Programming with Android: Views, Layouts and Events

Federico Montori

Dipartimento di Informatica: Scienza e Ingegneria

Università di Bologna

Android: Where are we now …

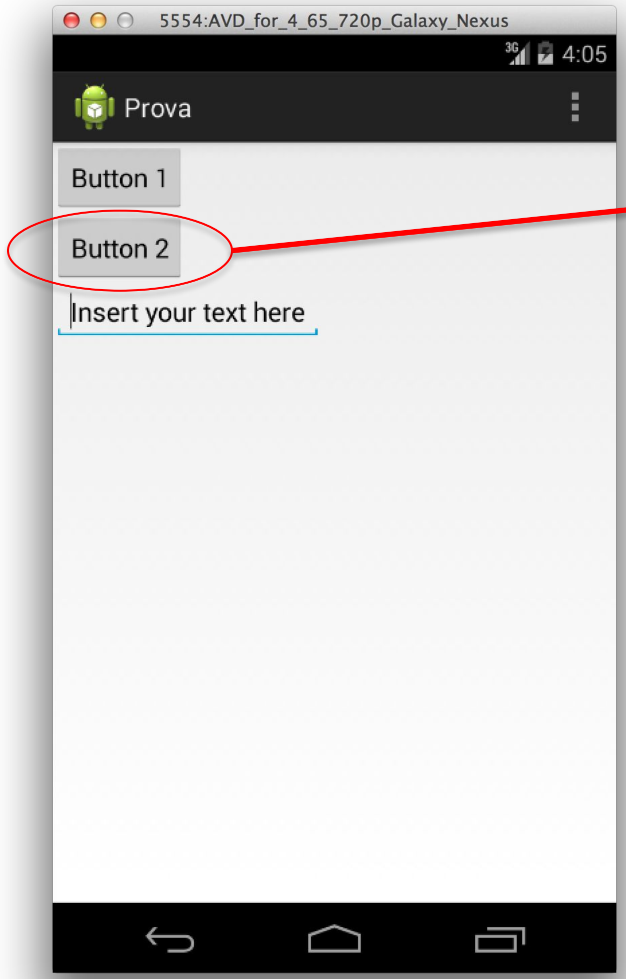
Android Applications' anatomy:

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

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

Views: Java and XML code

- Views 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”
        ….
/>
```

Views: Java and XML code

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

XML

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

JAVA

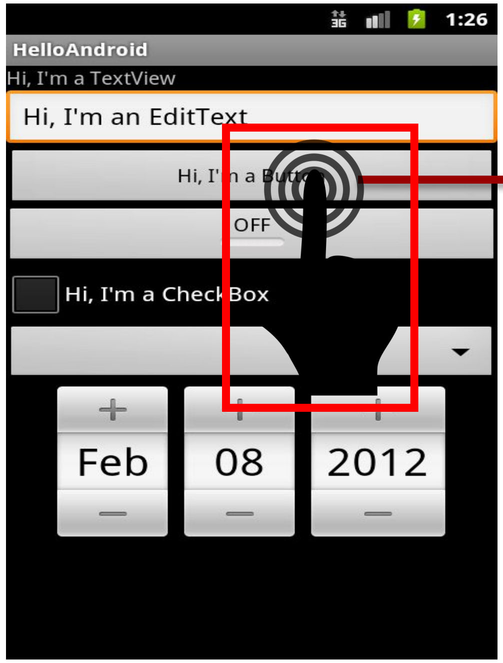
```
public TextView text;
text = (TextView)findViewById(R.id.name1);
```

CAST REQUIRED
UNTIL API 26

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

Views: Java and XML code

- Each View can generate events, that can be captured by **Listeners** (or other methods) that define the appropriate actions to be performed in response to each event.



ONCLICK event

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

Views: Java and XML code

- Each View 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 = findViewById(R.id.name1);
text.setVisibility(true)
text.requestFocus();
```

Views and Events

Views 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 Handlers** (general)
3. Through **Event Listeners** (general, <u>recommended</u>)

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 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 Handlers** (general)
3. Through **Event Listeners** (general, recommended)

Views and Events

Event Handlers □ Some views have **callback** methods to handle specific events

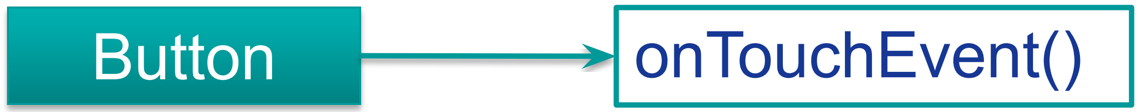
When a **Button** is touched □ **onTouchEvent**() called

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 …

Views and Events

- Each View contains several methods that are called when an event occurs:
    - e.g. onTouchEvent() when the View is clicked
- In order to intercept it we should extend the View class and override the call.
- This is impractical… much better to have a separate class that handles all the hassle.



Views and Events

Views 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

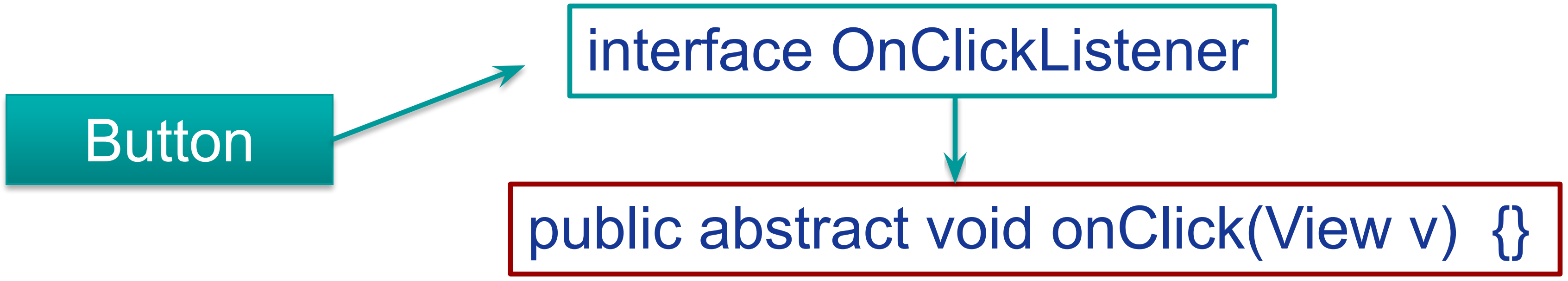
PROBLEM: How to **handle** these events?

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

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.**setOnClickListener**() method

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

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.**setOnClickListener**() method

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

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.**setOnClickListener**() method

From Java 8 we can use the **LAMBDA** notation!
https://www.w3schools.com/java/java_lambda.asp#:~:text=Lambda%20Expressions%20were%20added%20in,the%20body%20of%20a%20method.

```
Button btn = findViewById(R.id.buttonNext);
btn.setOnClickListener(v -> {
        // 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

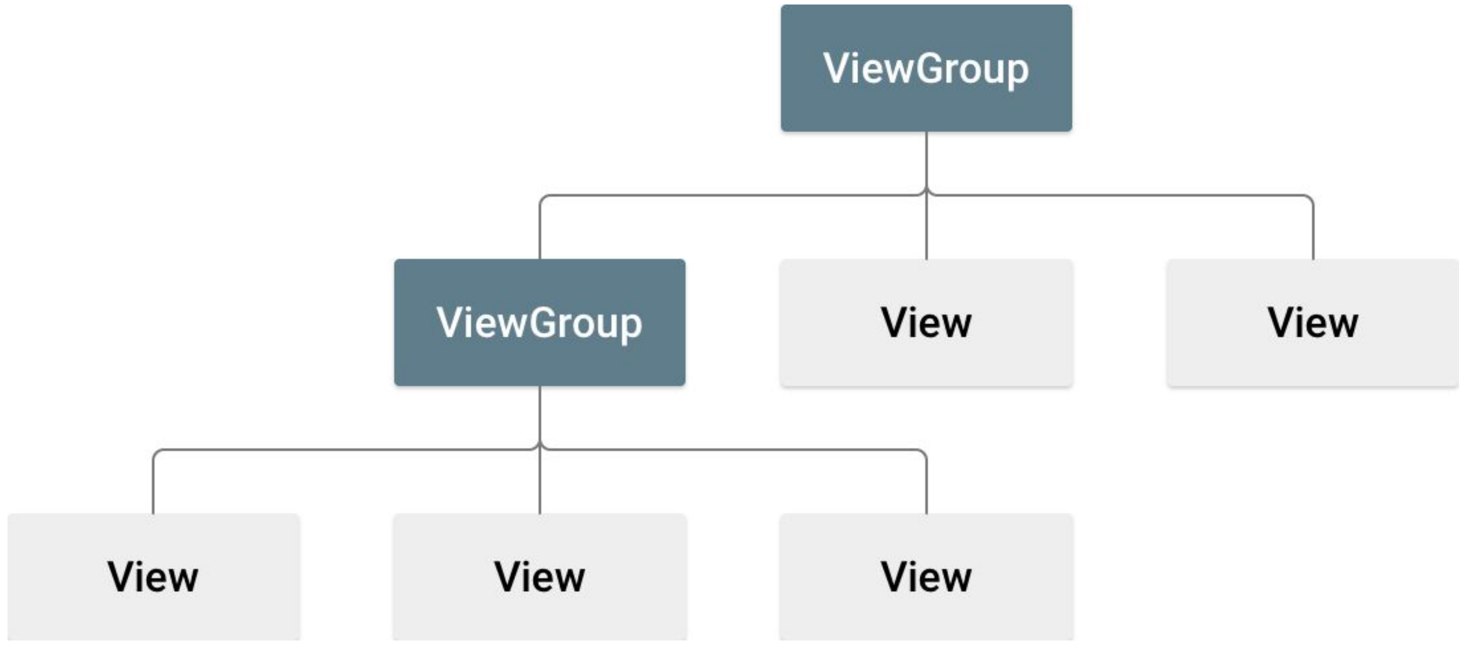
- Possible to fire an event directly from the Java code (without user's interaction) … useful for debugging purpose.
- Tipically in the form **performXXX()**
- The corresponding listener (if set) will be invoked…

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

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

Layouts: outline

- View objects represent something a user can see and interact with (rectangular areas).



- ViewGroup objects are invisible containers that define a layout structure for the Views declared in it.
- **<u>NB. ViewGroup is a (subclass of) View</u>**

Layouts: outline

- Main difference between a Drawable and a View is reaction to events.
- Is declared in an XML file **OR** inside an Activity
- Every view has a unique ID
- Use findViewById(int id) to get it
- Views can be customized (buttons, texts, images…)
- Views that are not ViewGroups are often referred to as Widgets (not to be confused with App Widgets)

ViewGroup and layout

- ViewGroup is a view container
- It is responsible for placing other views on the display
- Every layout must extend a ViewGroup (i.e. a Layout **IS** a ViewGroup)
- Every view needs to specify:
  - android:layout_height
  - android:layout_width
  - A dimension or one of match_parent or wrap_content

Typical steps in Layout building

When building your app you first declare your layout(s) in XML in the folder “res/layouts”:

```xml
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">

   <TextView
       android:id="@+id/number_text"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Type in the number:"
       android:textAppearance="@style/TextAppearance.AppCompat.Large"
       app:layout_constraintBottom_toBottomOf="parent"
       app:layout_constraintHorizontal_bias="0.0"
       app:layout_constraintLeft_toLeftOf="parent"
       app:layout_constraintRight_toRightOf="parent"
       app:layout_constraintTop_toTopOf="parent"
       app:layout_constraintVertical_bias="0.0" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Note that you can still declare the layout programmatically...

Typical steps in Layout building

Your layout is then compiled into a View resource that has to be loaded by the Activity making use of it.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

You will notice that each of your Views and ViewGroups has a number of attributes:

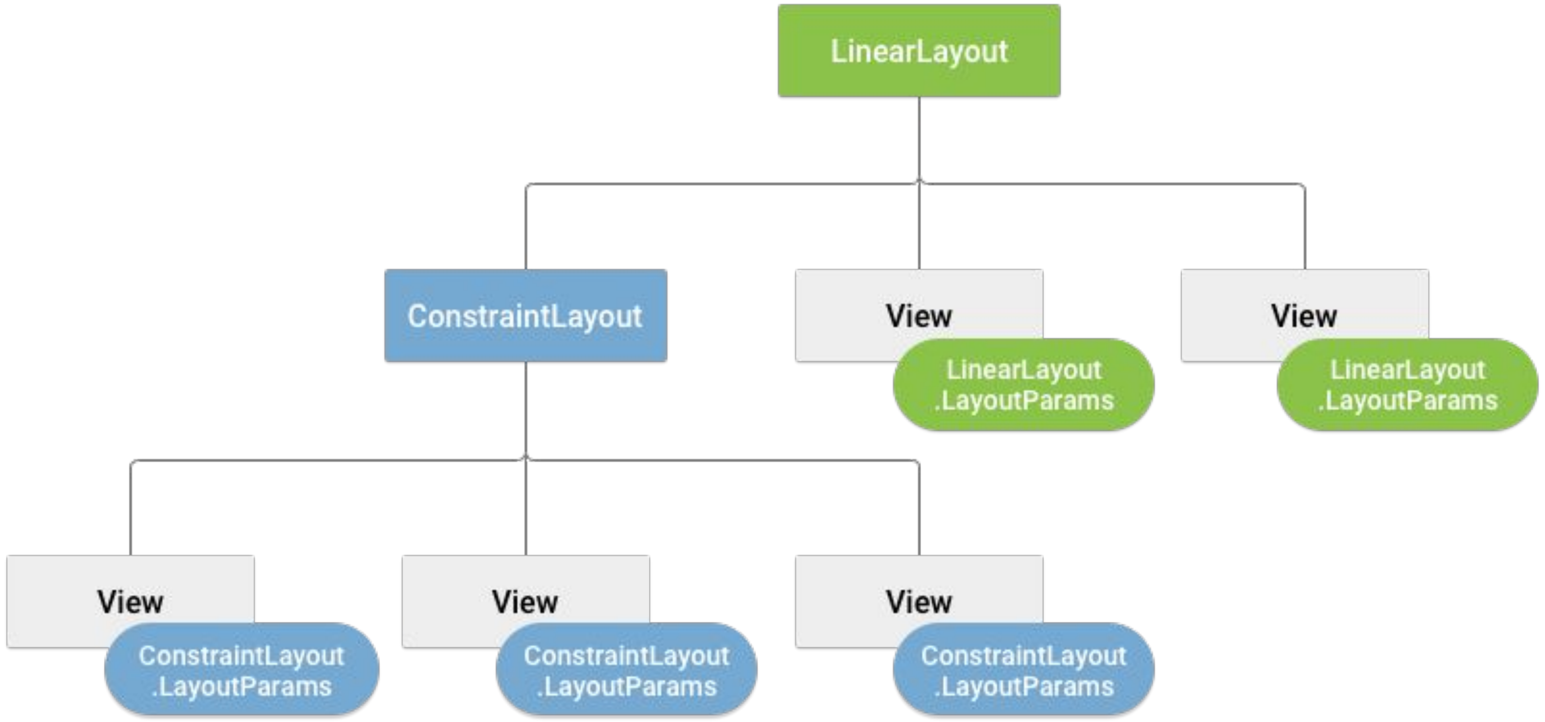
```
android:id="@+id/number_text"
```

@ means: “parse and expand the rest of the string as an id resource.

+ means: “this is going to be added as a **new** id in R.java

Layout **parameters**

XML layout attributes named *layout_something* define layout parameters for the View that are appropriate for the ViewGroup in which it resides.


LinearLayout
ConstraintLayout
View
LinearLayout .LayoutParams
View
LinearLayout .LayoutParams
View
ConstraintLayout .LayoutParams
View
ConstraintLayout .LayoutParams
View
ConstraintLayout .LayoutParams

Each View specifies layout params that each children must implement (if ViewGroup).

Each View must implement layout params required by the parent.

Layout parameters

E.g. each Layout needs the children Views to implement *layout_width* and *layout_height.*

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

Typically match_parent, wrap_content, 0dp...

