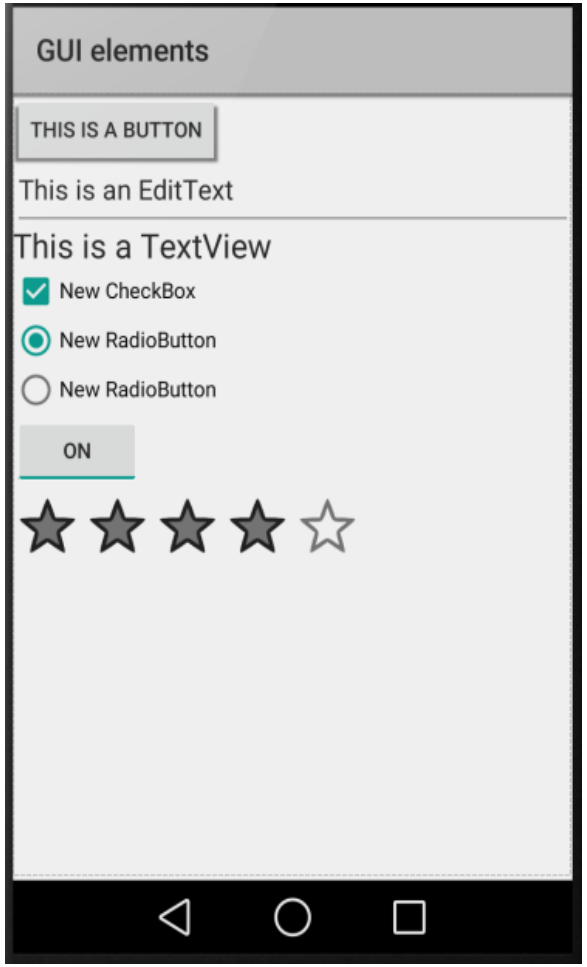
Android GUIs

Some GUI elements (widgets)

- All widgets are subclasses of *View*

- *Button:* standard button that can be clicked by the user
- *EditText:* an editable text
- *TextView:* read-only text label
- *CheckBox:* a two-state element (checked or unchecked)
- *RadioButton:* a two-state grouped button, only one can be enabled at a time
- *ToggleButton:* used for toggling between two states
- *RatingBar:* the user can touch to set the rating
- *Spinner:* lets you select an item from a list to display in the textbox



Declaring a widget

- Declaring a widget in XML

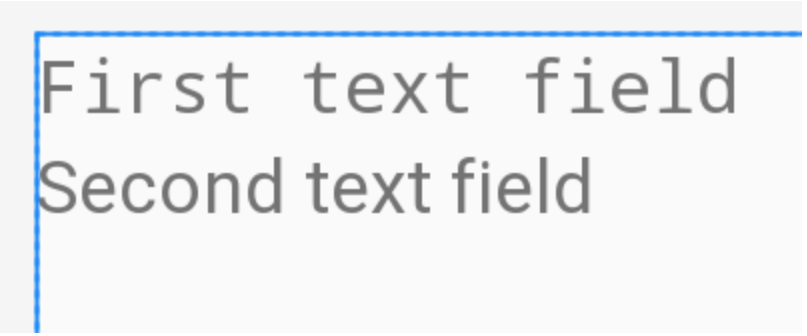
Type of widget, e.g. TextView

```
<widget_type

  List of attributes (e.g. format, width, length, etc)
  . . .
  . . .
/>
```

