequence hints)

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

Widgets: Linkify elements

- Simple **strings** could be **linkified** automatically.
- How? Pick a normal string, and use **Linkify.addLinks()** to

```
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: EditText

➢ **XML** tags: **<EditText> </EditText>**

✧ Similar to a TextView, but **editable** by the users

✧ An appropriate **keyboard** will be displayed

```
 <EditText
       android:text=”@string/textDefault"
       android:id="@+id/editText”
       android:inputType= “textCapSentences” | “textCapWords” |
                              “textAutoCorrect” | “textPassword” |
                         “textMultiLane” | “textNoSuggestions”
/>
```

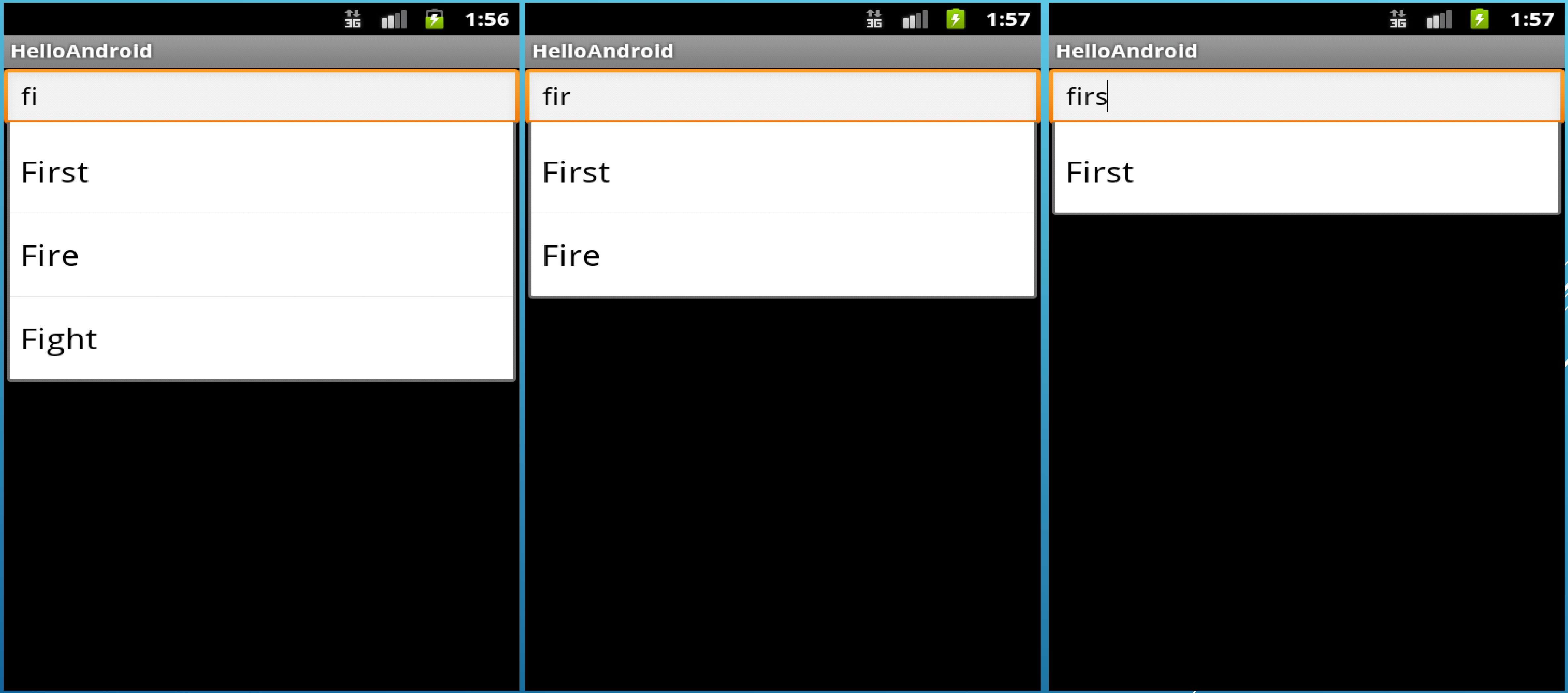
Widgets: AutocompleteTextView

- **XML** tags: **<AutoCompleteTextView> </Auto…View>**

- Used to make easier the input by the users …
  - As soon as the user starts typing, hints are displayed
- A list of hints is given through 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

- **XML** tags: **<Button> </Button>**
  - Superclass of a TextView, but not directly **editable** by users
  - Can generate events related to click, long click, drag, etc

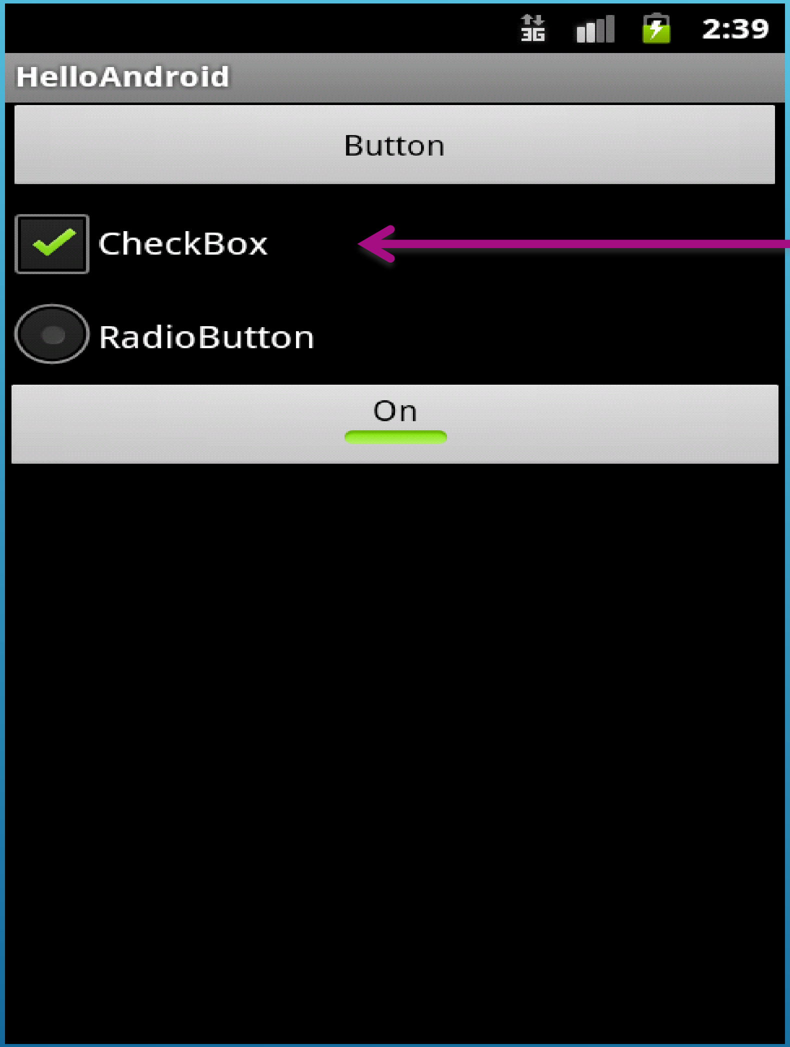
```
 <Button
       android:text="@string/textButton"
       android:id="@+id/idButton"
       android:background="@color/blue”
/>
```