Layouts and Views are rectangular objects. Can get the origin coordinates by *getLeft()* and *getTop()*

Some useful methods

- getLeft()
- getTop()
- getMeasuredWidth()
- getMeasuredHeight()
- getWidth()
- getHeight()
- requestLayout()
- invalidate()

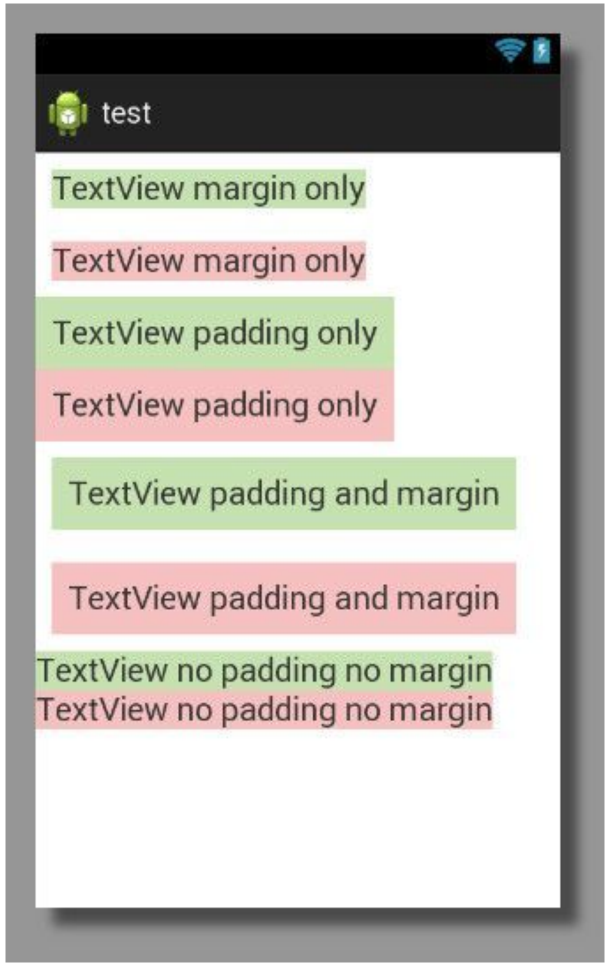
There is a difference between how big the View wants to be (e.g. *getMeasuredWidth()*) and how big it is drawn (e.g. *getWidth()*).

What is the difference between **android:width** and **android:layout_width**?

The first is a View attribute
The second implements an attribute from the parent layout.

More on Layout parameters

Android supports also padding and margins…


test

TextView margin only

TextView margin only

TextView padding only

TextView padding only

TextView padding and margin

TextView padding and margin

TextView no padding no margin

TextView no padding no margin

Padding is a View property

- android:padding

Margin is a layout property

- android:layout_margin

Layouts

- Some layouts are pre-defined by Android
- Some of these (most common and legacy) are:
  - LinearLayout
  - RelativeLayout
  - TableLayout
  - FrameLayout
  - ConstraintLayout
- A layout can be declared inside another layout

LinearLayout

- Dispose views on a single row or column, depending on android:layout_orientation
- The orientation could also be declared via setOrientation(int orientation)
  - orientation is one of HORIZONTAL or VERTICAL
- Has two other attributes:
  - layout_gravity
  - layout_weight

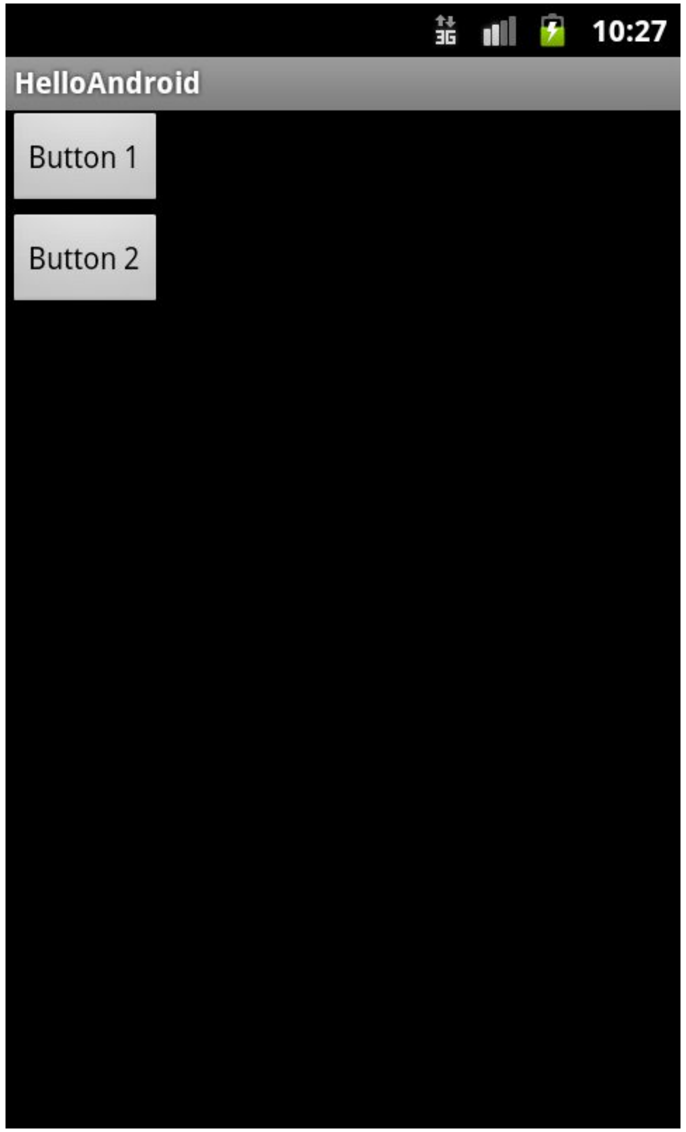
LinearLayout

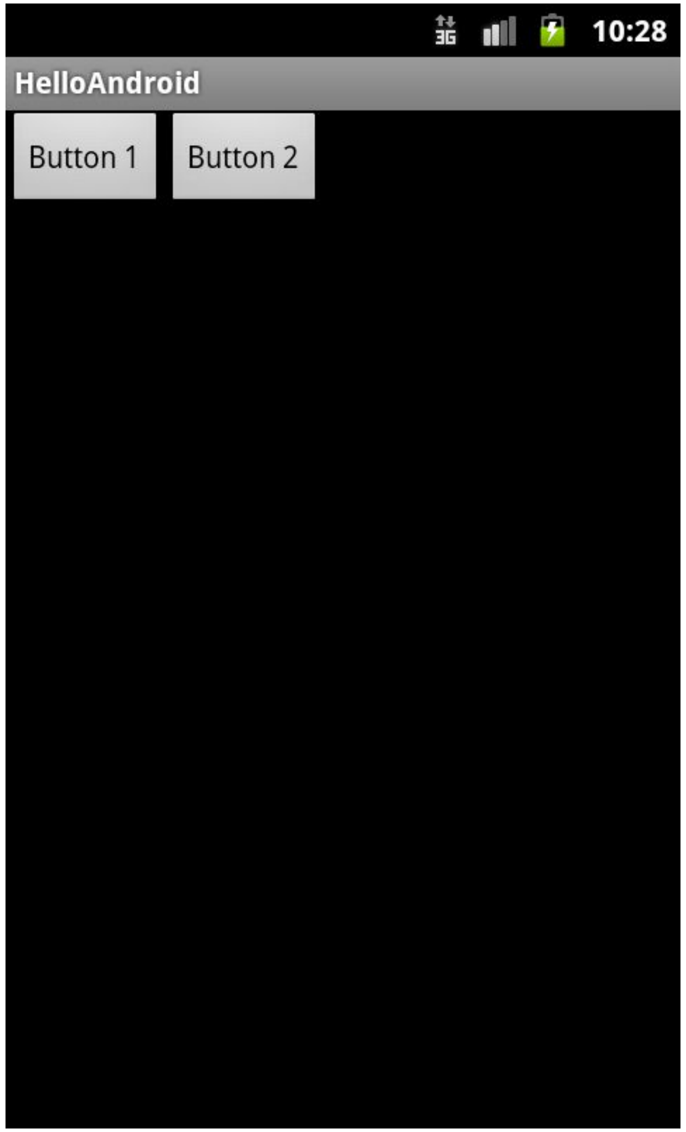
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >                  <!-- Can be horizontal -->

   <Button
      android:id="@+id/button1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/buttonString1" />

   <Button
      android:id="@+id/button2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/buttonString2" />
</LinearLayout>
```

Sometimes you may encounter *fill_parent* instead of *match_partent*.

The first one is deprecated since API 8 and they are exactly the same.

LinearLayout

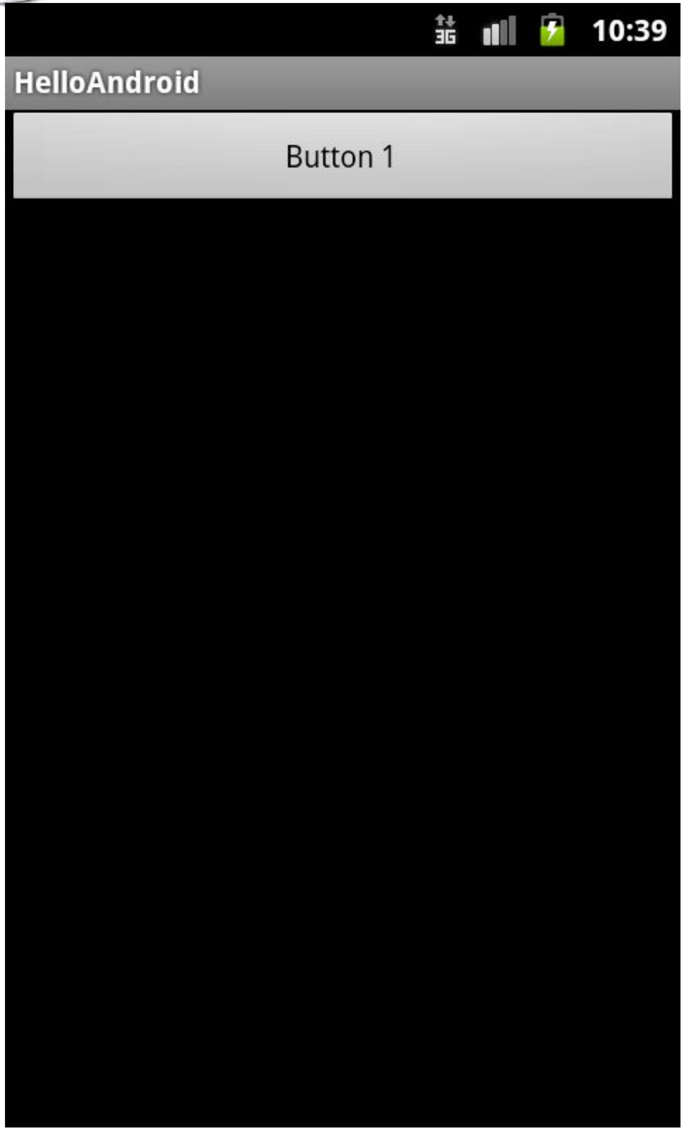


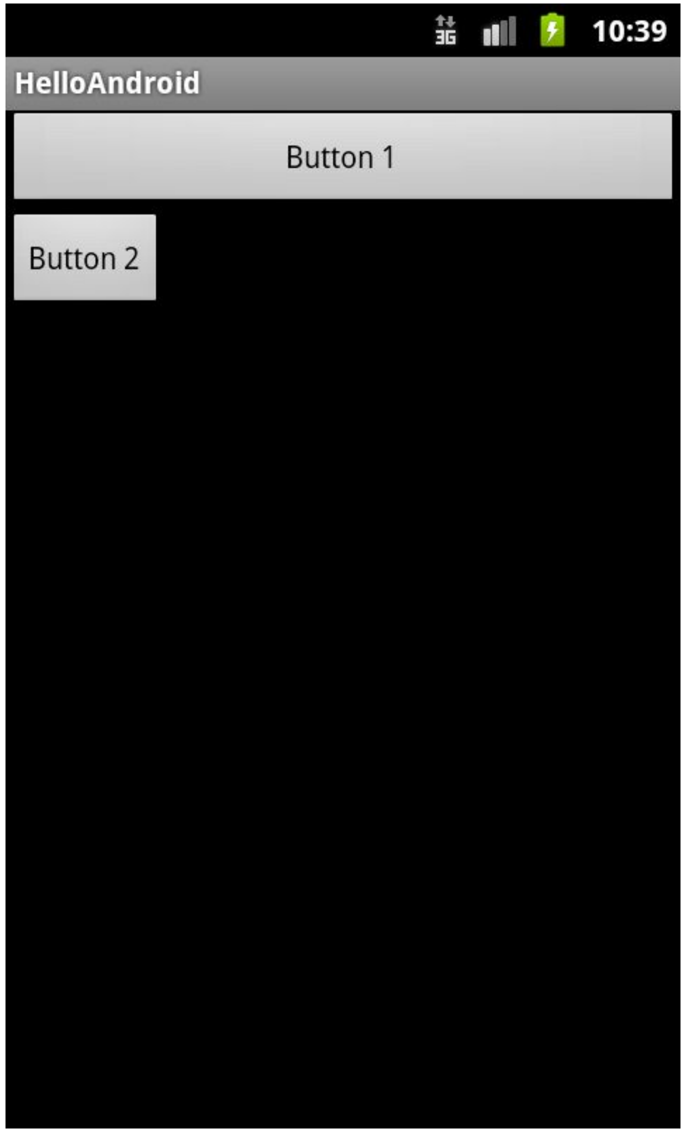


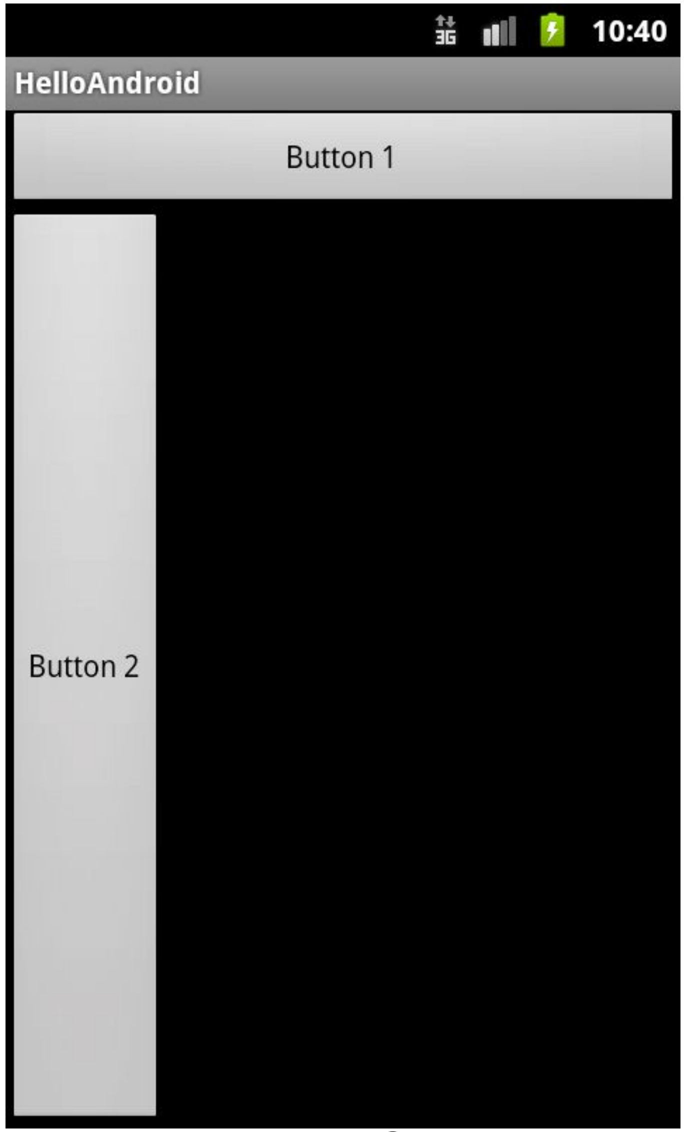
LinearLayout

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

    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/buttonString1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:text="@string/buttonString2" />
</LinearLayout>
```