TextView

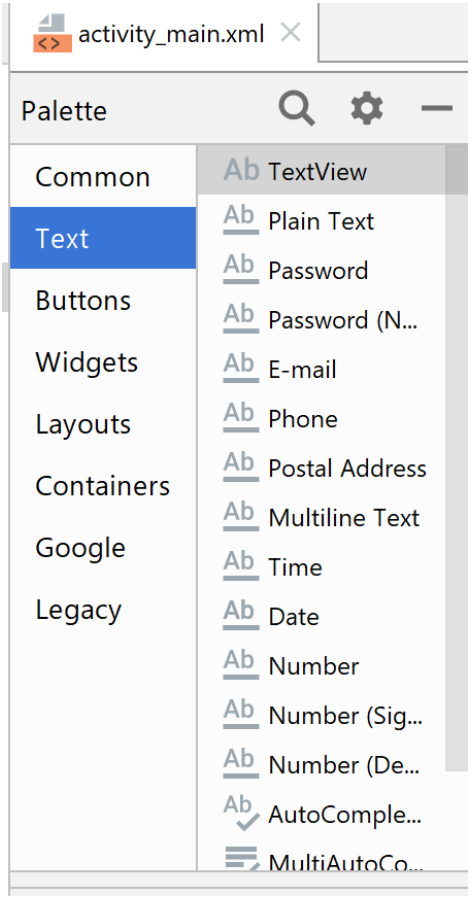
- Label, no interaction
- Common attributes:
  - *layout_width*:
  - *layout_height*:
  - *textColor*: e.g. #FF0000
  - *typeface*: monospace, serif, …

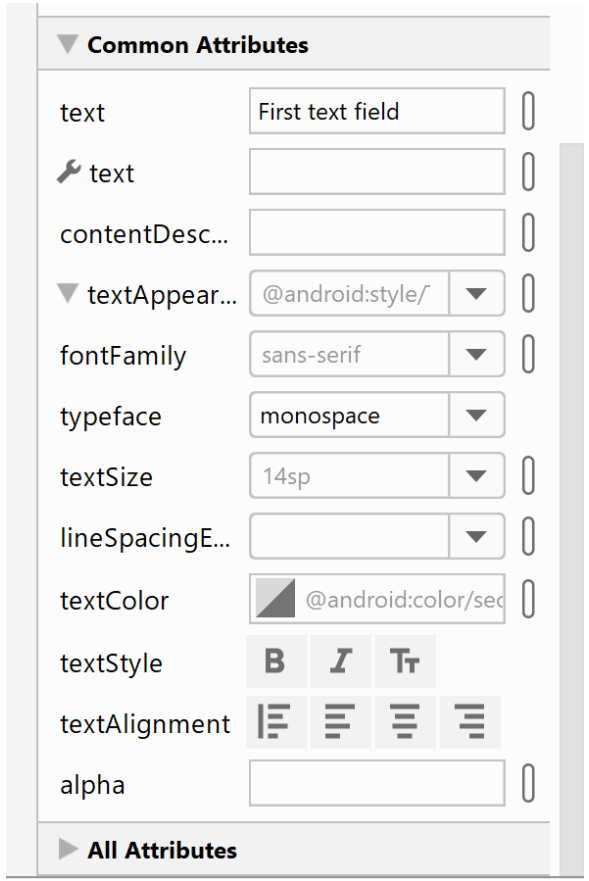
```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="First text field"
    android:typeface="monospace"
 />
```