```
<selector>
<item android:color="#ff819191"
     android:state_pressed="true”>
</item>
</selector>
```

res/color/blue.xml

- **CompoundButton**: Button + *state* (checked/unchecked)

Widgets: Button and CompoundButton

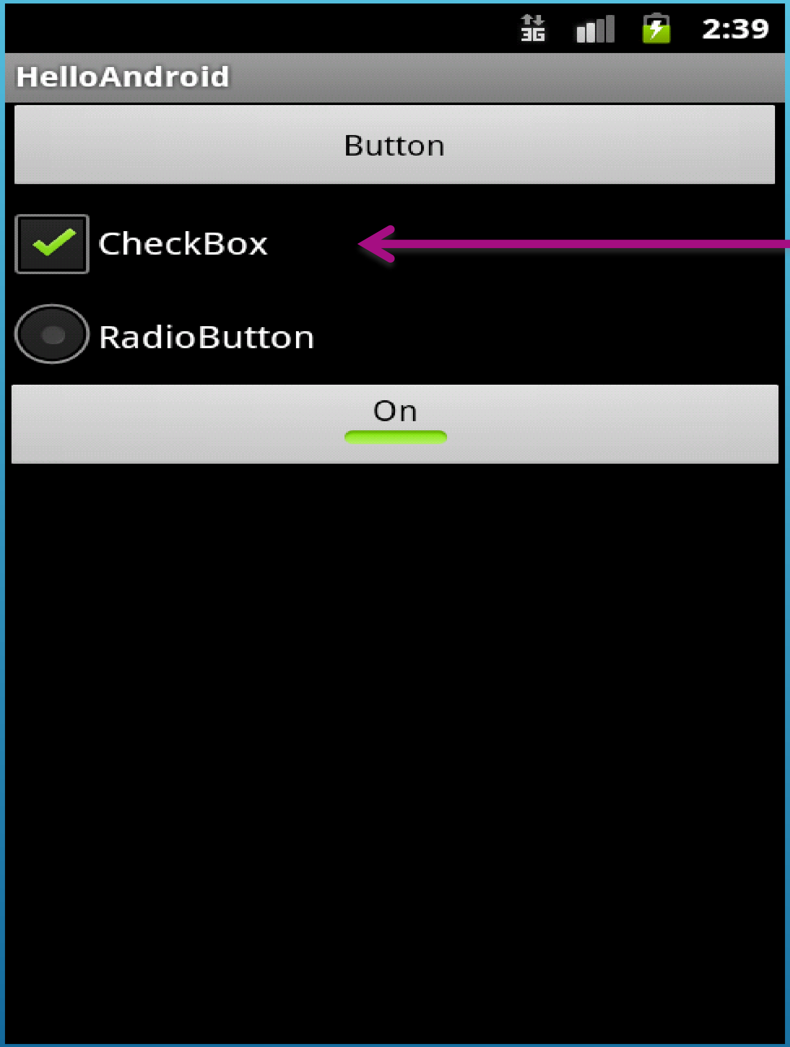


checkBox CompoundButton

XML tags: **<CheckBox>**
</CheckBox>