LinearLayout







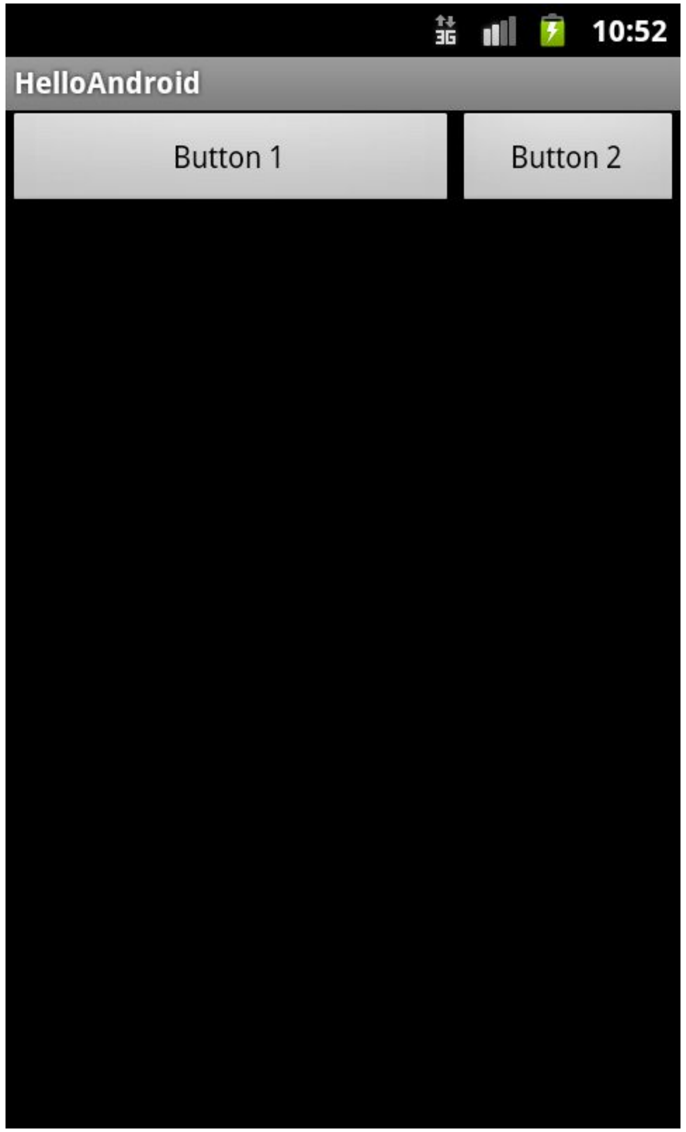
LinearLayout weight

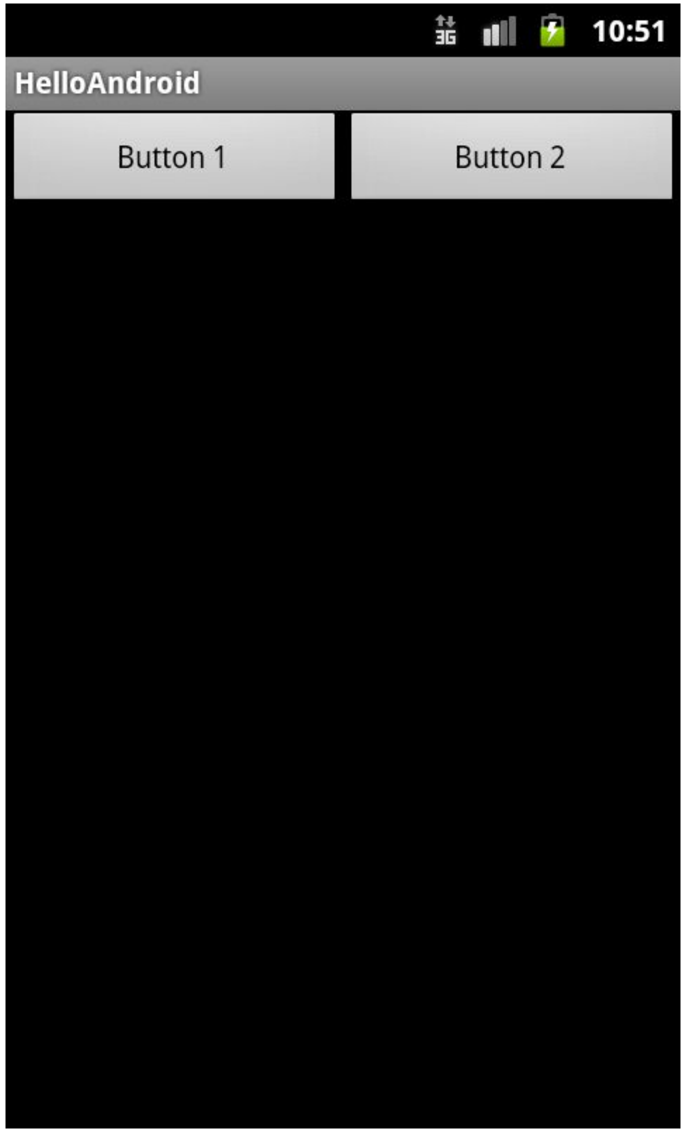
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"    android:layout_height="match_parent"    android:orientation="horizontal" >

   <Button
      android:id="@+id/button1"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:text="@string/buttonString1"
      android:layout_weight="2" />

   <Button
      android:id="@+id/button2"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:text="@string/buttonString2"
      android:layout_weight="1"       />
</LinearLayout>
```

If the Views are to share the space, they are assigned a weight and their layout_width is set to 0dp (if layout is horizontal).

0dp means pretty much: fill the available space

LinearLayout weight

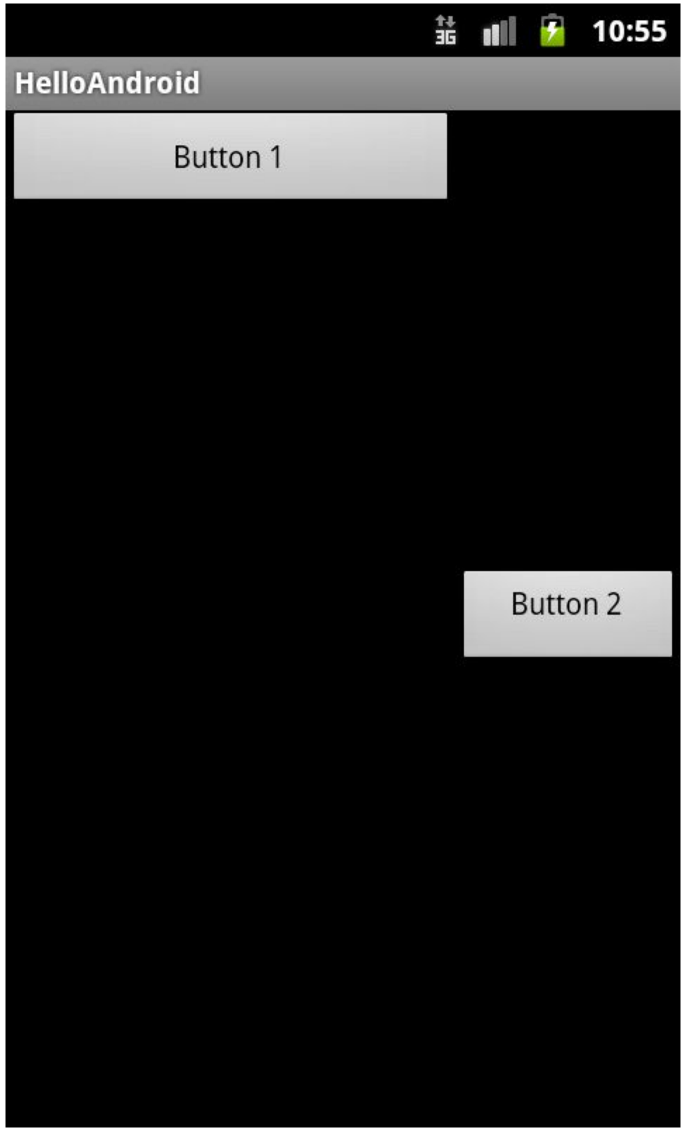




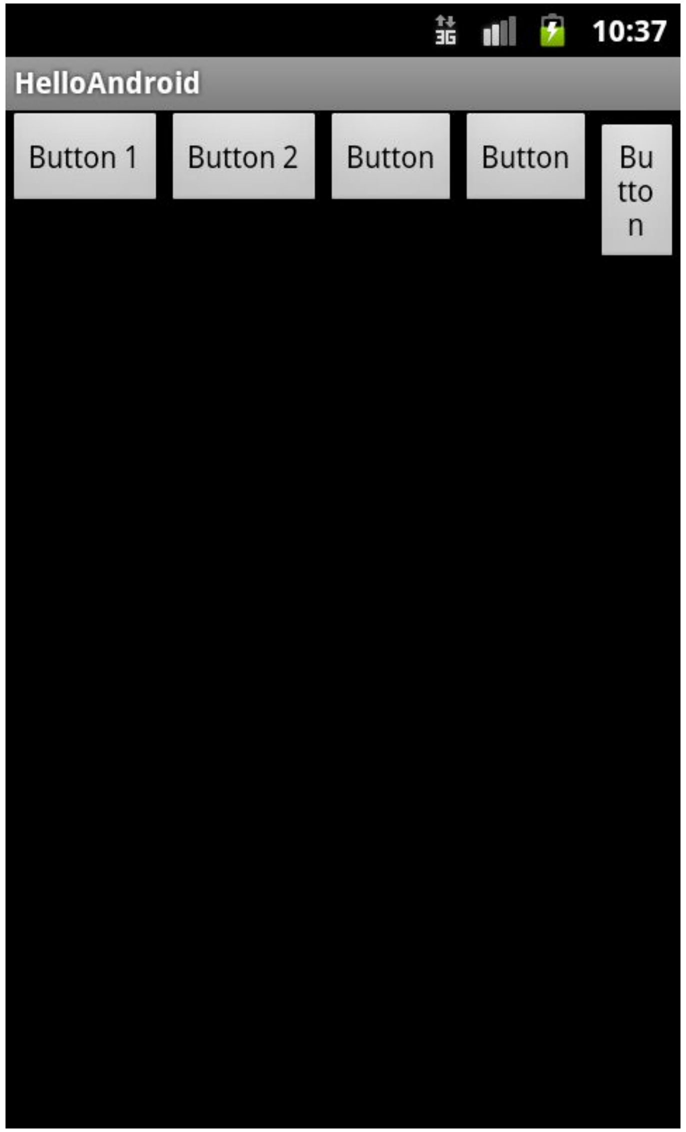
LinearLayout gravity

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


   <Button
      android:id="@+id/button1"
      android:layout_width="match_parent"      android:layout_height="wrap_content"
      android:text="@string/buttonString1"
      android:layout_weight="2" />
   <Button
      android:id="@+id/button2"
      android:layout_width="match_parent"      android:layout_height="wrap_content"
      android:text="@string/buttonString2"
      android:layout_weight="1"
      android:layout_gravity="center_vertical"
      android:gravity="top|center" />


</LinearLayout>
```

LinearLayout gravity



LinearLayout problem



This happens with weights…

Without weights views can even disappear...

RelativeLayout

- Disposes views according to the container or according to other views
- The gravity attribute indicates what views are more important to define the layout
- Useful to align views in “blocks”

RelativeLayout

android:layout_alignParentTop

If "true", makes the top edge of this view match the top edge of the parent.

android:layout_centerVertical

If "true", centers this child vertically within its parent.

android:layout_below

Positions the top edge of this view below the view specified with a resource ID.

android:layout_toRightOf

Positions the left edge of this view to the right of the view specified with a resource ID.

RelativeLayout

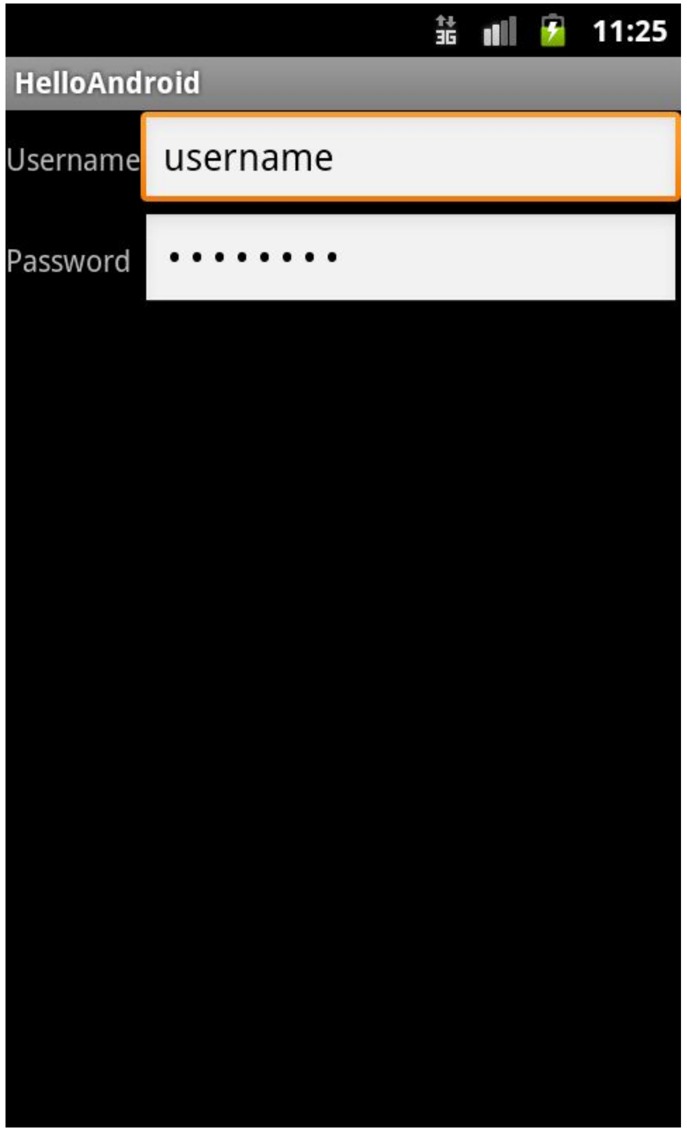
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"    android:layout_height="match_parent" >

   <EditText
      android:id="@+id/username"       android:text="username"
      android:inputType="text"
      android:layout_width="0dp"       android:layout_height="wrap_content"
      android:layout_alignParentRight="true"
      android:layout_toRightOf="@+id/usernameLabel" >
   </EditText>

   <TextView
      android:id="@+id/usernameLabel"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignBaseline="@+id/username"
      android:text="Username" />
```

alignBaseline aligns the text within the box, not the box iteself.

RelativeLayout

```
<EditText
    android:id="@+id/password"      android:text="password"
    android:inputType="textPassword"
    android:layout_below="@+id/username"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/username"
    android:layout_alignParentRight="true"
    android:layout_toRightOf="@+id/passwordLabel" >
  </EditText>

  <TextView
    android:id="@+id/passwordLabel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/password"
    android:text="Password" />
</RelativeLayout>
```

RelativeLayout



TableLayout

- As the name say, similar to a Table
- Has some attributes to customize the layout:
  - android:layout_column
  - android:layout_span
  - android:stretchColumns
  - android:shrinkColumns
  - android:collapseColumns
- Each row is inside a <TableRow> element

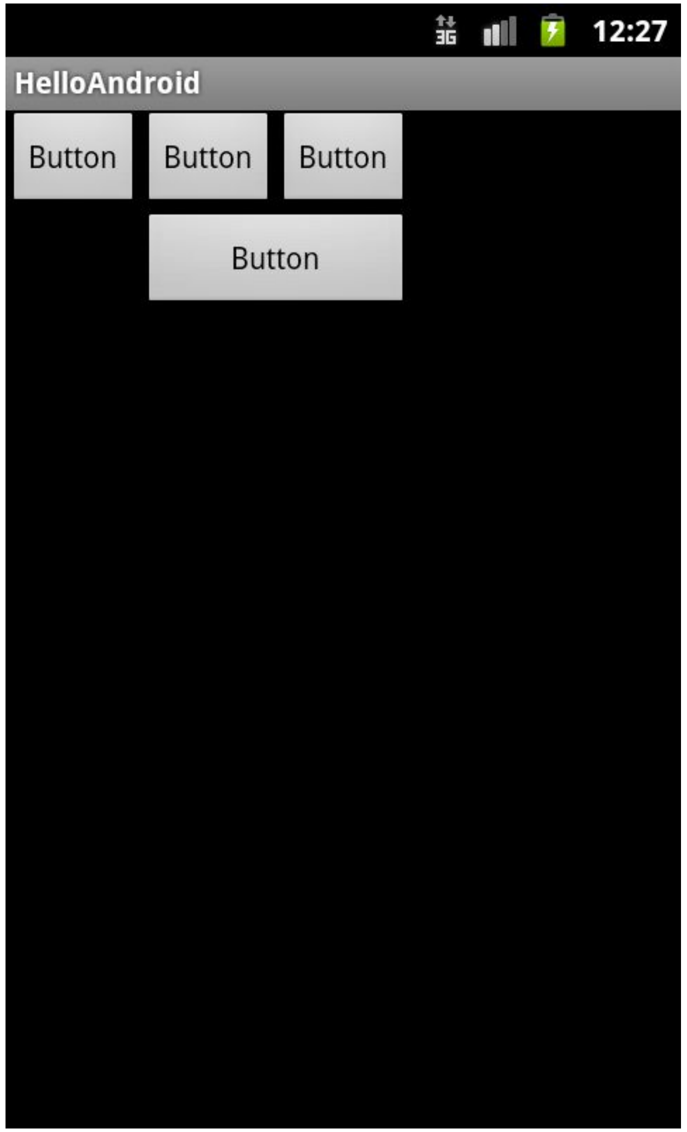
TableLayout

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout android:layout_width="fill_parent"
    android:layout_height="fill_parent" xmlns:android="http://schemas.android.com/apk/res/android" android:id="@+id/tableLayout">

        <TableRow android:layout_width="wrap_content" android:layout_height="wrap_content“ android:id="@+id/firstRow">
                <Button android:id="@+id/button1“
                        android:layout_width="wrap_content“
                        android:layout_height="wrap_content“
                        android:text="Button" />
                <Button android:id="@+id/button2“
                        android:layout_width="match_parent"
                        android:layout_height="match_parent"
                        android:text="Button" />
                <Button android:id="@+id/button3"
                        android:layout_width="match_parent"
                        android:layout_height=”match_parent"
                        android:text="Button" />
        </TableRow>
```

TableLayout

```
<TableRow
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/secondRow">

   <Button     android:layout_column="1"
               android:layout_span="2"
        android:id="@+id/button4"
               android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button">
   </Button>
</TableRow>

</TableLayout>
```