TextView

- Drag&drop from palette then set properties

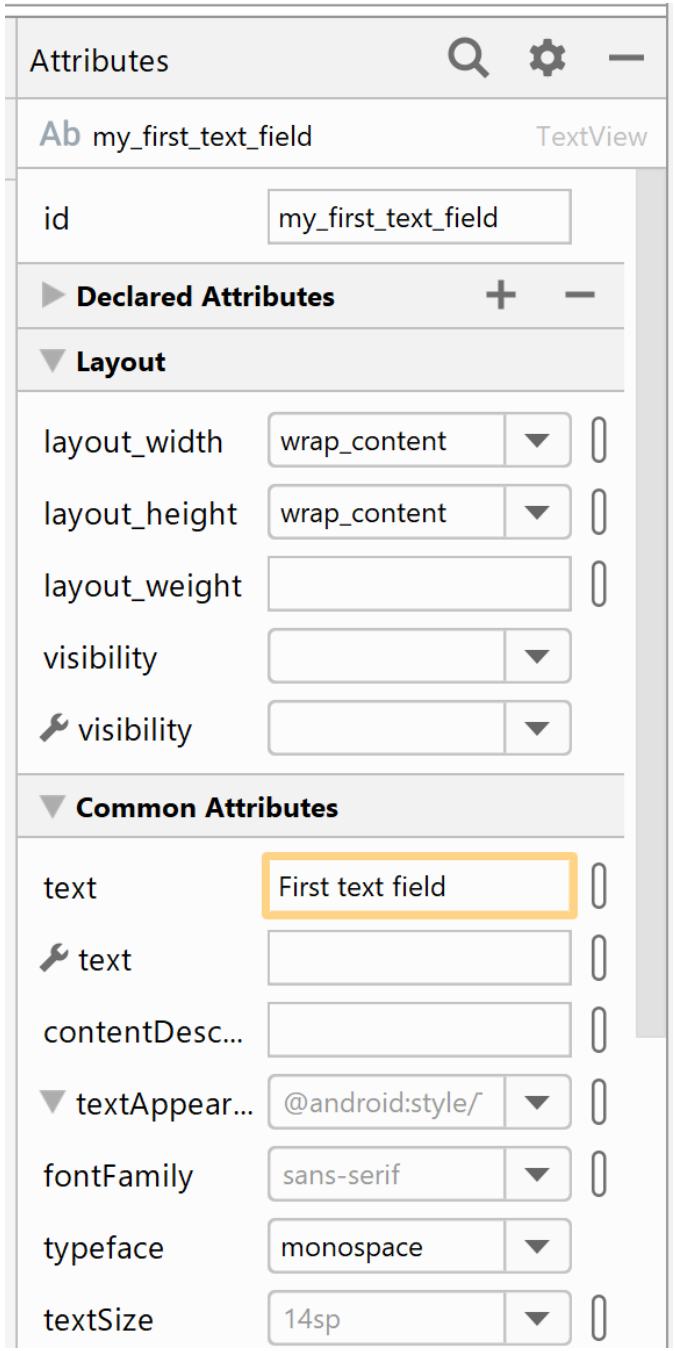




TextView

- You have to provide an **id** to all the widgets you want to use in Java code
  - Extract information
  - Modify appearance
  - …

```
<TextView
    android:id="@+id/my_first_text_field"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="First text field"
    android:typeface="monospace"
/>
```



Button

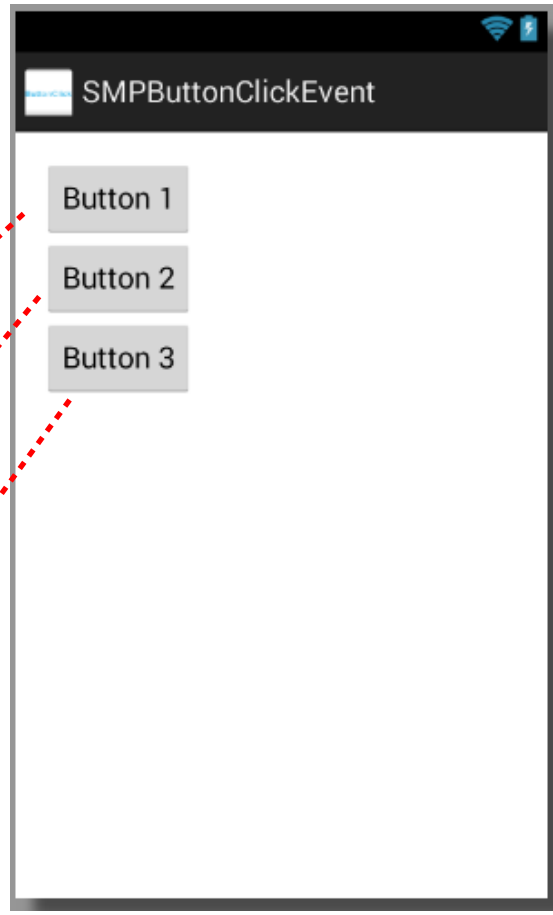
- Button with text, image, or both
- Appearance can be customized in the same way of TextField
- Button is a subclass of TextView, attributes are similar (width, height, color, …)