```
<CheckBox
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/buttonCheck"
      android:text="CheckBox"
      android:checked="true"
/>
```

Widgets: Button and CompoundButton



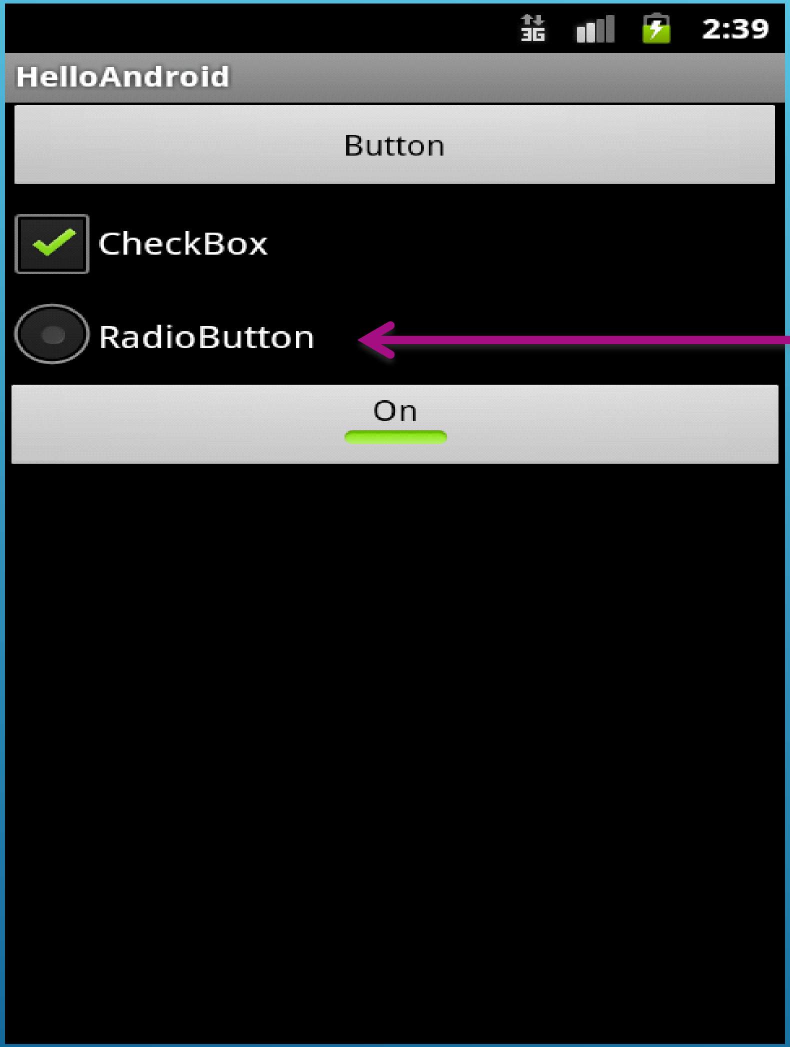
checkBox CompoundButton

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

✧ public boolean **setChecked**(bool)

Listener:
onCheckedChangeListener

Widgets: Button and CompoundButton

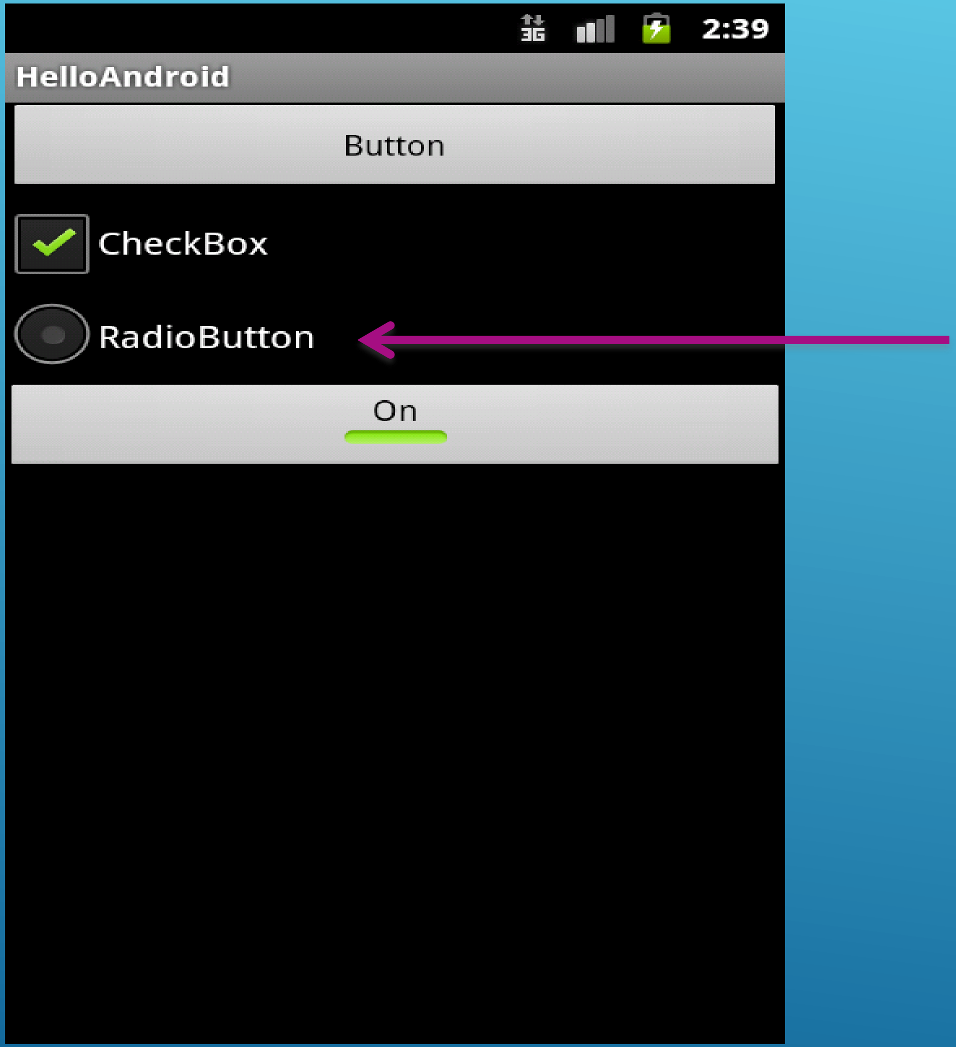


radioButton CompoundButton

XML tags: **<RadioButton>**
</RadioButton>