TableLayout



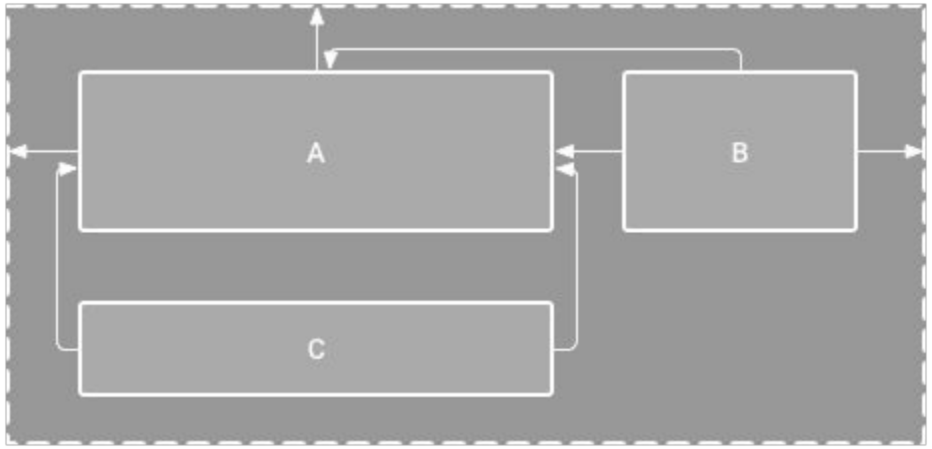
FrameLayout and AbsoluteLayout

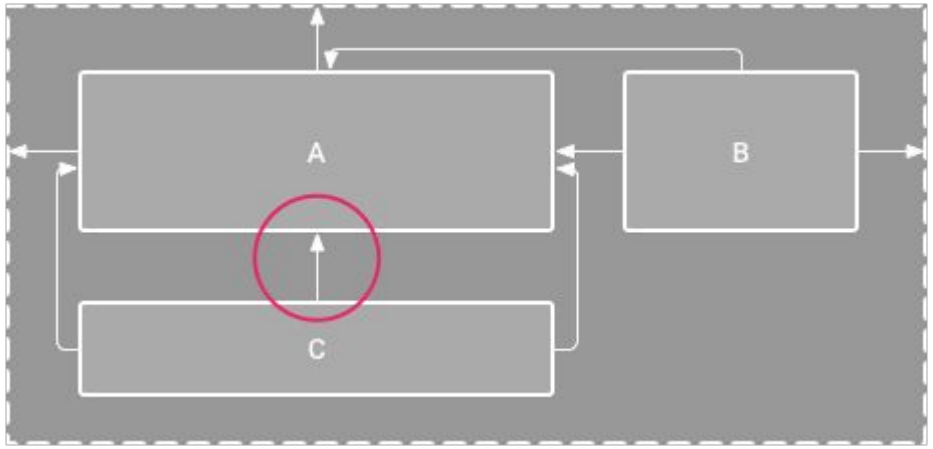
- FrameLayout
  - Adds an attribute, android:visibility
  - Blocks out portion of the screen to suit (typically) only one object.
  - Size equal to the size of its largest (non GONE) child.
- AbsoluteLayout
  - Deprecated
  - Specify position with x and y

ConstraintLayout

- Flat view hierarchy
- Similar to RelativeLayout
- > Android 2.3
- Overarching idea: define constraints (top/bottom/left/right) for each view
- Each constraint has to be defined to another (previously declared) view, another layout or an invisible guideline.
- You may have noticed that it is the default one...

ConstraintLayout: example





- Both layouts are fine
  - The left one has no top constraint on C, which will then be placed at the top

ConstraintLayout: how to use

For previous versions only, in Androidx it is built-in.

- Add directions to build.gradle

```
repositories {
    maven {
        url 'https://maven.google.com'
    }
}
```

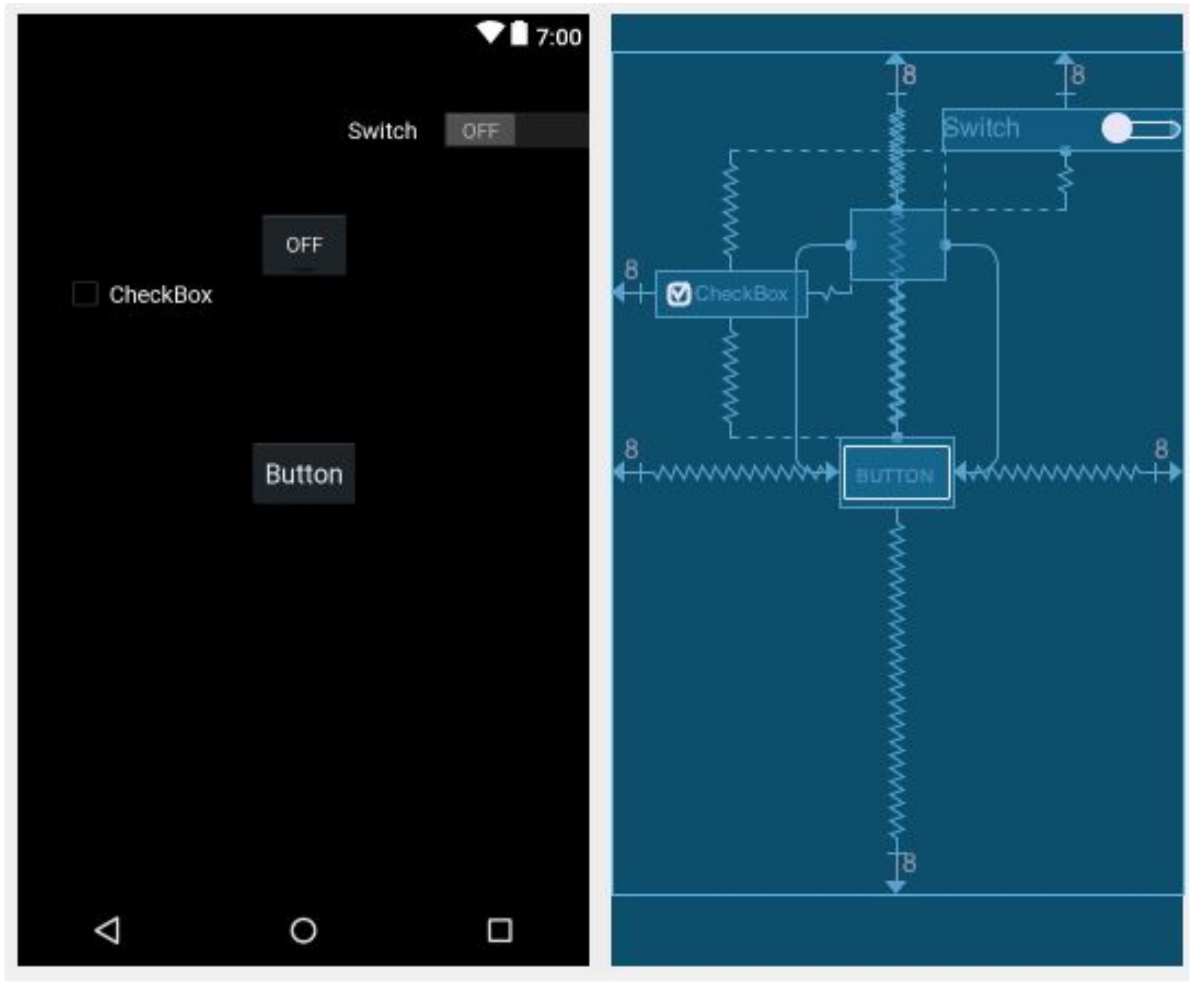
```
dependencies {
    compile 'com.android.support.constraint:constraint-layout:1.0.2'
}
```

- Sync the project

ConstraintLayout: how to create one

- Converting a Layout
  - Just right click on the layout and select the conversion option
- Creating a ConstraintLayout in older versions
  - Create a new layout
  - As root-tag, put ***android.support.constraint.ConstraintLayout***

ConstraintLayout: how to create one

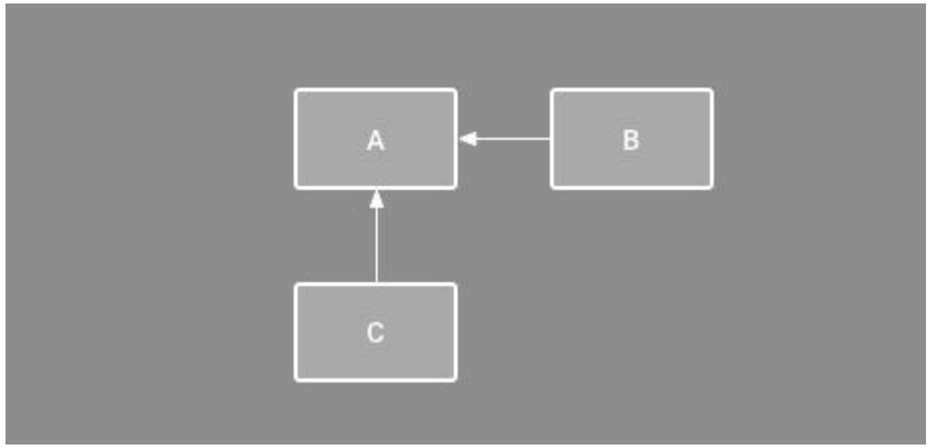


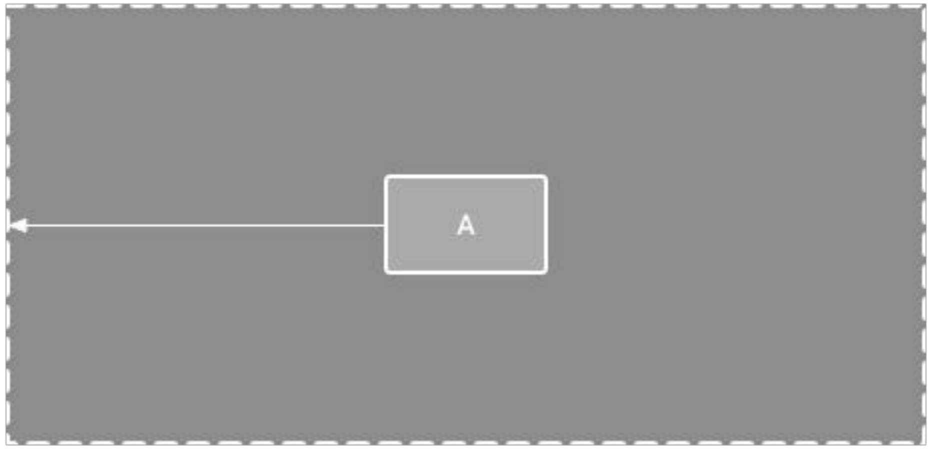
- In the layout editor you’ll see on the right the constraints, and on the left a preview
- *Layouts are drawn according to the available space*

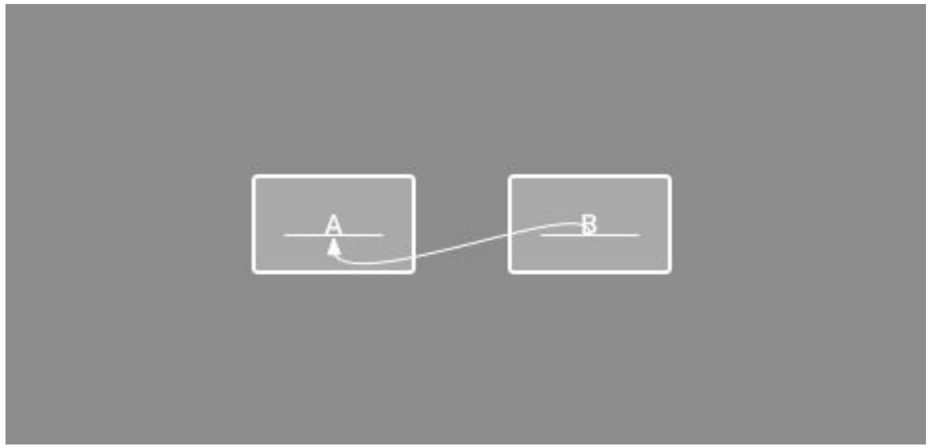
ConstraintLayout: constraints

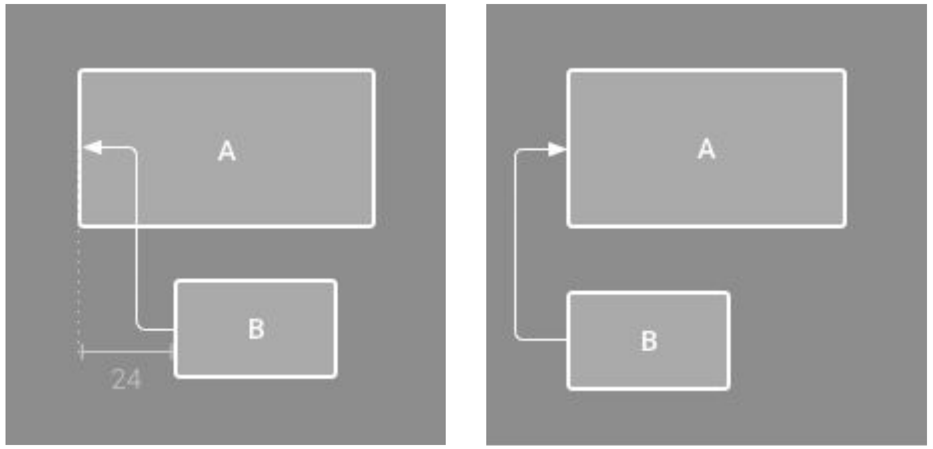
- Each view needs at least one constraint per plane (plane = vertical | horizontal)
- Constraints can be defined only between anchor points sharing the same plane
- Each handle can define one constraint
- Multiple handles can define a constraint to a single anchor point
- Adding 2 opposite constraints places the view in the middle