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Press me!"
/>
```

Button

- To specify the receiver of events, two options:
  - XML-based
  - Java-based
- Code to be executed is always in the Java part

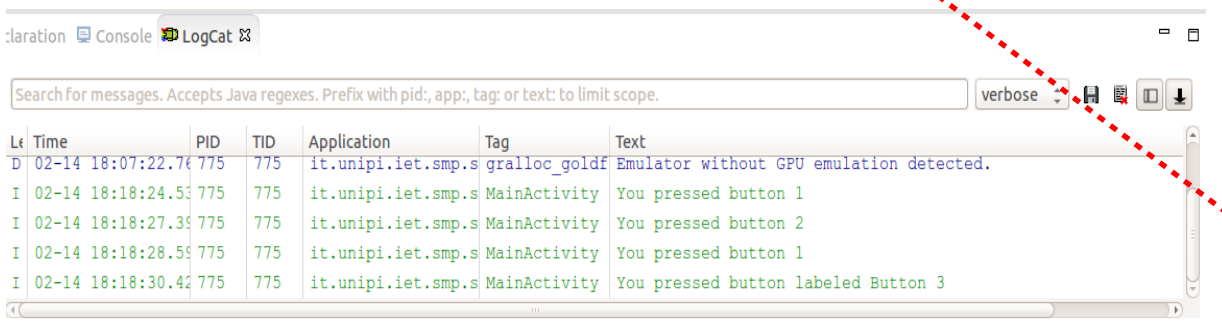
```
public class MainActivity extends Activity implements OnClickListener {

        final static String TAG = "MainActivity";

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
                Button b1 = (Button) findViewById(R.id.button1);
                Button b2 = (Button) findViewById(R.id.button2);
                b1.setOnClickListener(this);
                b2.setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
                if(v.getId() == R.id.button1)
                        Log.i(TAG, "You pressed button 1");
                else if(v.getId() == R.id.button2)
                        Log.i(TAG, "You pressed button 2");
        }

        public void myMethod(View v) {
                CharSequence l = ((Button) v).getText();
                Log.i(TAG, "You pressed button labeled " + l);
        }
}
```



Button

The method must

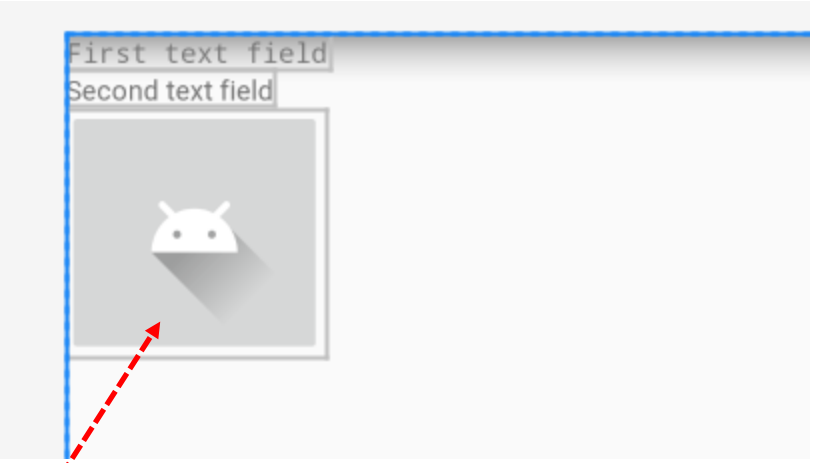
- be public
- return void
- accept a single *View* parameter