```
<RadioButton
        android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:id="@+id/buttonRadio"
          android:text="ButtonRadio"
          android:checked="true"
          />
```

Widgets: Button and CompoundButton

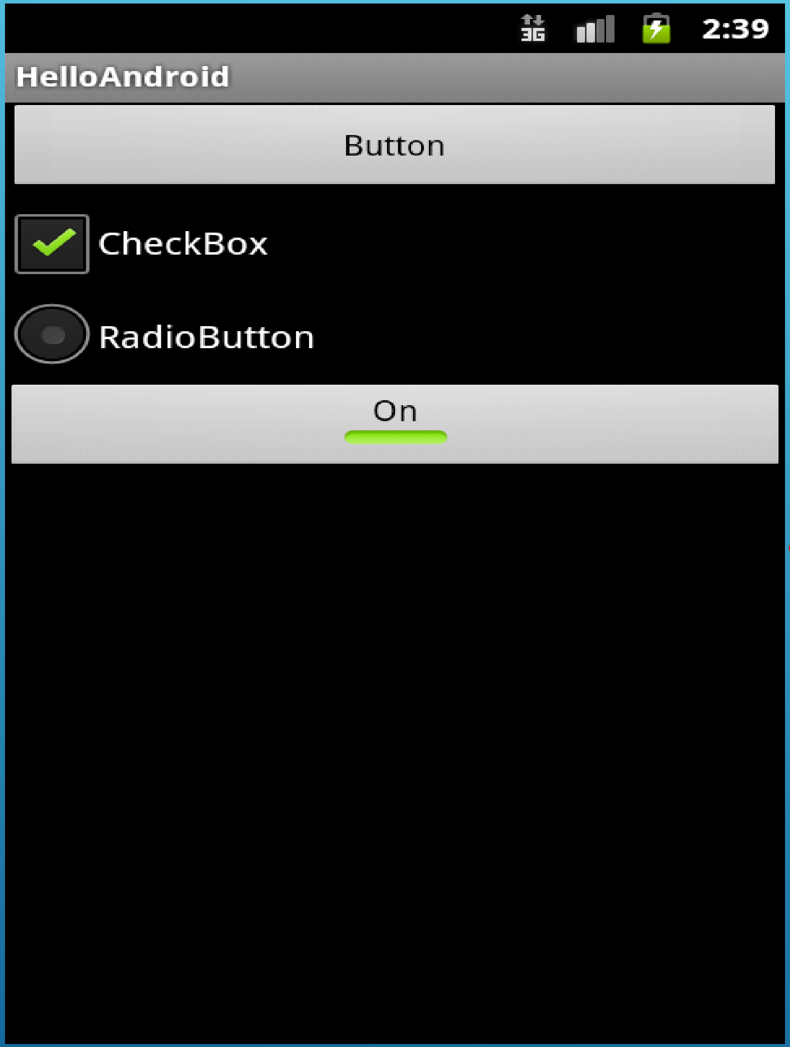


radioButton CompoundButton

- ✧ Define multiple (mutual-exclusive) options through a <**RadioGroup**> tag.
- ✧ Only one button can be checked within the same **RadioGroup**.

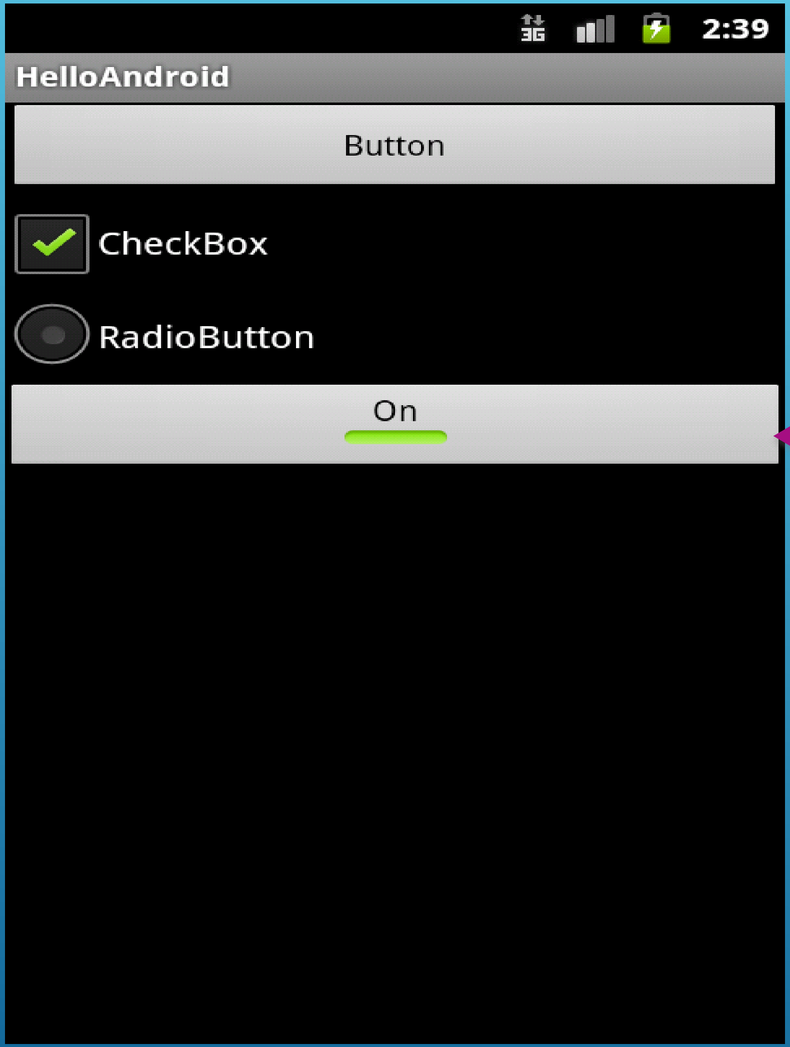
Listener:
OnCheckedChangeListener

Widgets: Button and CompoundButton