ConstraintLayout: constraints examples

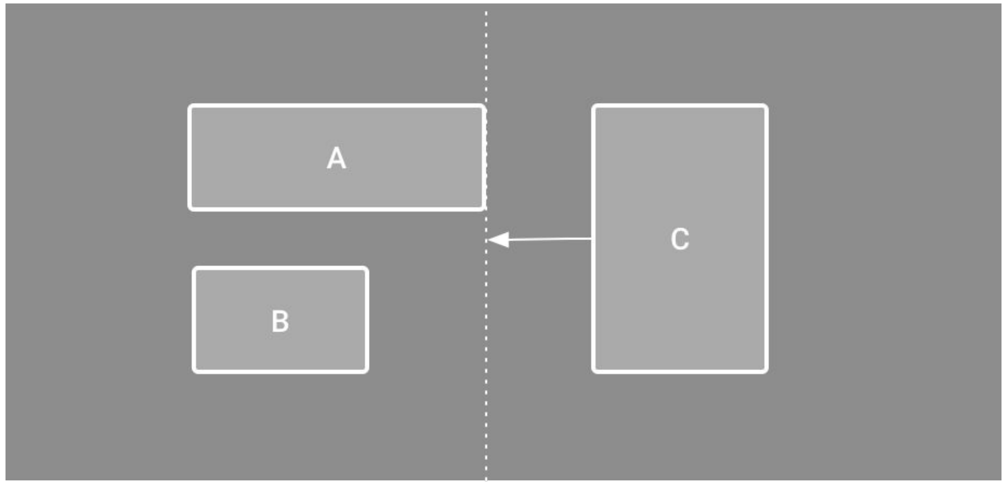


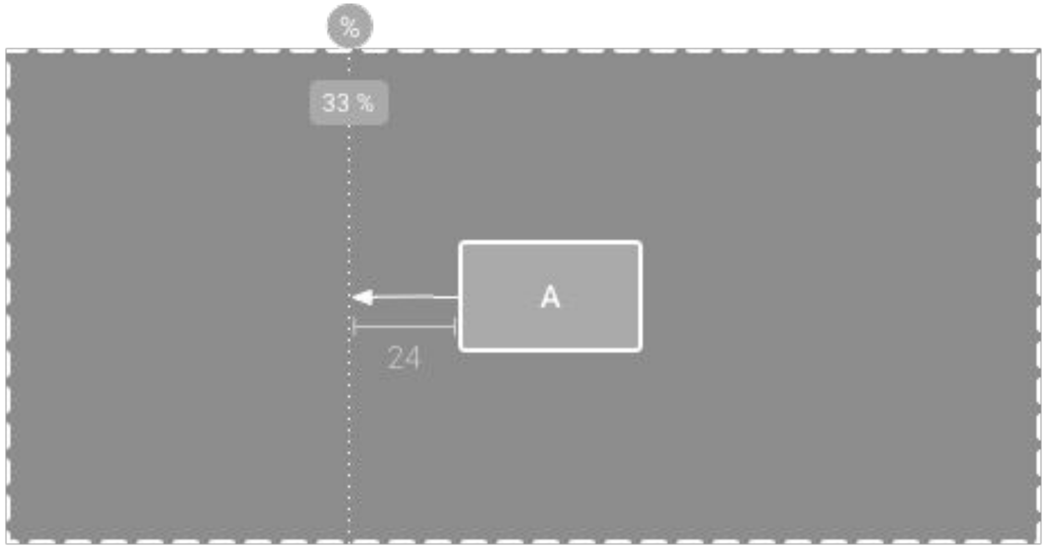


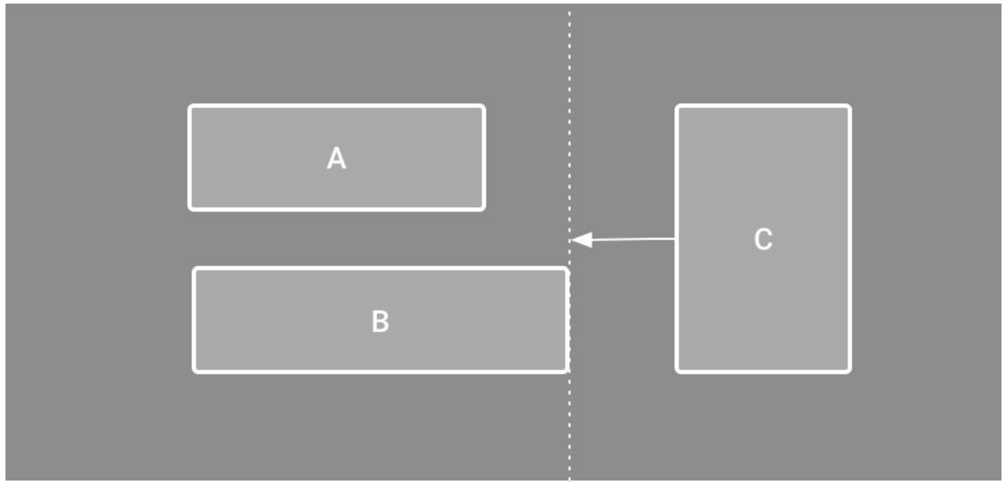


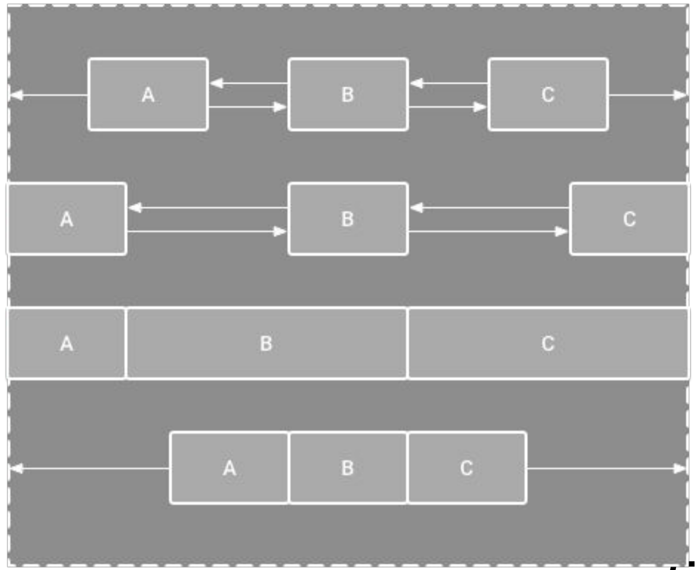


ConstraintLayout: constraints examples









ConstraintLayout

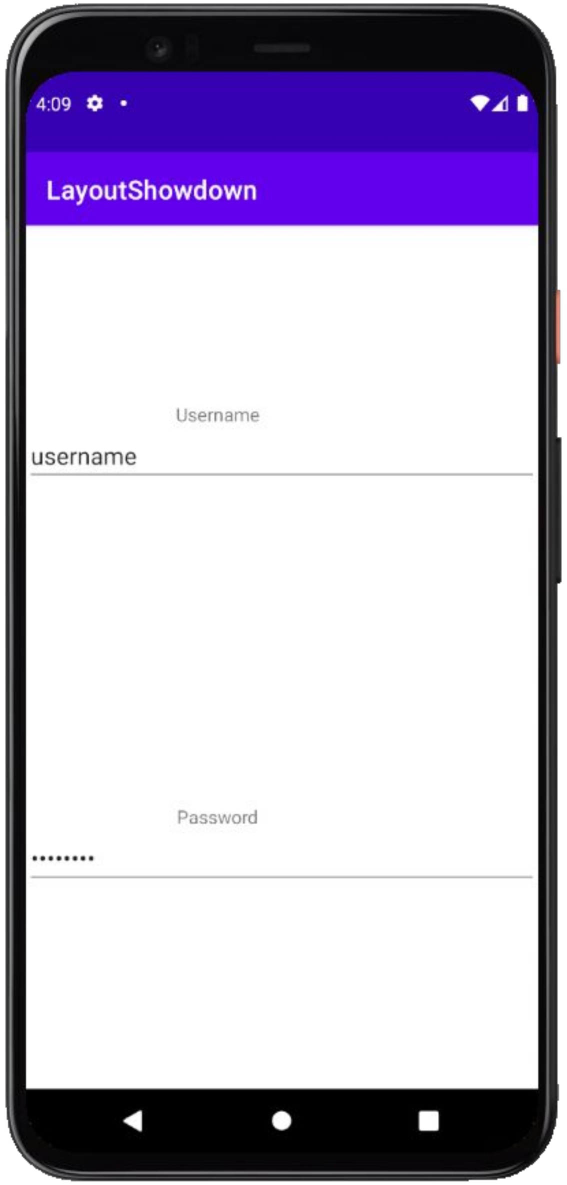
```
<androidx.constraintlayout.widget.ConstraintLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  android:layout_width="match_parent"    android:layout_height="match_parent">

  <androidx.constraintlayout.widget.Guideline
     android:id="@+id/myGuideline"      android:layout_width="match_parent"
     android:layout_height="wrap_content"      android:orientation="vertical"
     app:layout_constraintGuide_percent="0.75" />

  <EditText
     android:id="@+id/username"       android:layout_width="match_parent"
     android:layout_height="wrap_content"       android:layout_toRightOf="@+id/usernameLabel"
     android:inputType="text"       android:text="username"
     app:layout_constraintBottom_toBottomOf="parent"  app:layout_constraintLeft_toLeftOf="parent"
     app:layout_constraintRight_toRightOf="parent"    app:layout_constraintTop_toTopOf="parent"
     app:layout_constraintVertical_bias="0.25" />
```

ConstraintLayout

```
<TextView
    android:id="@+id/usernameLabel"       android:layout_width="wrap_content"
    android:layout_height="wrap_content"      android:text="Username"
    app:layout_constraintBottom_toTopOf="@+id/username"
    app:layout_constraintEnd_toStartOf="@+id/myGuideline"
    app:layout_constraintHorizontal_bias="0.5"  app:layout_constraintStart_toStartOf="parent" />

  <EditText
    android:id="@+id/password"       android:layout_width="match_parent"
    android:layout_height="wrap_content"      android:inputType="textPassword"
    android:text="password"       app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"     app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"     app:layout_constraintVertical_bias="0.75" />

  <TextView
    android:id="@+id/passwordLabel"       android:layout_width="wrap_content"
    android:layout_height="wrap_content"      android:text="Password"
    app:layout_constraintBottom_toTopOf="@+id/password"
    app:layout_constraintEnd_toStartOf="@+id/myGuideline"
    app:layout_constraintStart_toStartOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

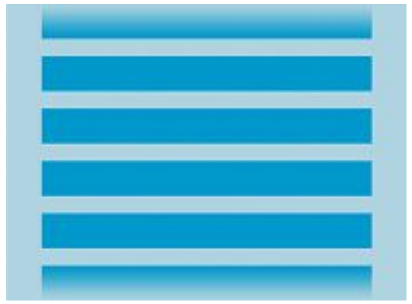
ConstraintLayout

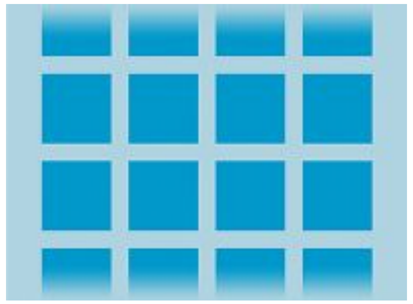
Constraint guideline is invisible...



Dynamic Layouts

- Sometimes the layout needs to be populated at runtime with Views (all the same type of View).
- e.g. ListView, GridView…

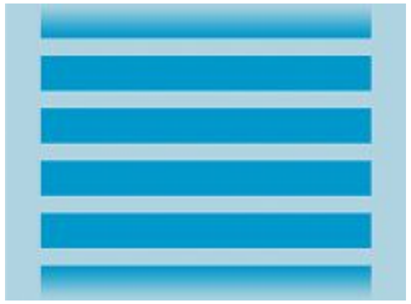


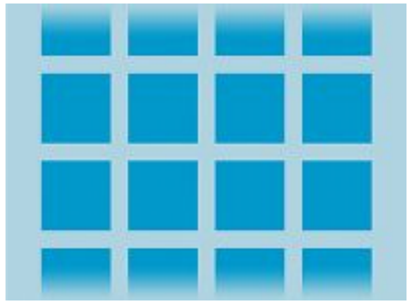


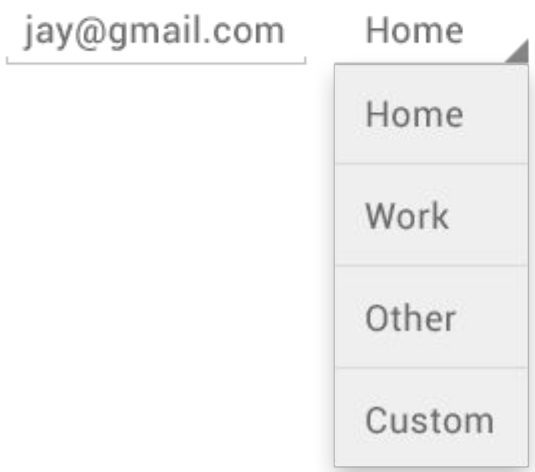
- These Layouts subclass and AdapterView and use an Adapter to retrieve data from another source and maps it into the elements of the AdapterView.

AdapterView

- A ViewGroup subclass
- Its subchilds are determined by an Adapter
- Some subclasses:
  - ListView
  - GridView
  - Spinner
  - Gallery







Adapters

- Used to visualize dynamic data (e.g. ArrayAdapter)
- Make a ViewGroup to interact with data
- Some methods:
  - isEmpty()
  - getItem(int position)
  - getCount()
  - getView()

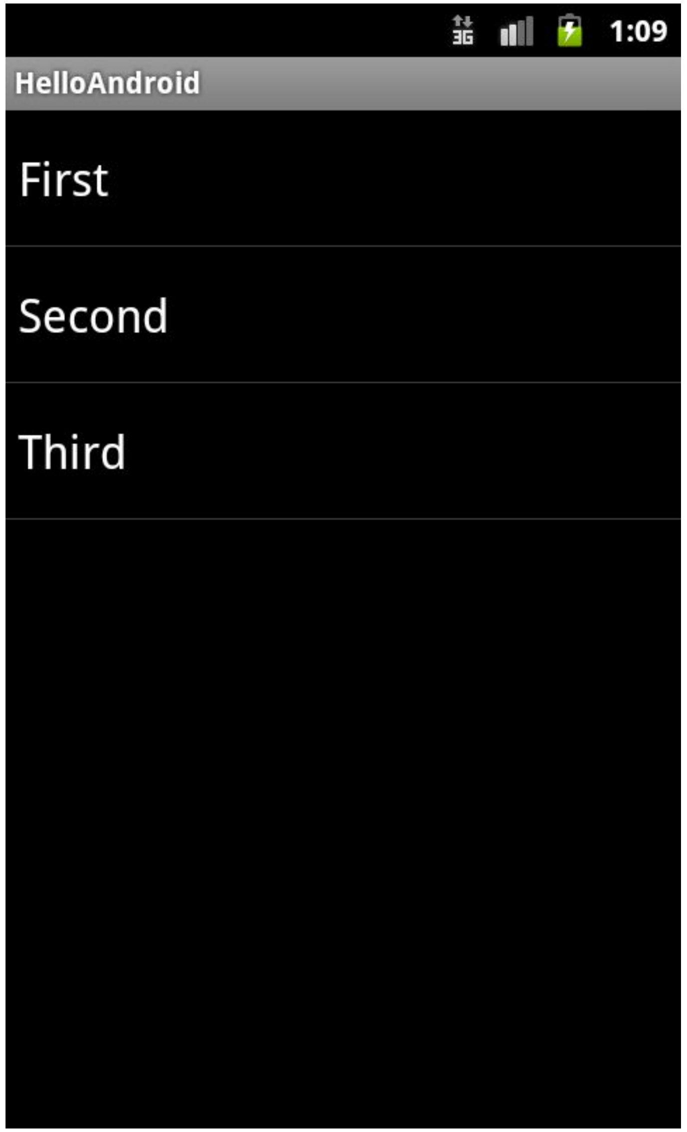
```
ArrayAdapter<String> adapter = new
   ArrayAdapter<String>(this,
   android.R.layout.simple_list_item_1,
   myStringArray);
ListView listView = (ListView)
   findViewById(R.id.listview);
listView.setAdapter(adapter);
```

- You can use SimpleCursorAdapter in case the data structure is a Cursor from a DB query.

ListView example

```
public class HelloAndroidActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.list);

       String[] data = {"First", "Second", "Third"};
       ListView lv = (ListView)findViewById(R.id.list);
       lv.setAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, data));
    }
}

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

ListView



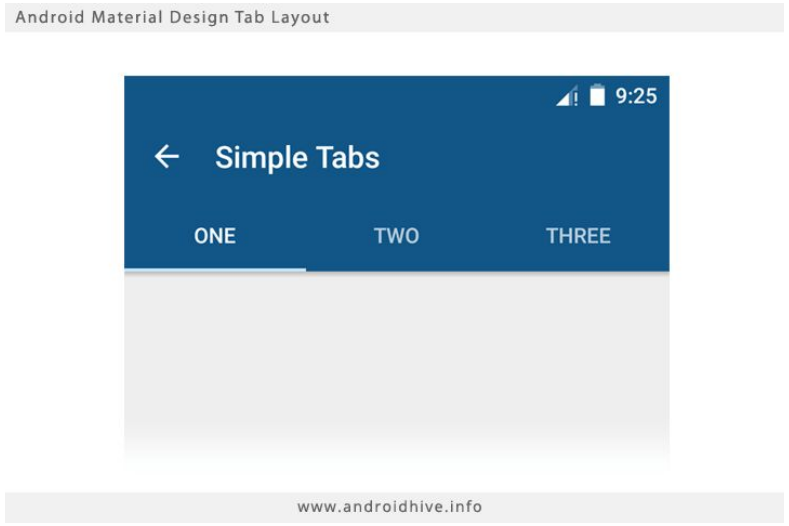
First

Second

Third

Other views/adapters

- Spinner, selection of multiple items
- Gallery, images
- ExpandableListView, list with hidden values
- TabWidget, tabbed layouts



RecyclerView

- ListView available since API version 1
- Since Lollipop, RecyclerView has been introduced
  - Better handling of events
  - Separates data and layout
- Start with

```
dependencies {
    implementation "androidx.recyclerview:recyclerview:1.1.0"
}
```

- Then add a **RecyclerView** to the Layout

Step 1: LayoutManager

- For each RecyclerView, we have to define a LayoutManager
  - “A LayoutManager measures and positions item views on the RecyclerView. It also handles view focus and visibility”
- In simple words, it is responsible for placing items in the layout
- Examples: LinearLayoutManager, GridLayoutManager, StaggeredGridLayoutManager, WearableLinearLayoutManager

Step 2: Adapter

- Create a class that extends RecyclerView.Adapter
  - Extend also RecyclerView.ViewHolder (a structure carrying the view of each item and its metadata such as position...)
- Override some methods:
  - getItemCount()
  - onCreateViewHolder()
    - Creates a new ViewHolder item (refer to elements in the dedicated layout)
  - onBindViewHolder()
    - Bind the appropriate data to the ViewHolder (give a behavior to these elements)

Differences: ListView and RecyclerView

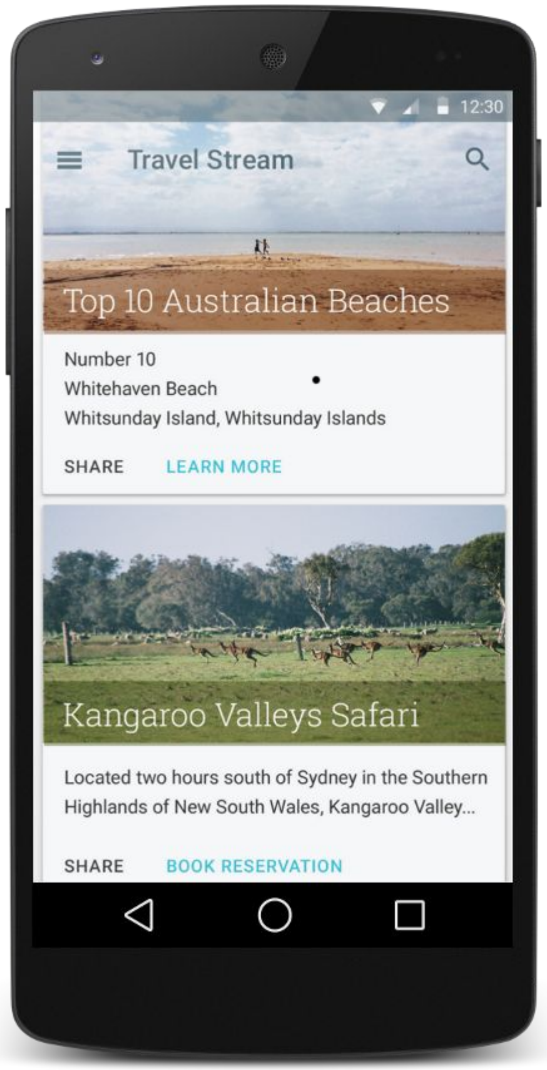
- More efficient
- LayoutManager flexibility: think about LinearLayout and GridLayout
- Possible to add custom Decorations
- Animations made easy
- More than just notifyDataSetChanged()
  - notifyItemInserted(), notifyItemRemoved(), notifyItemChanged() and more