```
...
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:text="Button 1" />
  <Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/button1"
    android:layout_below="@+id/button1"
    android:text="Button 2" />
  <Button
    android:id="@+id/button3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/button2"
    android:layout_below="@+id/button2"
    android:text="Button 3"
    android:onClick="myMethod"/>
...
```

Images

- *ImageView*: shows an image
- *ImageButton*: a button with an image



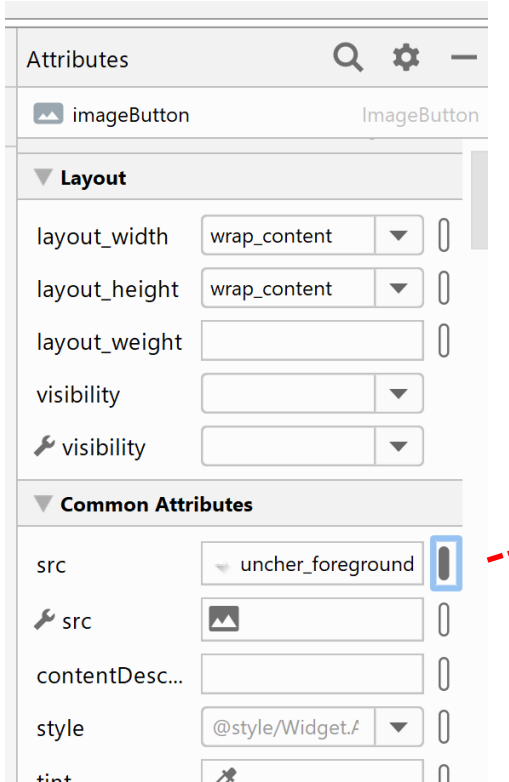