```
<RadioGroup
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical”>
        <RadioButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/buttonRadio1"
            android:text="Option 1"
            android:checked="true” />
      <RadioButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/buttonRadio2"
            android:text="Option 2” />
   </RadioGroup>
```

Widgets: Button and CompoundButton

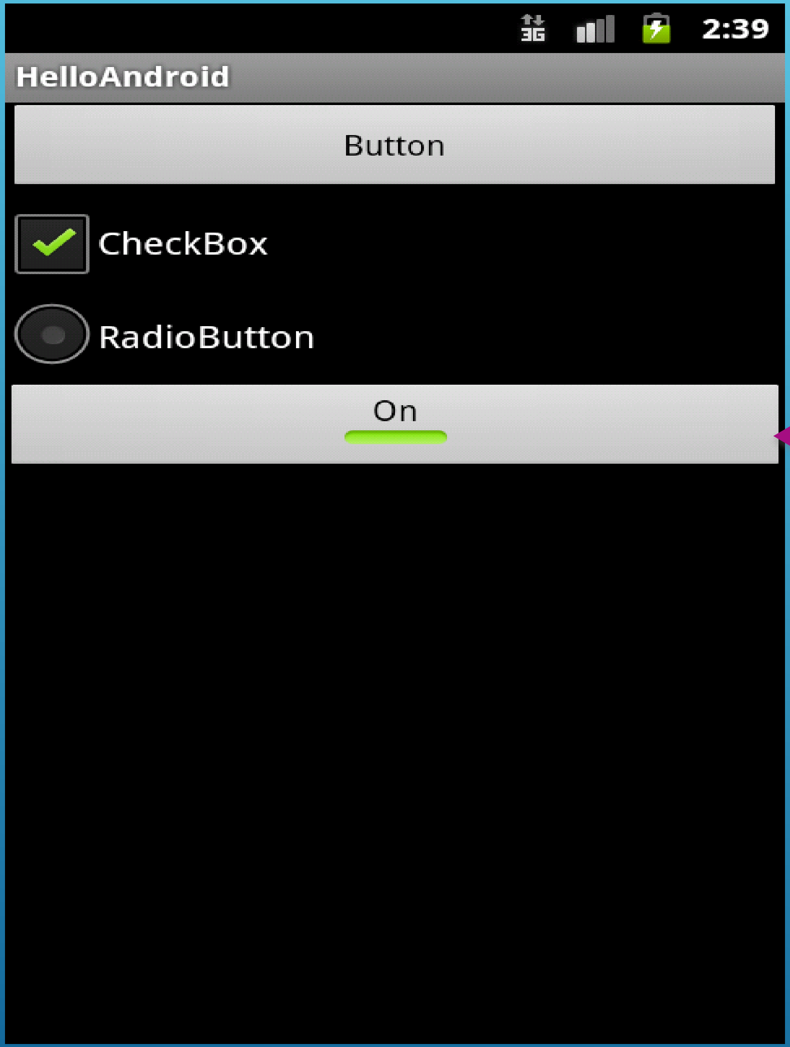


toggleButton
CompoundButton

XML tags: **<ToggleButton>**
</ToggleButton>

```
<ToggleButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/toggleButtonId"
        android:textOn="Button ON"
        android:textOff="Button OFF"
        android:checked="false”
/>
```

Widgets: Button and CompoundButton

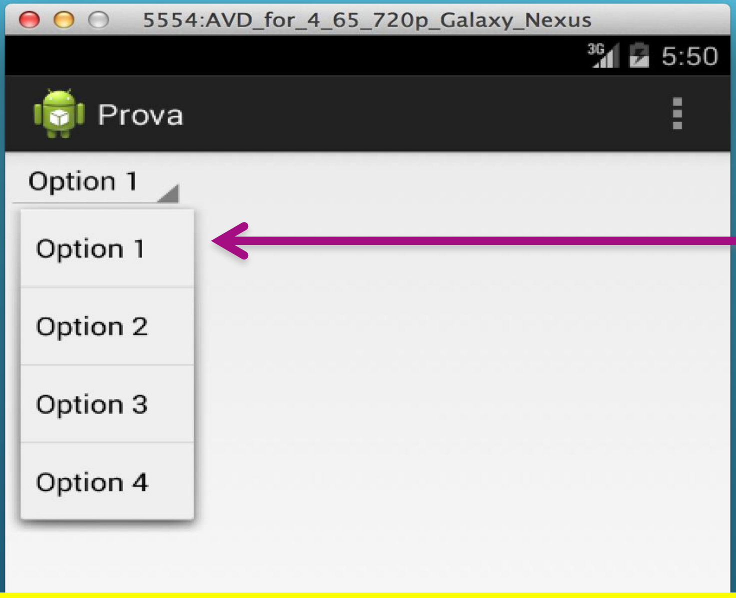


toggleButton
CompoundButton

- It can assume only 2 states: *checked/unchecked*
- Different labels for the states with: **android:textOn** and **android:textOff** XML attributes.

Listener:
OnCheckedChangeListener

Widgets: Spinners



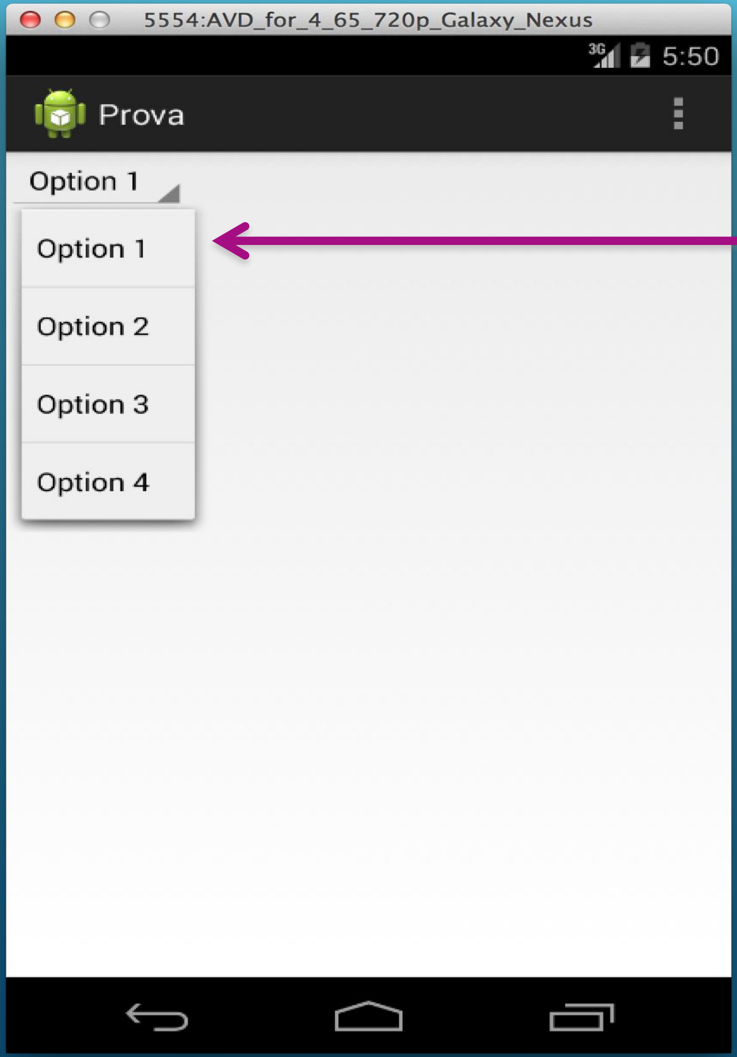
Spinner component

XML tags: **<Spinner>**
</Spinner>