CardView

- A CardView is a ViewGroup
- It contains views
- Need to add

```
dependencies {
    implementation "androidx.cardview:cardview:1.0.0"
}
```

- Useful to group content related to the same entity
- Needless to say, you can do a RecycleView of CardViews

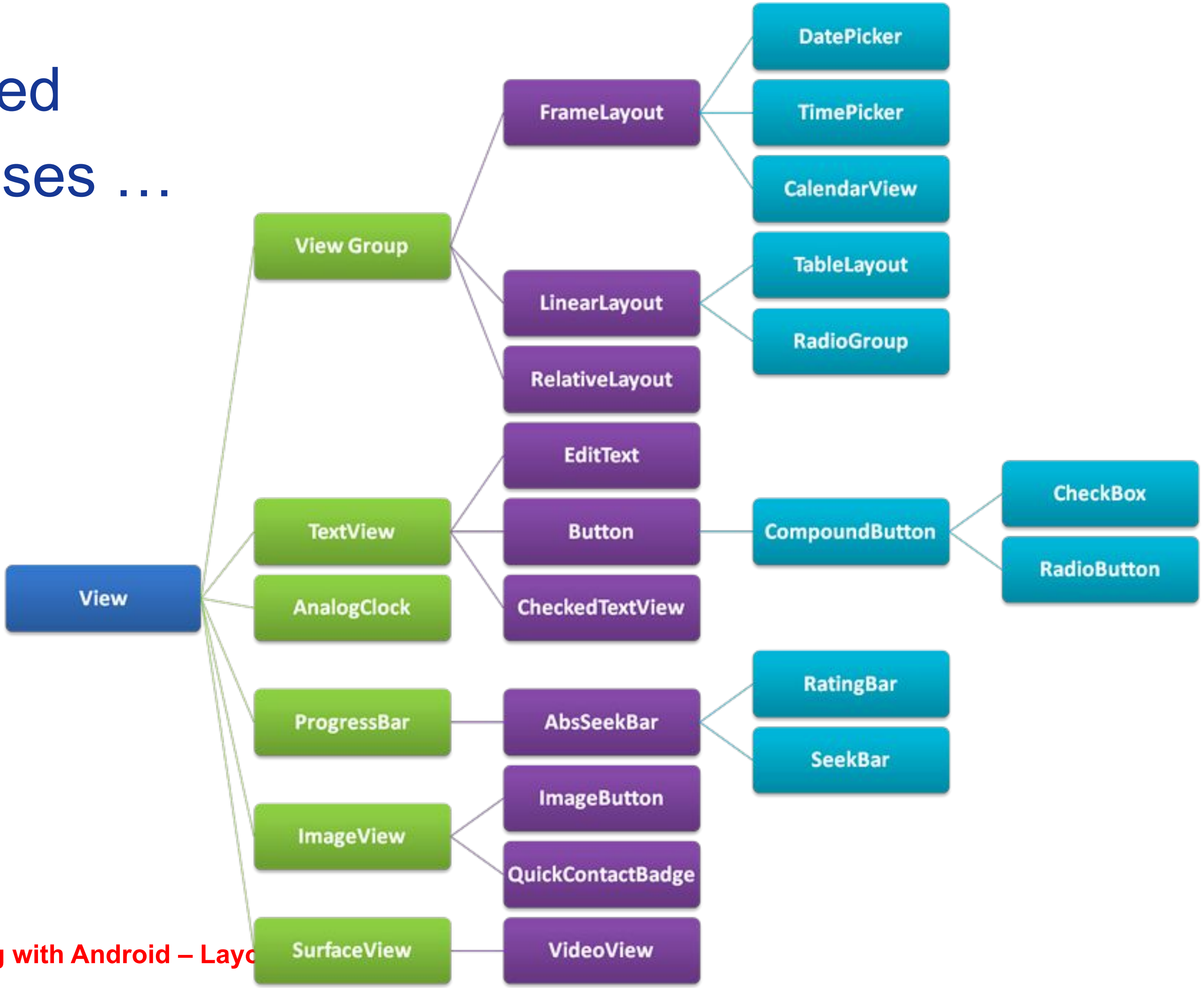


Number 10
Whitehaven Beach
Whitsunday Island, Whitsunday Islands

Located two hours south of Sydney in the Southern Highlands of New South Wales, Kangaroo Valley...

Views: Hierarchy of the classes …

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



Views: TextView

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

- Can 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”
/>
```

Views: 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 **setHint**(CharSequence hints)

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

Views: Linkify elements

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

Views: 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”
/>
```

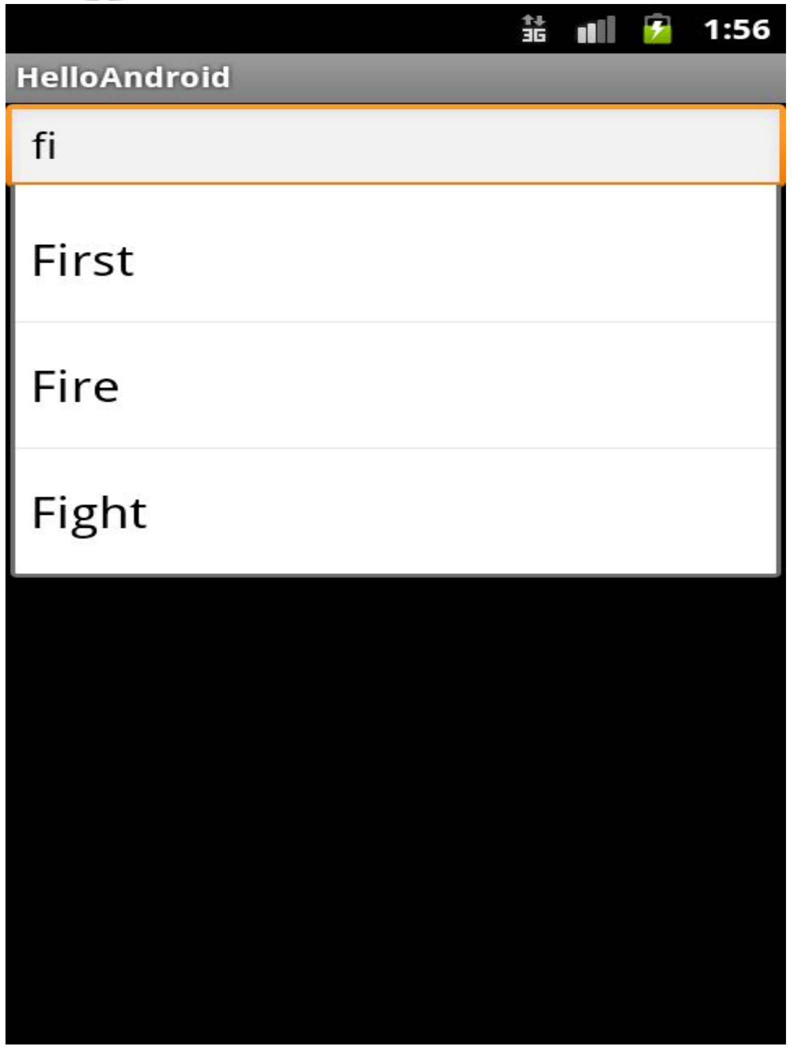
Views: AutocompleteTextView

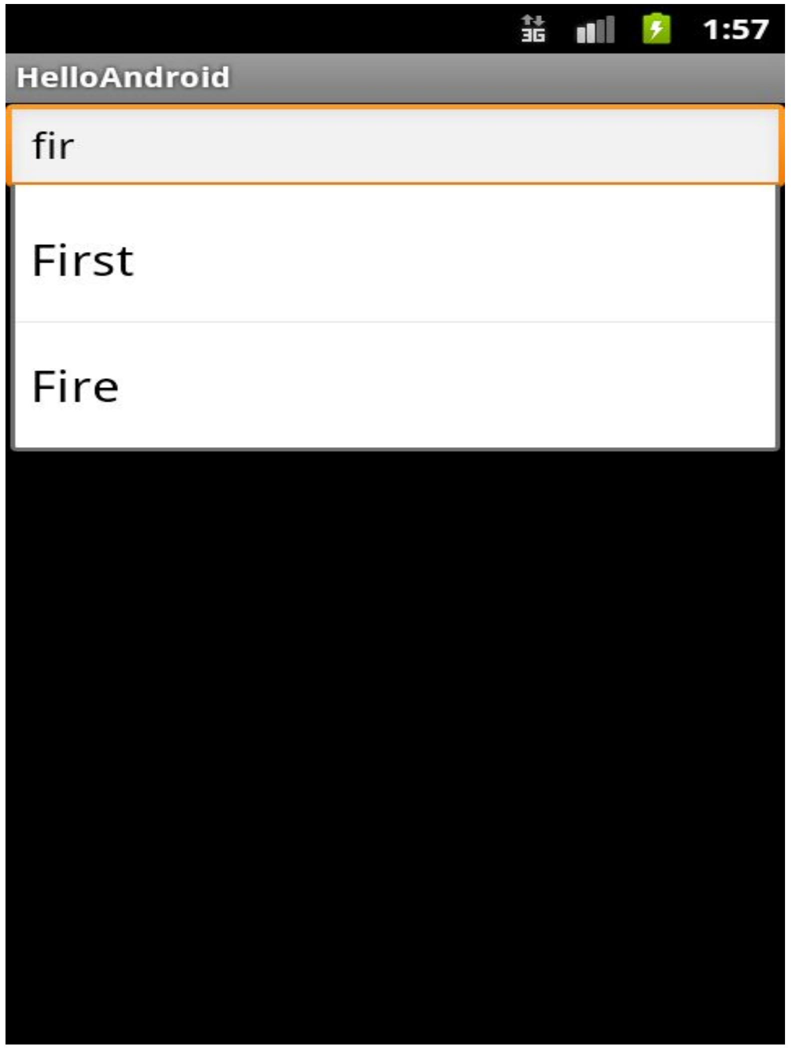
- **XML** tags: **<AutoCompleteTextView> </Auto…View>**

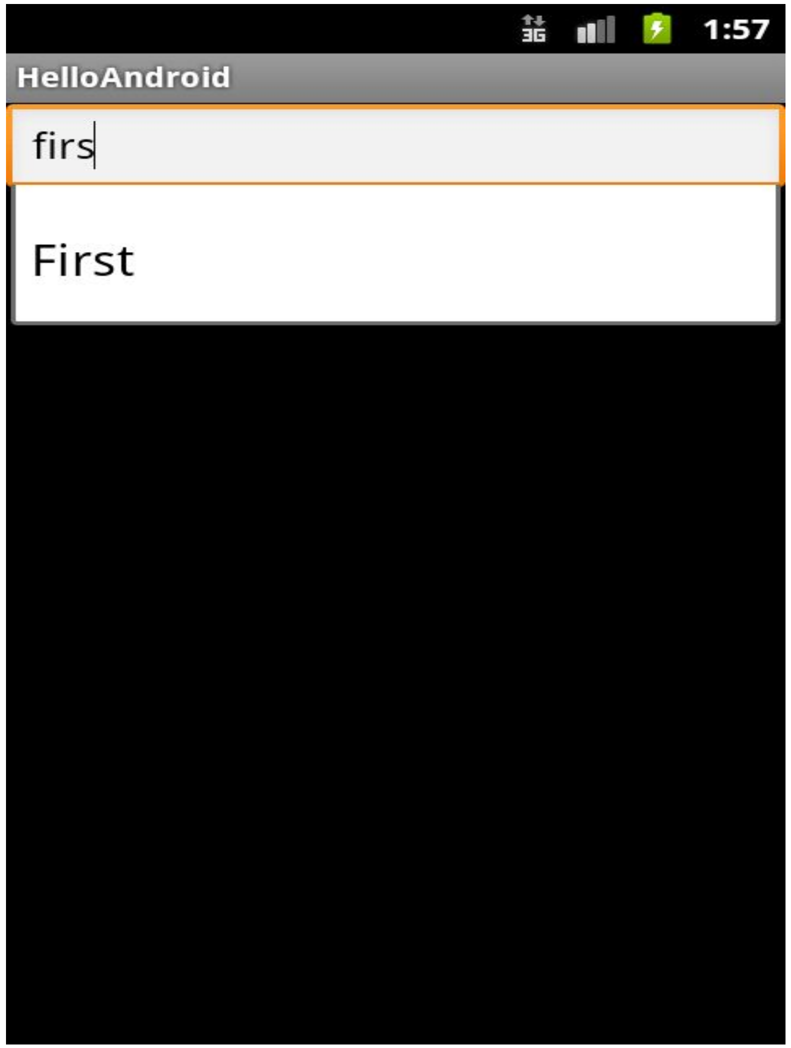
- Used to make easier the input by the users …
  - As soon as the user starts typing, hints are displayed
- A list of hints is given through an **Adapter**

```
 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);
```

Views: AutocompleteTextView







Views: 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

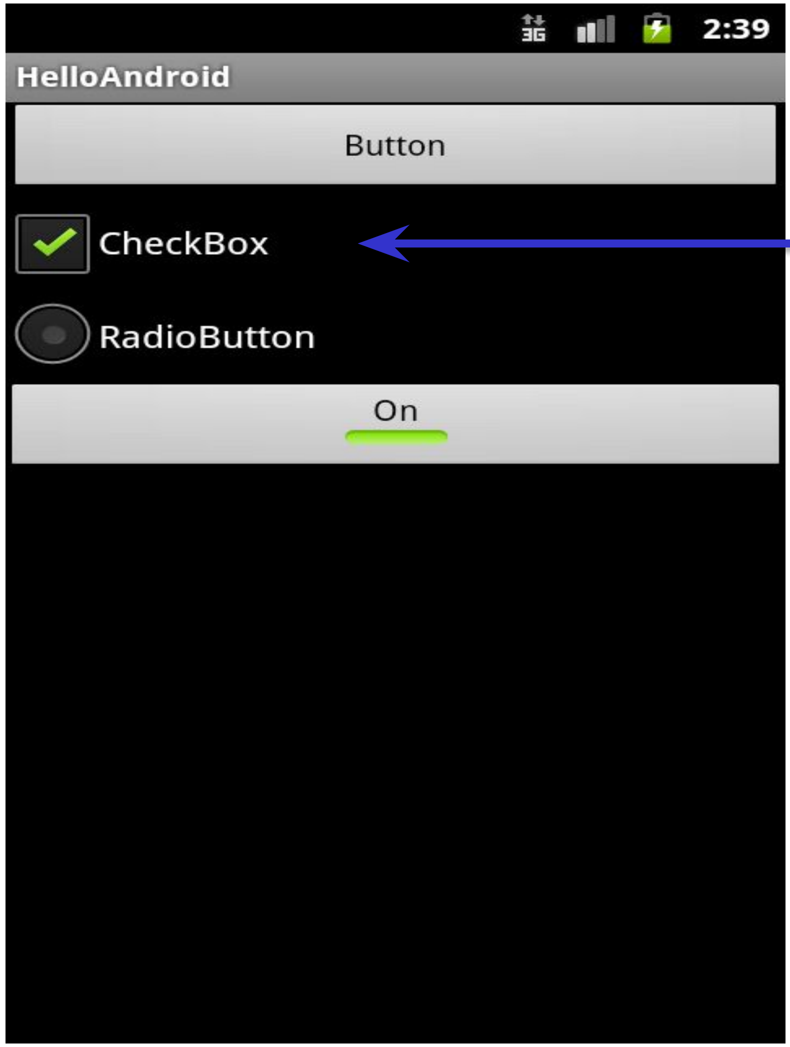
```
<Button
      android:text="@string/textButton"
      android:id="@+id/idButton"
      android:background="@color/blue”
/>
```