```
<ImageButton
    android:id="@+id/imageButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_launcher_foreground"
/>
```

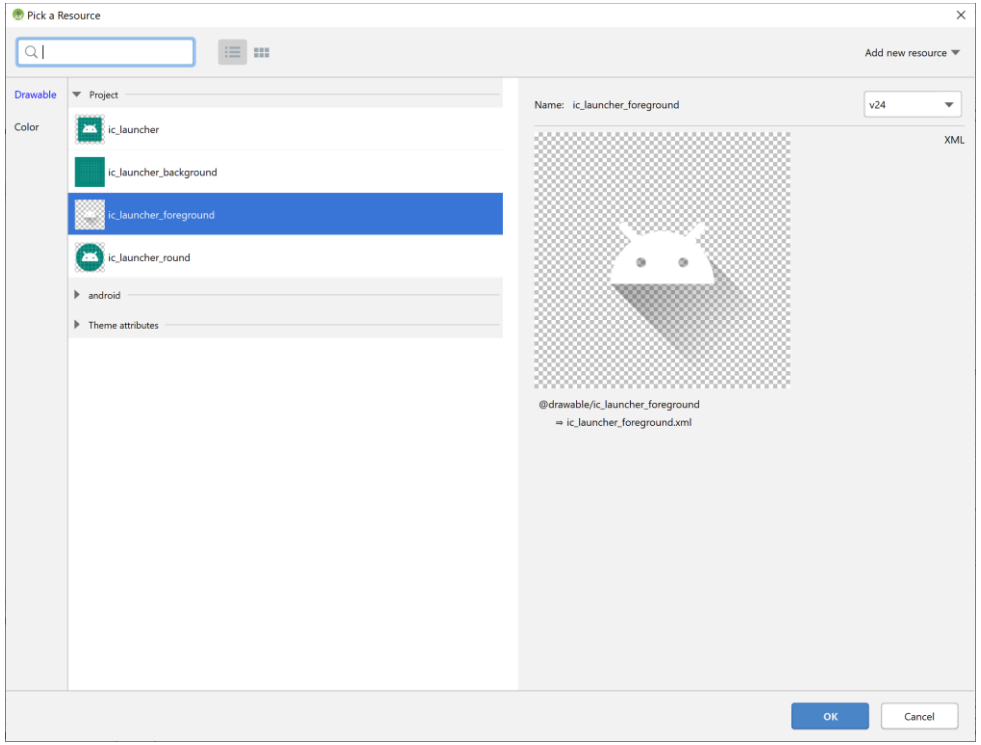
Attributes to scale, center, crop, etc the image

android:src attribute is used to specify the image in drawable folder (e.g. *@drawable/icon*)

Images

- The image resource can be specified by
  - Writing XML code (AndroidStudio suggests as you type)
  - Using the resource selection mechanism of the design editor

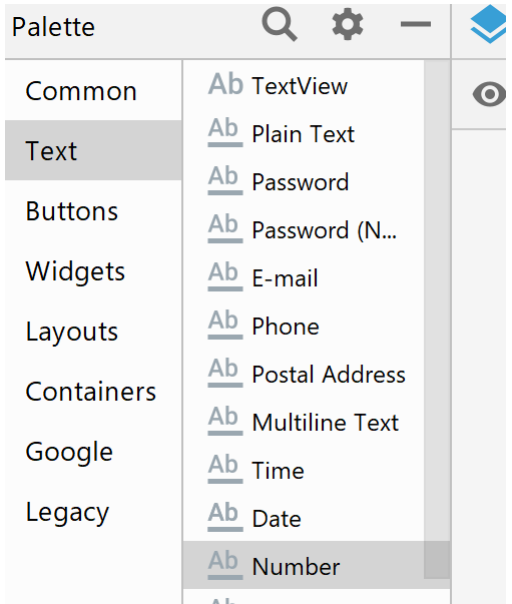




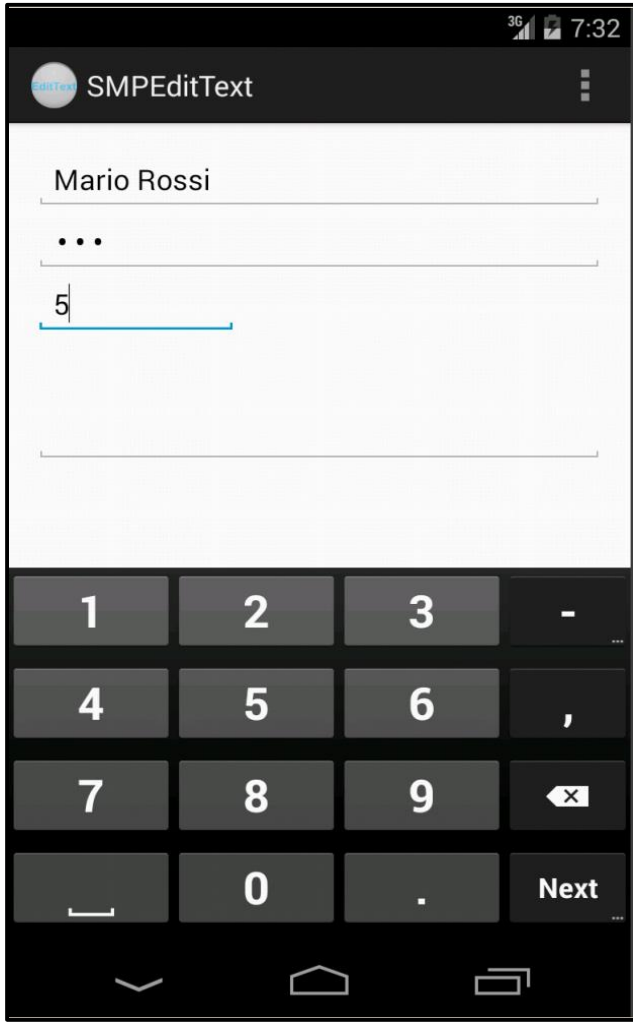
EditText

- Text box for user input
- *android:inputType:* type of keyboard, the set of allowed characters, autocorrection
  - Number, date, password, …

Chosen by using palette

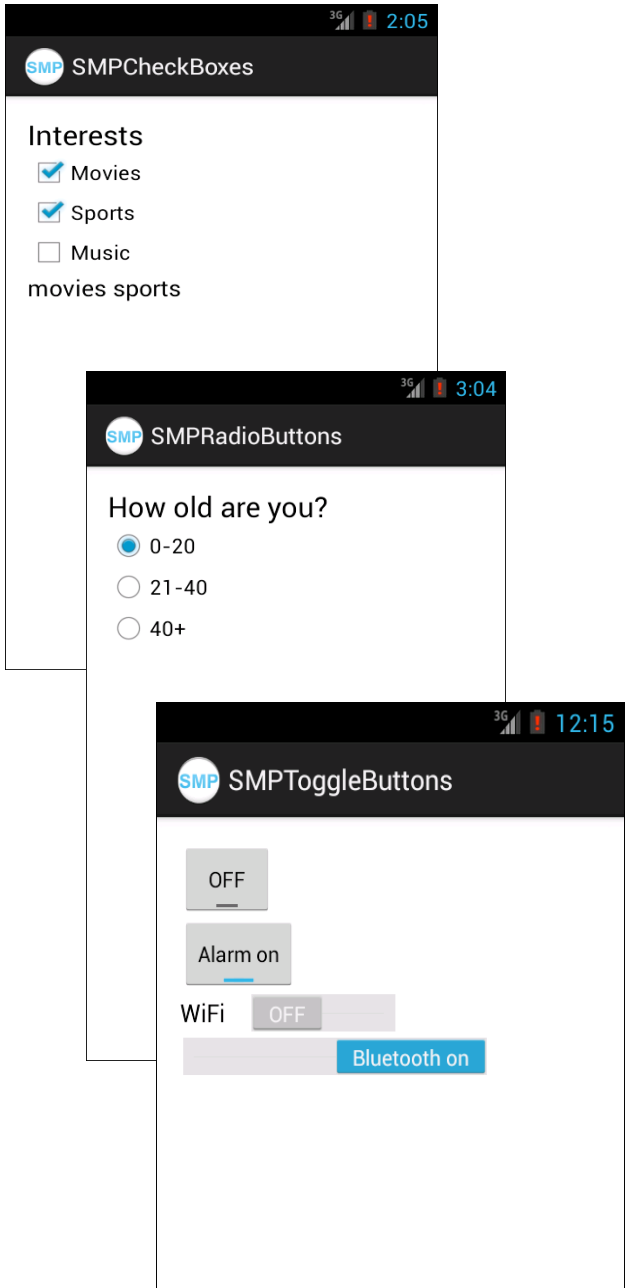


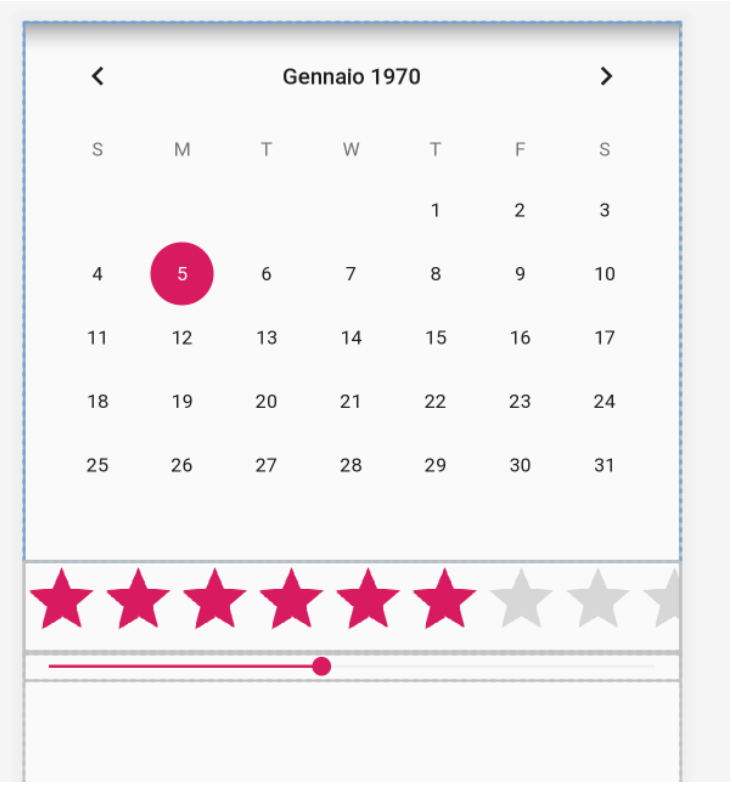