```
<resources>
  <string-array name="stringOptions">
       <item>Option 1</item>
       <item>Option 2</item>
       <item>Option 3</item>
       <item>Option 4</item>
</string-array>
</resources>
```

```
<Spinner
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:id="@+id/spinnerId"
   android:entries="@array/stringOptions">
 </Spinner>
```

Widgets: Spinners



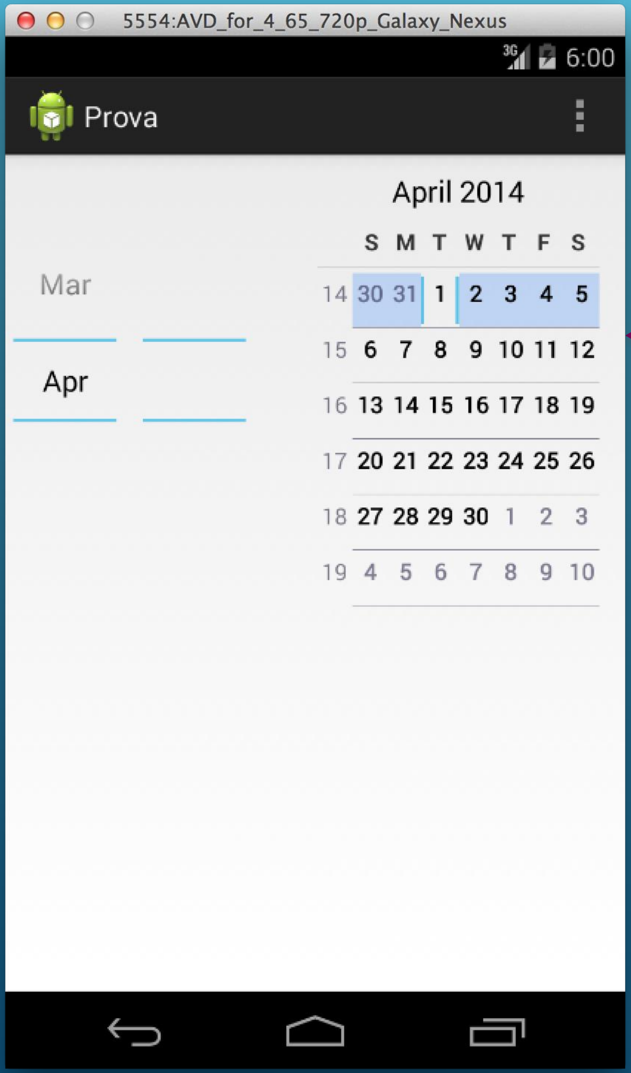
Spinner component

XML tags: **<Spinner>**
</Spinner>

- ✧ Provides a quick way to select values from a specific set.
- ✧ The spinner value-set can be defined in XML (through the **entries** tag) or through the *SpinnerAdapter* in Java

Listener:
OnItemSelectedListener

Widgets: Button and CompoundButton

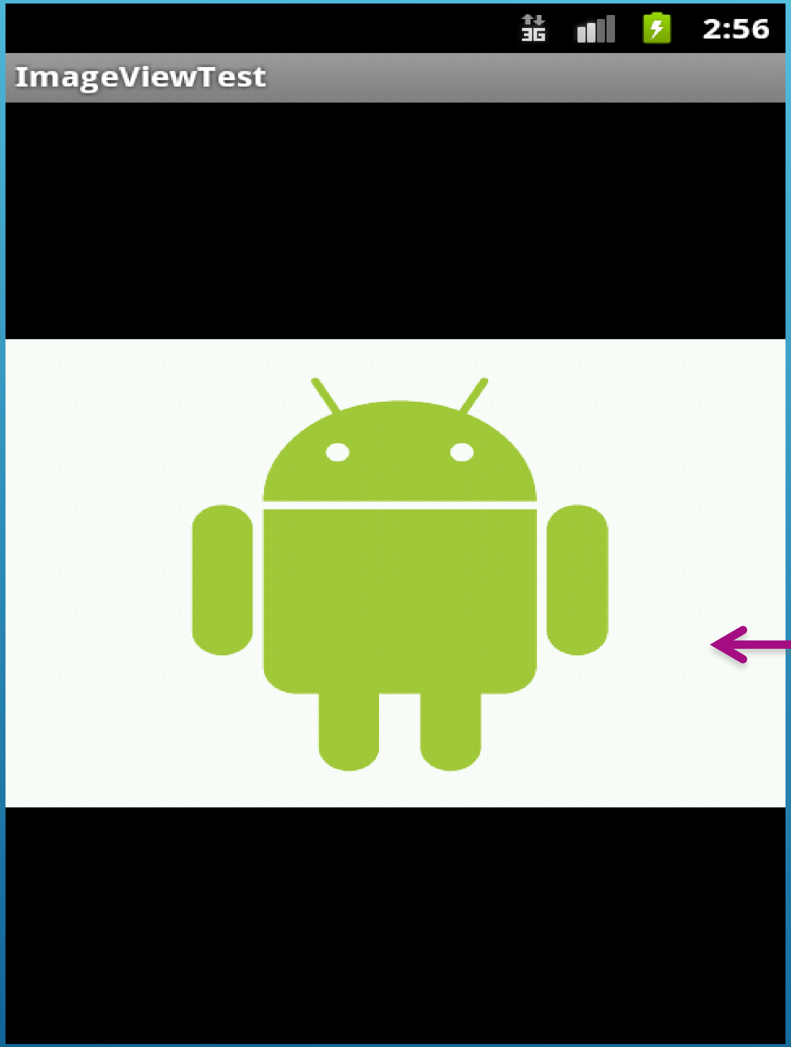


DataPicker component

XML tags: **<DataPicker>**
</DataPicker>