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

res/color/blue.xml

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

Views: Button and CompoundButton

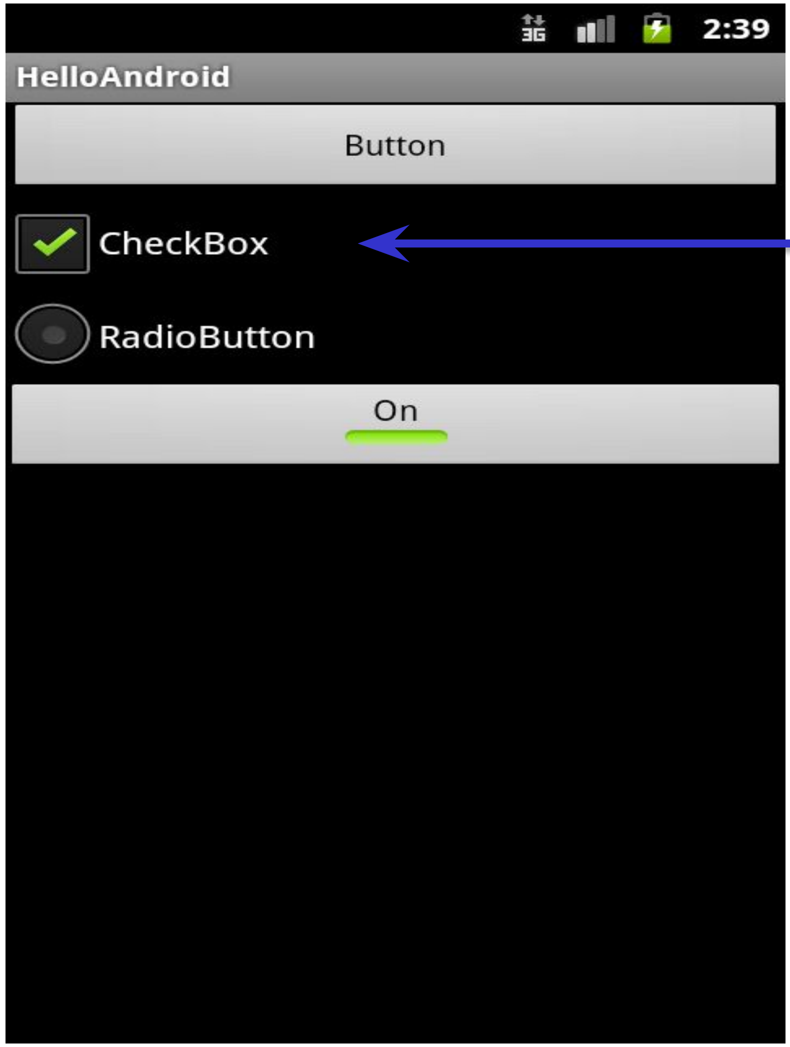


checkBox CompoundButton

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”
/>
```

Views: Button and CompoundButton

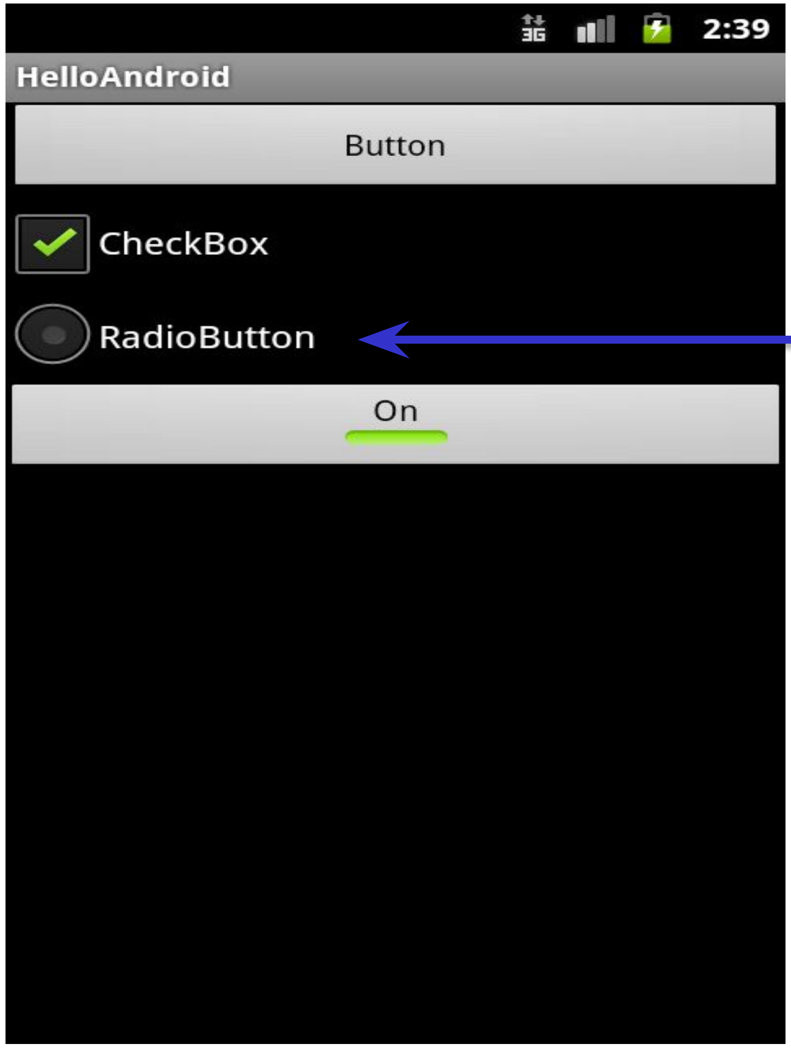


checkBox CompoundButton

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

Listener:
onCheckedChangeListener

Views: Button and CompoundButton

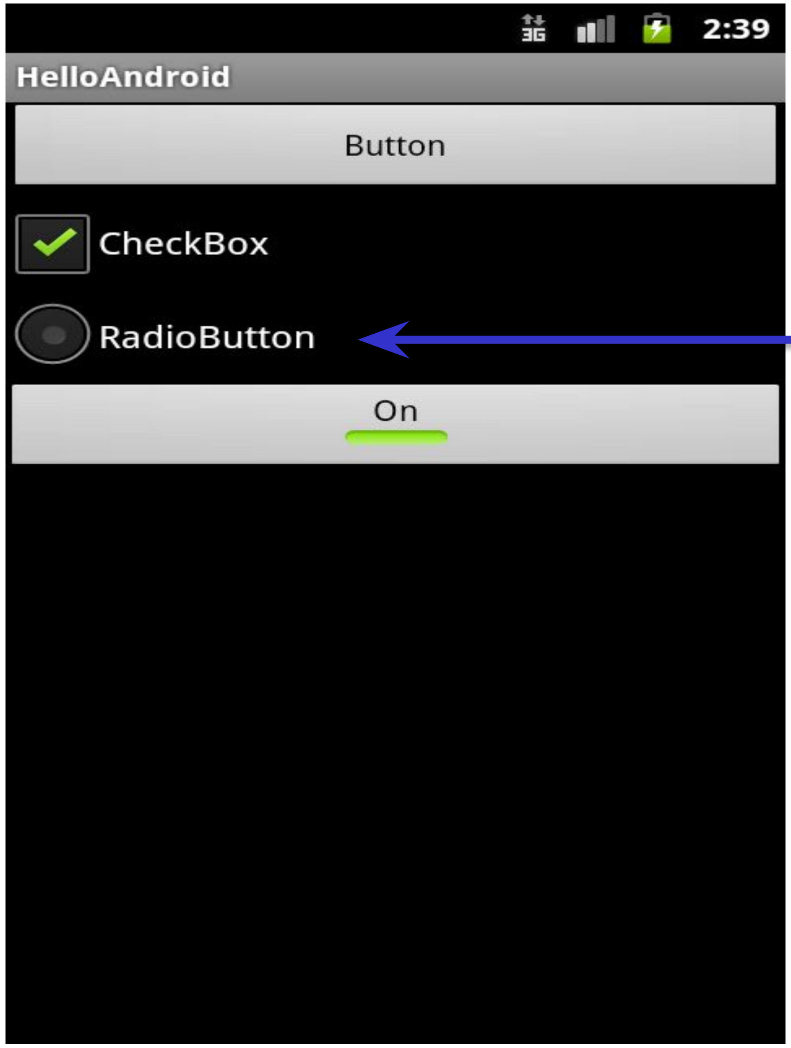


radioButton CompoundButton

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"
          />
```

Views: Button and CompoundButton

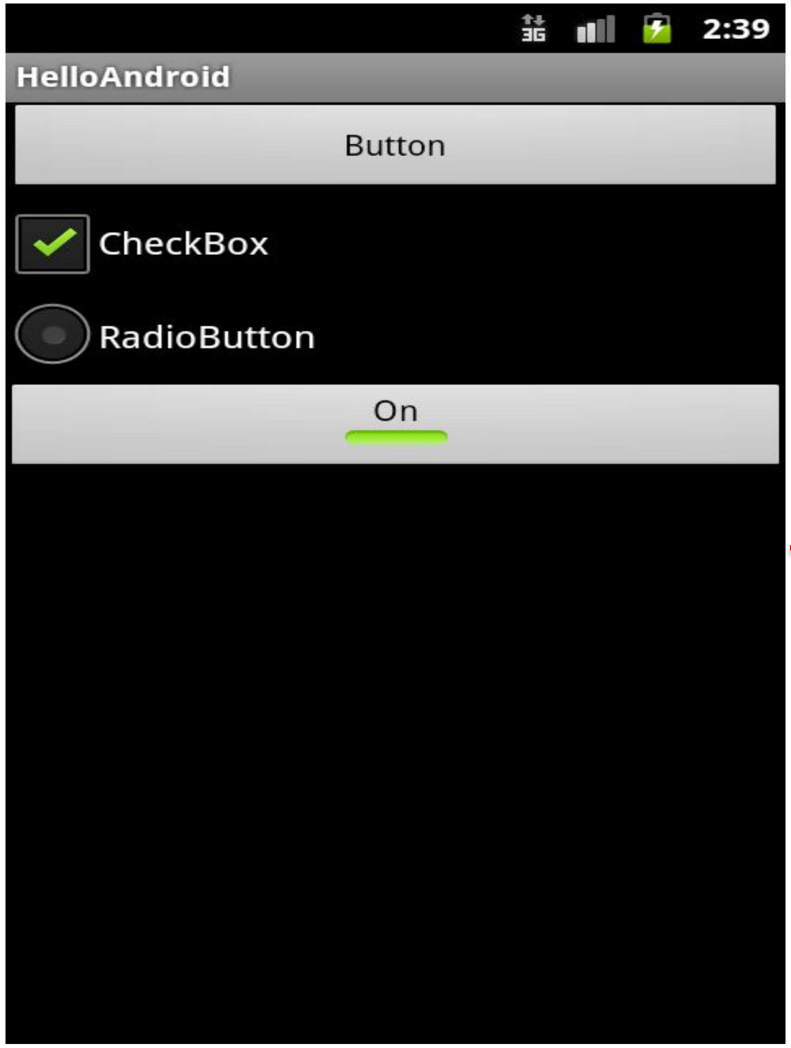


radioButton CompoundButton

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

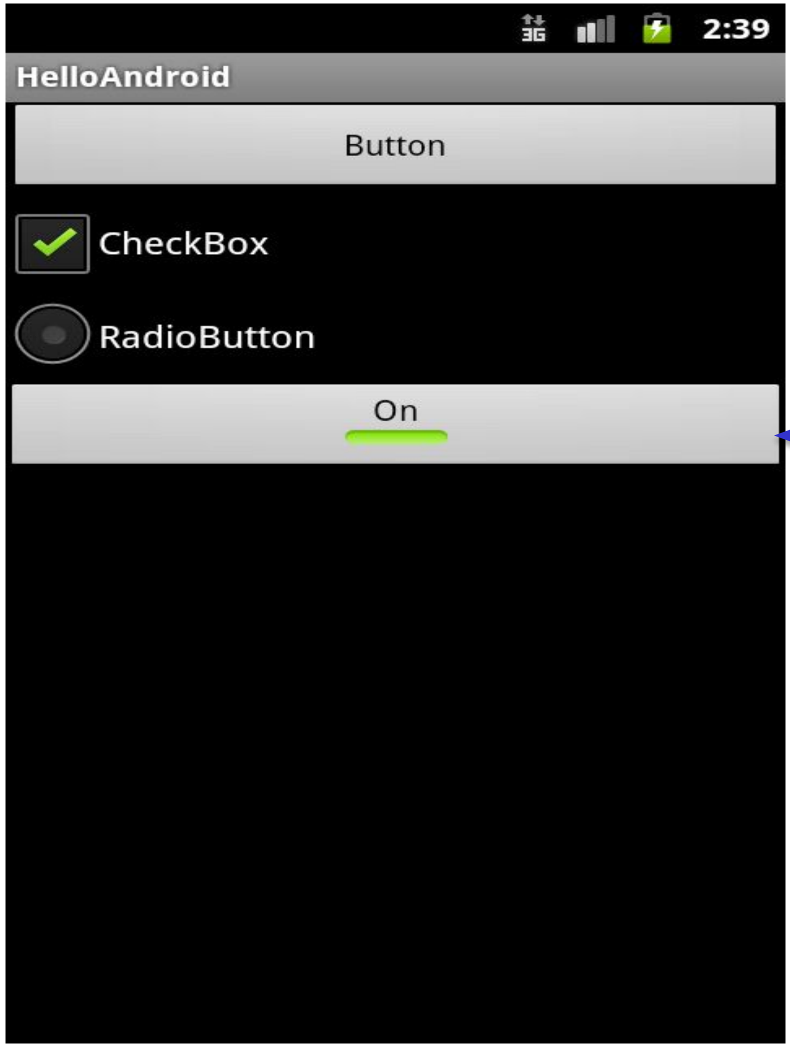
Listener:
OnCheckedChangeListener

Views: 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>
```

Views: Button and CompoundButton

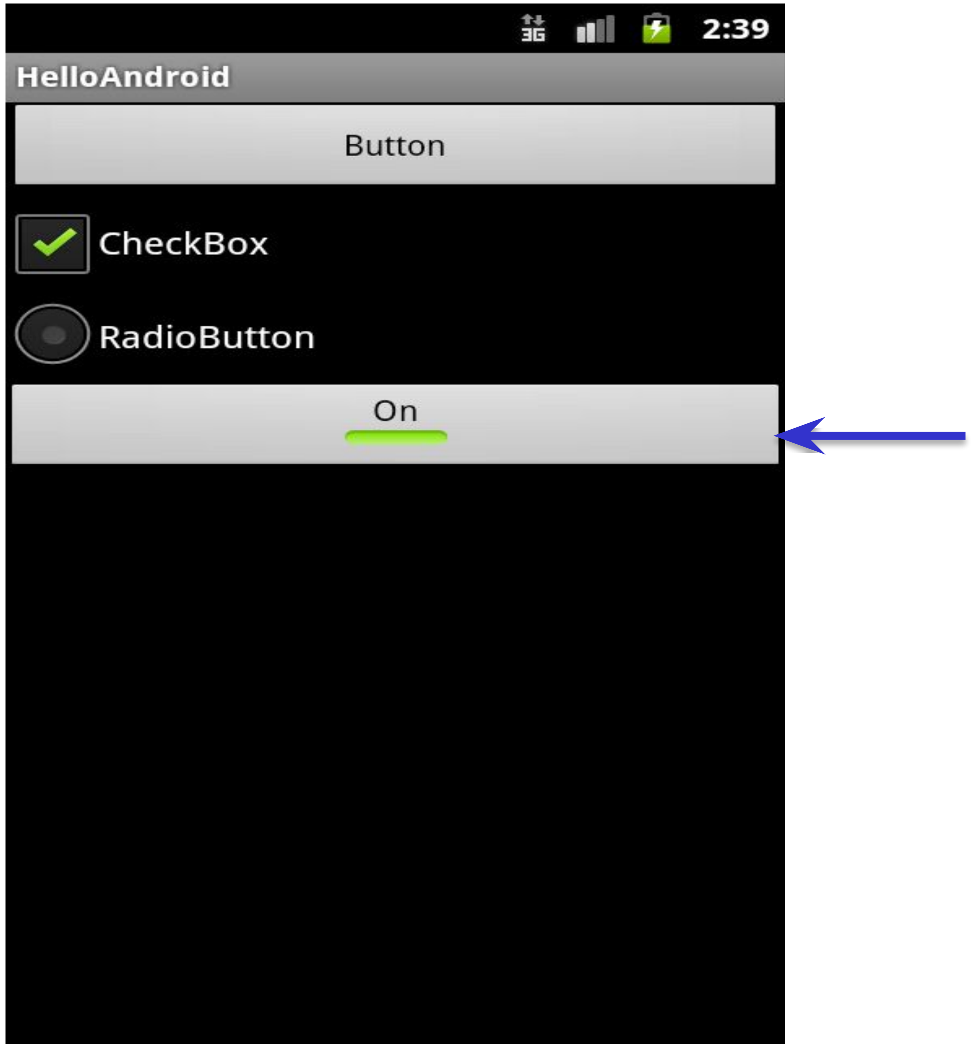


toggleButton CompoundButton

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”
/>
```