```
…
        <EditText
  android:id="@+id/nameField"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:inputType="textCapWords" >
  <requestFocus />
</EditText>
<EditText
  android:id="@+id/passwordField"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:inputType="textPassword" />
<EditText
  android:id="@+id/numberField"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:ems="6"
  android:inputType="number" />
<EditText
  android:id="@+id/multiLineField"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:minLines="3"
  android:inputType="textMultiLine"
        />
```



Other widgets

- CheckBox
- RadioButton
- ToggleButton
- Switch
- DateView
- RatingBar
- ProgressBar





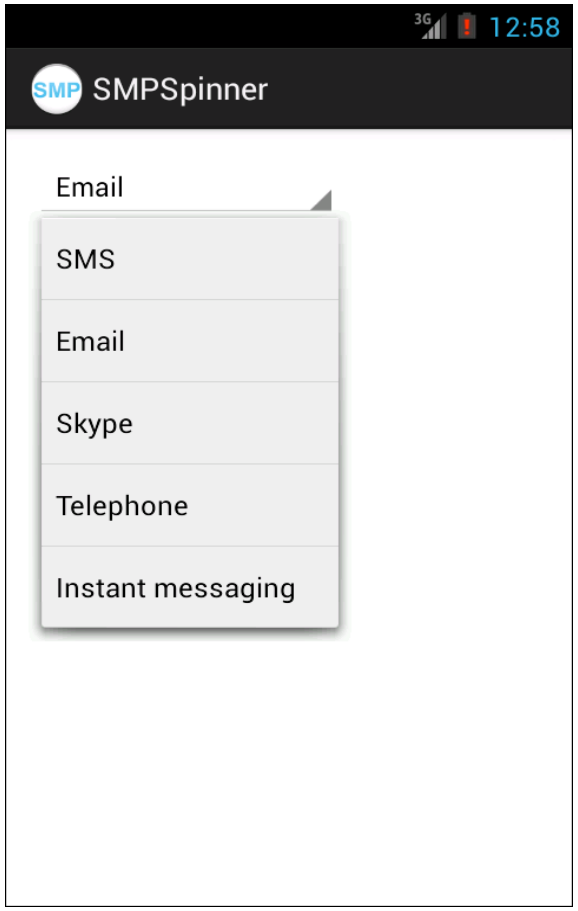
Spinners

- Select one value from a set
- List of values provided as an XML resource