```
<DatePicker
    android:layout_width="wrap_content"
    android:layout_height="wrap_content”
    android:id=“@+id/datePickerId”
    android:endYear="1990"
    android:startYear="2014"
    android:maxDate="10/10/2014”
/>
```

Widgets: ImageView



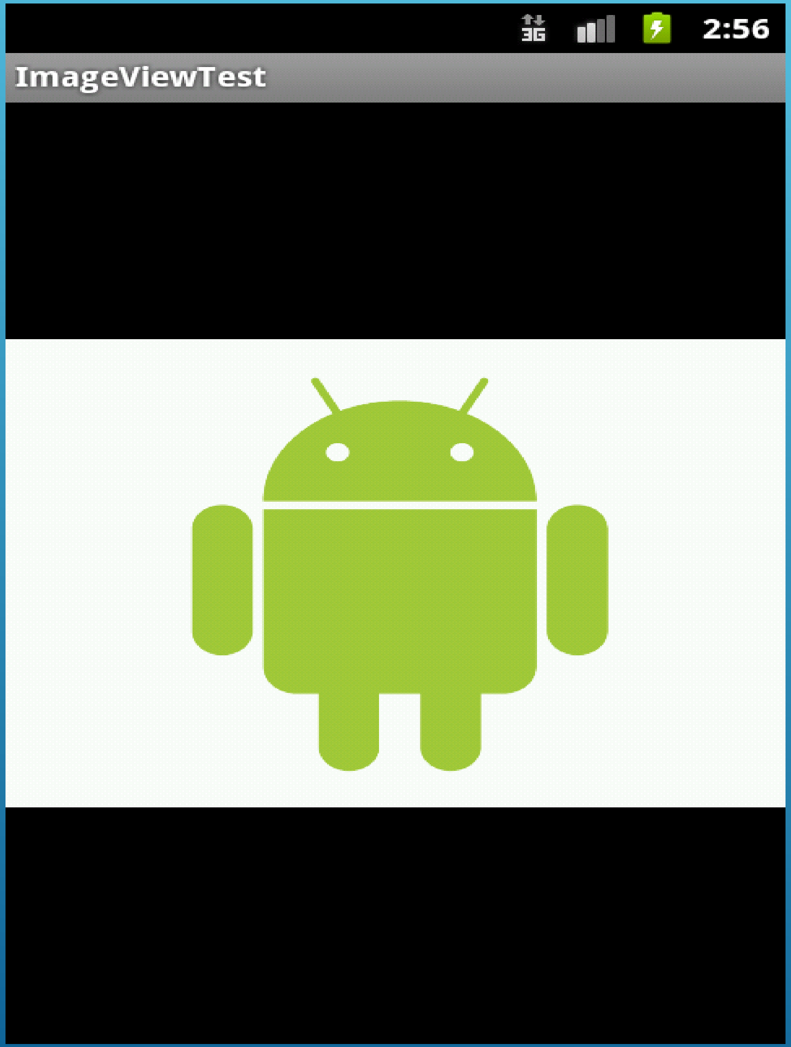
ImageView component

XML tags: **<ImageView>**
</ImageView>

```
<ImageView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:id="@+id/imageId"
   android:src="@drawable/android">
```

e/

Widgets: ImageView



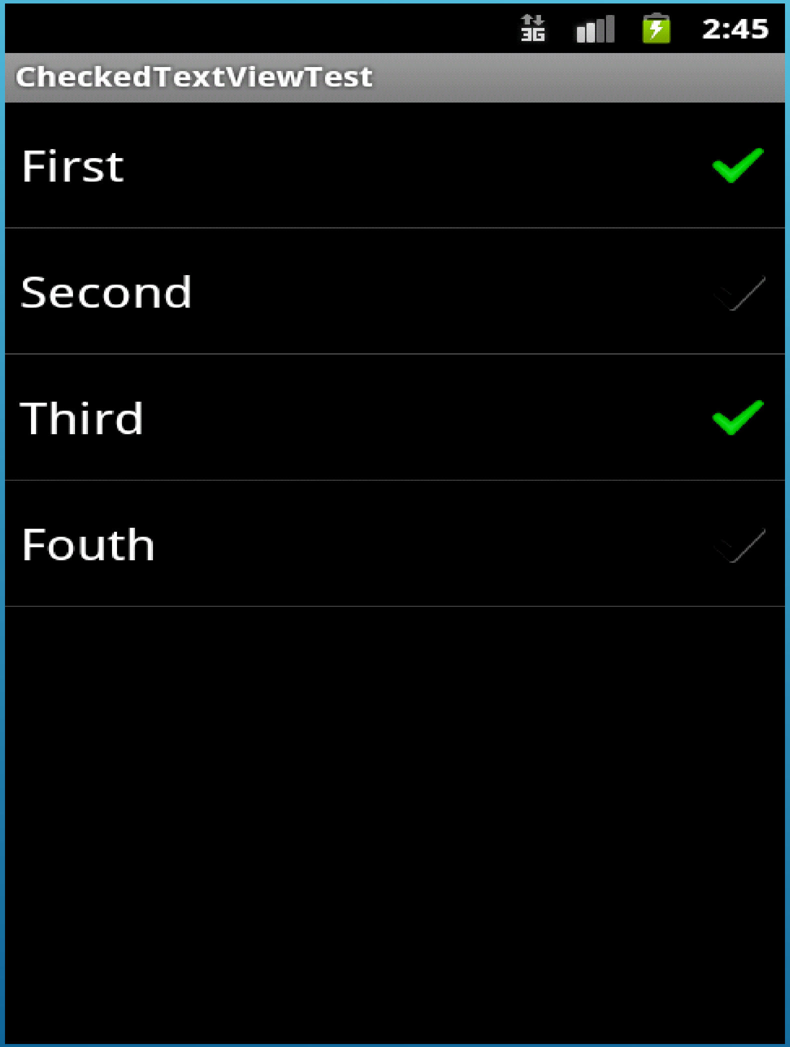
ImageView: subclass of View object.

Some methods to manipulate an image:

- void **setScaleType**(enum scaleType)
- void **setAlpha**(double alpha)
- void **setColorFilter**(ColorFilter color)

CENTER, CENTER_CROP, CENTER_INSIDE, FIT_CENTER, FIT_END, FIT_START, FIT_XY, MATRIX

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()

Views and Events

Views/Widgets are interactive components …

- … Upon certain action, an appropriate **event** will be fired
- Events generated by the user's interaction: click, long click, focus, items selected, items checked,drag, etc

PROBLEM: How to **handle** these events?

1. Directly from **XML**
2. Through **Event Listeners** (general, recommended)
3. Through **Event Handlers** (general)

Views **and** **Events**

- For a limited set of components, it is possible to manage the events through **callbacks**, directly indicated in the XML.