Views: Button and CompoundButton



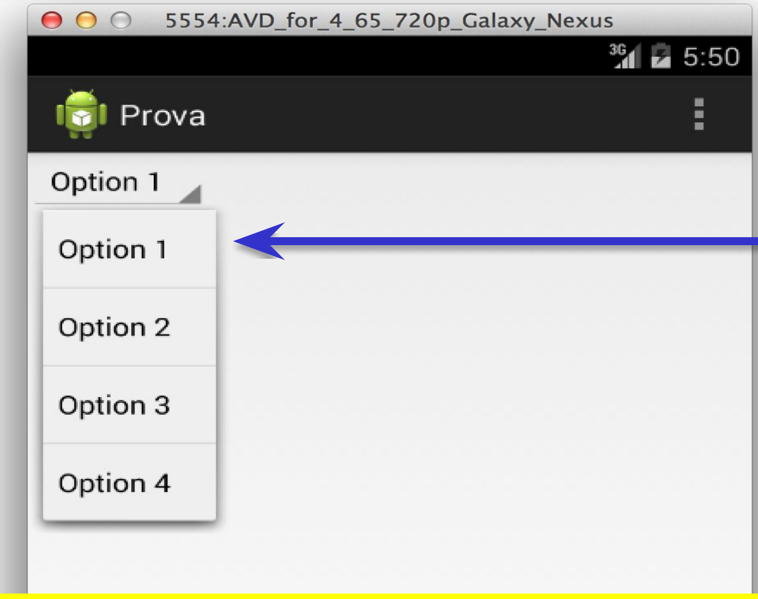
toggleButton CompoundButton

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

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

Listener:
OnCheckedChangeListener

Views: Spinners



Spinner component

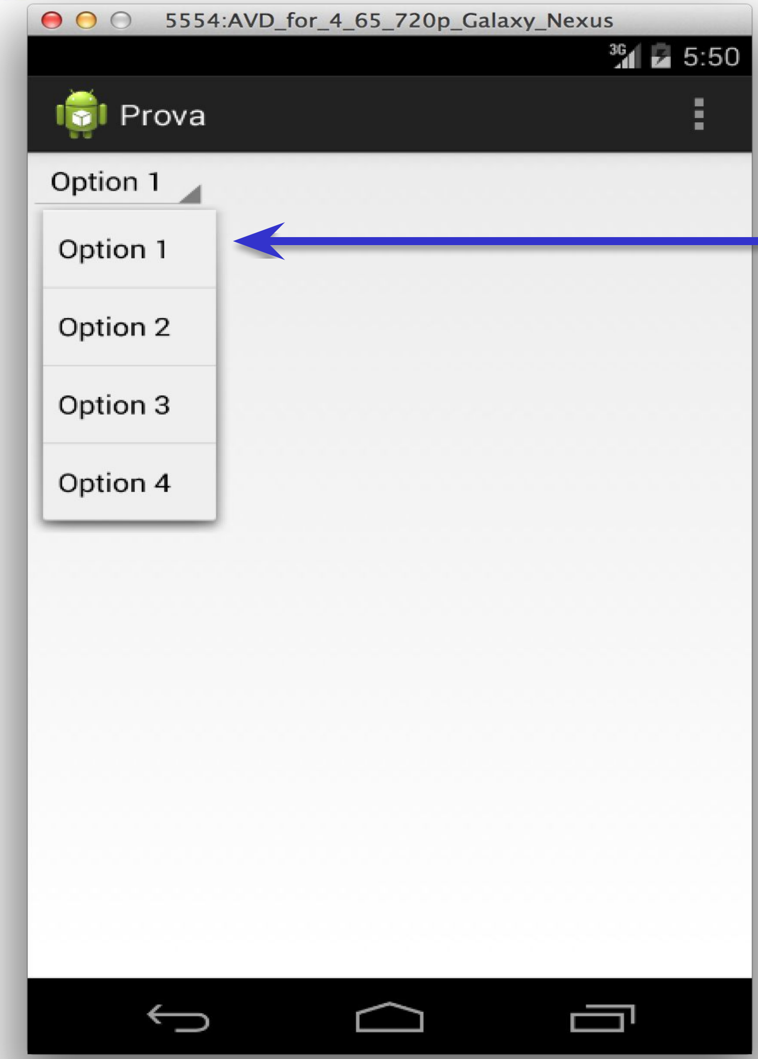
XML tags: **<Spinner>** **</Spinner>**

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

res/values.xml

Views: Spinners



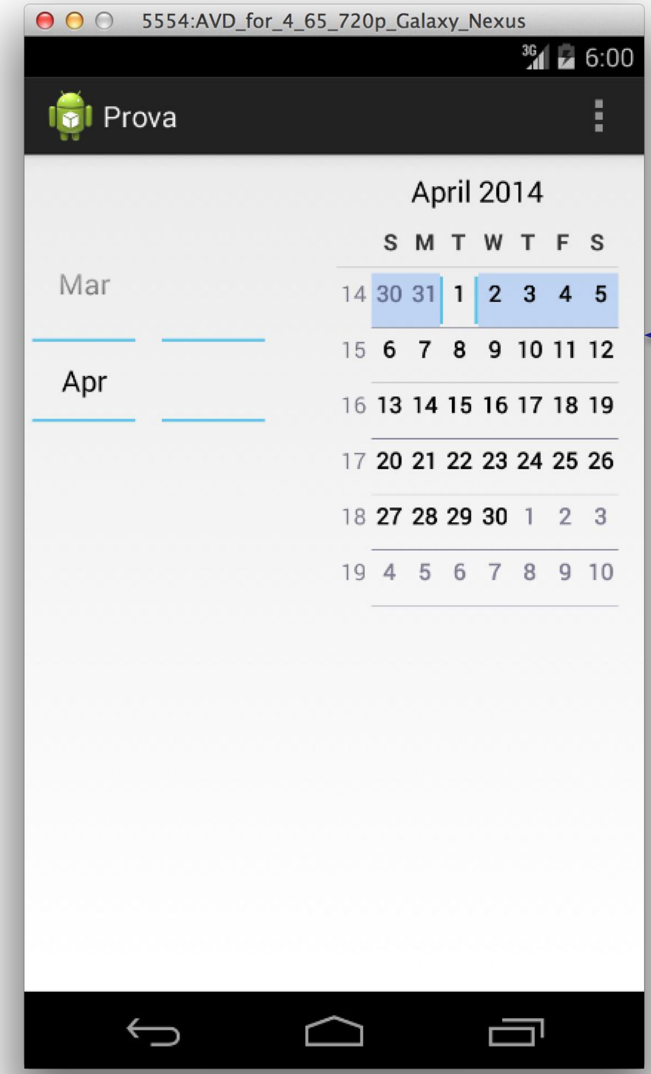
Spinner component

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

Views: Button and CompoundButton

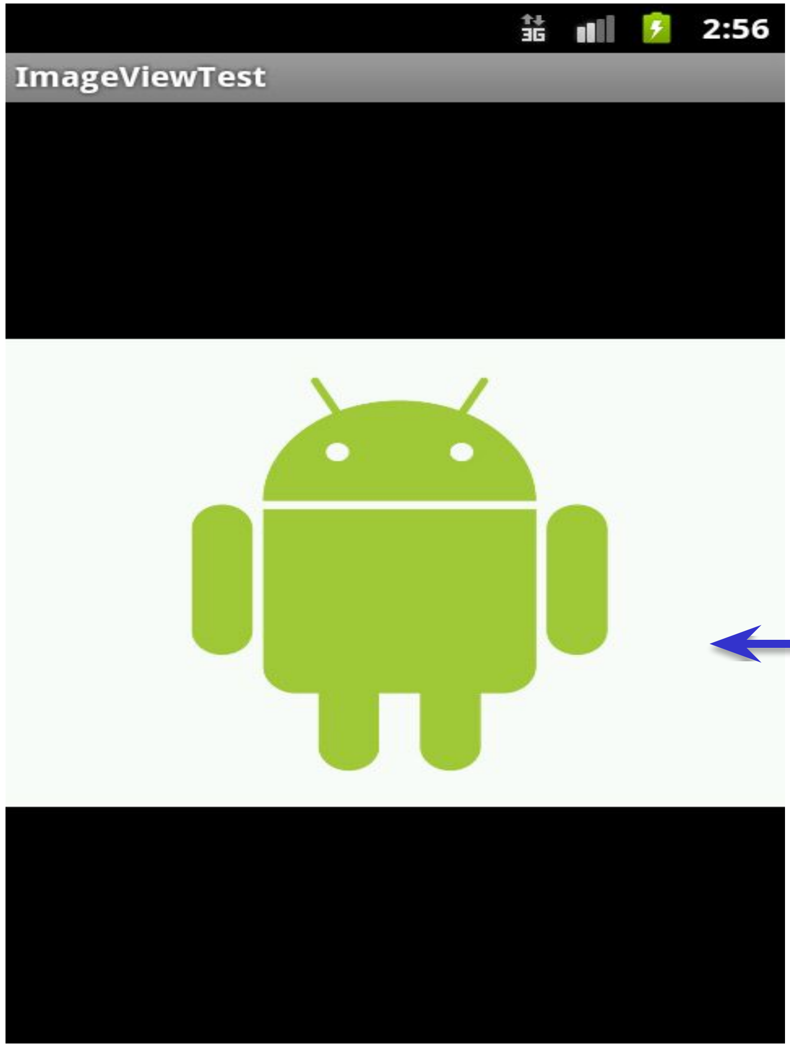


DatePicker component

XML tags: **<DatePicker>**
</DatePicker>

```
<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”
/>
```

Views: ImageView



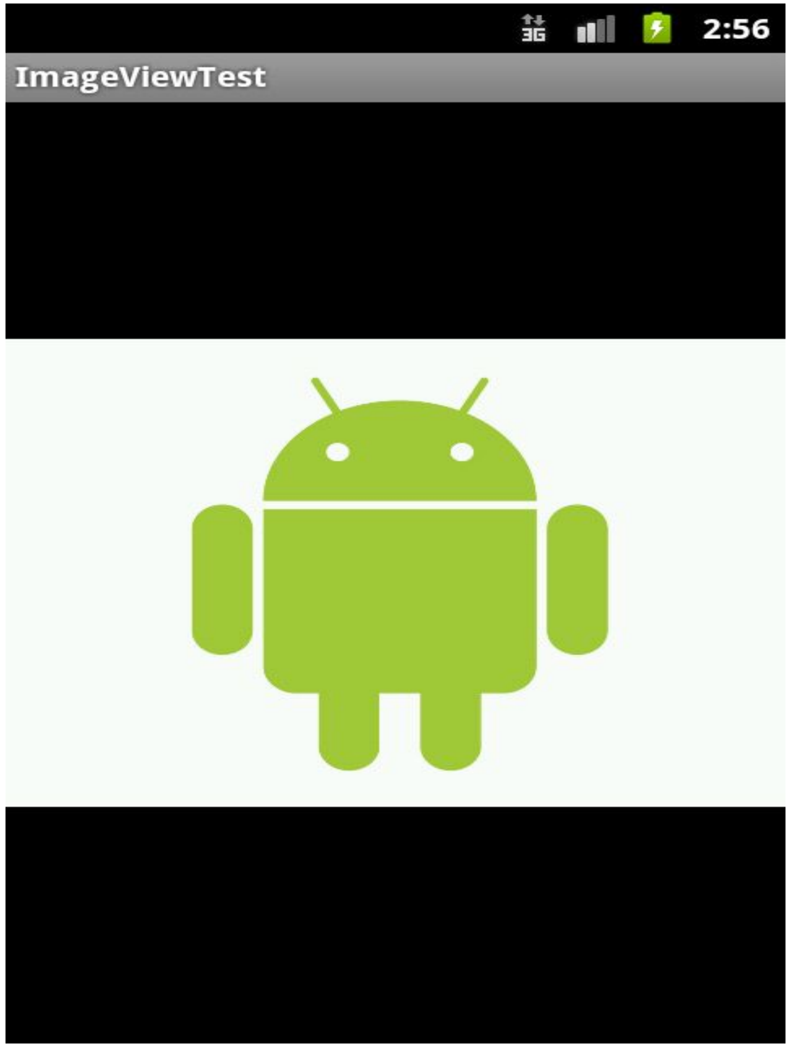
ImageView component

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

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

Source: **android**.jpg in drawable/

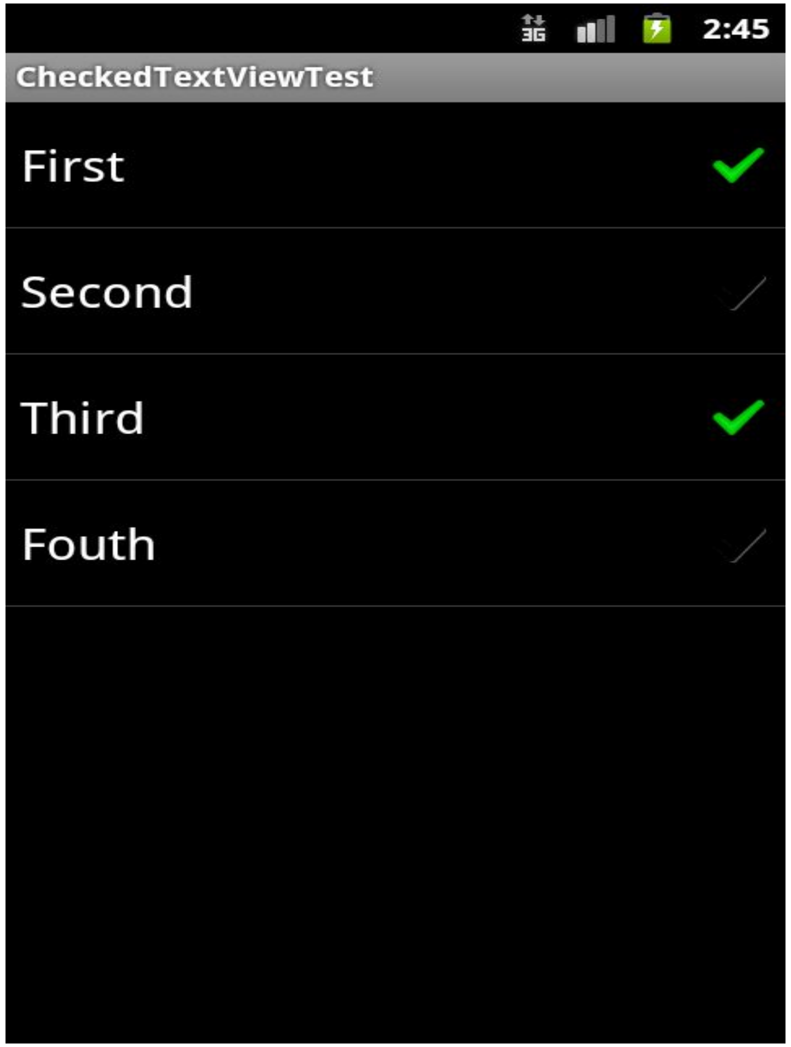
Views: ImageView



- **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

Views: CheckedTextView

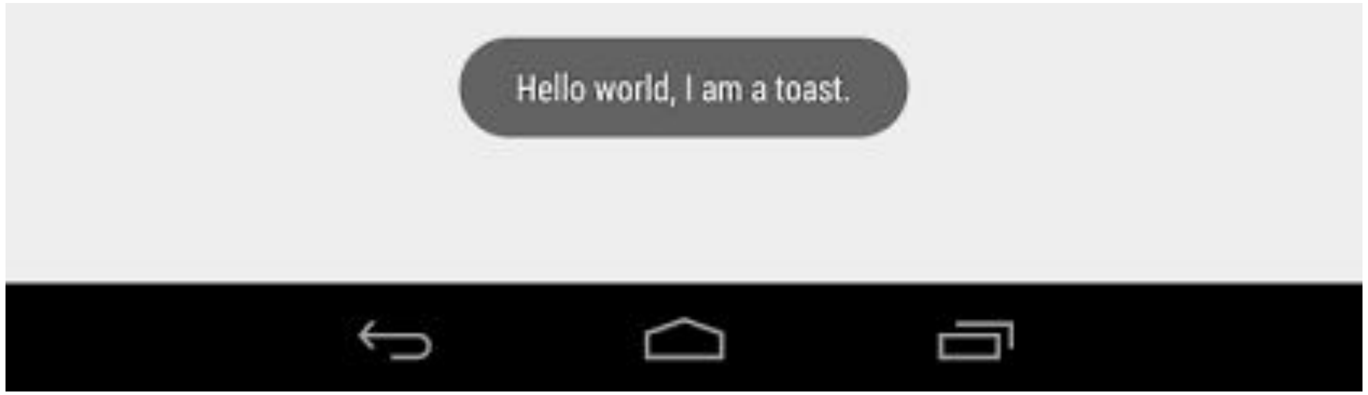


- **Checkable** version of a TextView
- Usable with a **ListView Adapter**
  - *Multiple* or *single* selection of items (CHOICE_MODE_SINGLE, CHOICE_MODE_MULTIPLE)
- Methods:
  - void setChoiceMode(int choiceMode)
  - long[] getCheckItemIds()
  - int getCheckedItemPosition()

Toast: making a toast

- Tiny messages over the Activity
- Used to signal to the user confirmation, little errors
- Can control the duration of the Toast
- As simple as:

```
Toast.makeText(this, “Hello world, I am a toast.”, Toast.LENGTH_SHORT).show();
```


Hello world, I am a toast.