```
<Button
      android:text="@string/textButton"
      android:id="@+id/idButton"
      android:onClick=”doSomething”
/>
```

XML Layout File

Java class

```
public void doSomething(View w) {
     // Code to manage the click event
}
```

Views and Events

Views/Widgets are interactive components …

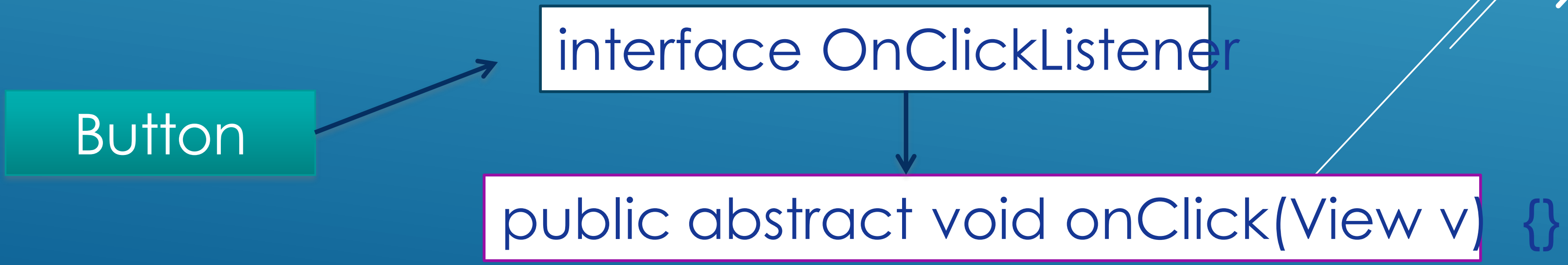
- … Upon certain action, an appropriate **event** will be fired
- Events generated by the user's interaction: click, long click, focus, items selected, items checked,drag, etc

PROBLEM: How to **handle** these events?

1. Directly from **XML**
2. Through **Event Listeners** (general, recommended)
3. Through **Event Handlers** (general)

Views and Events

- ➢ Each View contains a collection of nested **interfaces** (**listeners**).
    - Each listener handles a single **type of events**…
    - Each listener contains a single **callback** method …
    - The callback is invoked in occurrence of the event.


Button
interface OnClickListener
public abstract void onClick(View v) {}

Views and Events: ActionListener

To handle OnClick events through the ActionListener:

1. Implement the **nested interface** in the current Activity
2. Implement the **callback** method (onClick)
3. Associate the ActionListener to the Button through the View.**setOnClickEventListener**() method

```
Button btn = (Button)findViewById(R.id.btn);

btn.setOnClickListener(new OnClickListener() {

    (
    p



    }
});
```

```
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

To handle OnClick events through the ActionListener:

1. Create an **anonymous** OnClickListener object
2. Implement the **callback** method (onClick) for the anonymous object
3. Associate the ActionListener to the Button through the View.**setOnClickEventListener**() method

```
Button btn = (Button)findViewById(R.id.btn);

btn.setOnClickListener(new OnClickListener() {

    (
    p


    }
});
```

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

Views and Events: ActionListener

Some ActionListeners:

- **interface OnClickListener**

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

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

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

  abstract method: *onKey()*

Views and Events: ActionListener

Some ActionListeners:

- **interface OnCheckedChangeListener**

  abstract method: *onCheckedChanged()*
- **interface OnItemSelectedListener**

  abstract method: *onItemSelected()*
- **interface OnTouchListener**

  abstract method: *onTouch()*
- **interface OnCreateContextMenuListener**

  abstract method: *onCreateContextMenu()*

Views and Events: ActionListener

- Possible to fire an event directly from the Java code (without user's interaction) … useful for debugging purpose.
- Tipically in the form **performXXX()**
- The corresponding listener (if set) will be invoked…

```
…
Button button=(Button)findViewById(R.id.buttonNext);
button.performClick();
…
```

```
// Callback method
public void onClick(View v) {
                ….
}
```

Views and Events

- Views/Widgets are interactive components …
  - … Upon certain action, an appropriate **event** will be fired
  - Events generated by the user's interaction: click, long click, focus, items selected, items checked,drag, etc

- **PROBLEM**: How to **handle** these events?

1. Directly from **XML**
2. Through **Event Listeners** (general, recommended)
3. Through **Event Handlers** (general)

Views and Events

Event 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: *Events Handlers are used* for custom (user-defined) components …
- … *Events Listeners are used* for common View/Widget components …