```
...
<Spinner
    android:id="@+id/spinner1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true" />
...
```

res/values/preferred_method.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string-array name="methods_array">
    <item >SMS</item>
    <item >Email</item>
    <item >Skype</item>
    <item >Telephone</item>
    <item >Instant messaging</item>
  </string-array>
</resources>
```

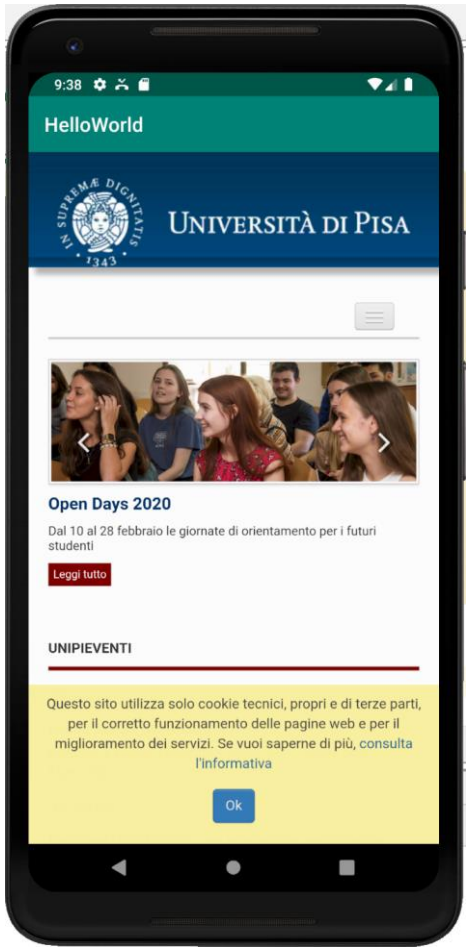


WebView

- Shows webpage

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        WebView myWebView = (WebView) findViewById(R.id.my_wv);
        myWebView.loadUrl("https://www.unipi.it");
    }
}
```

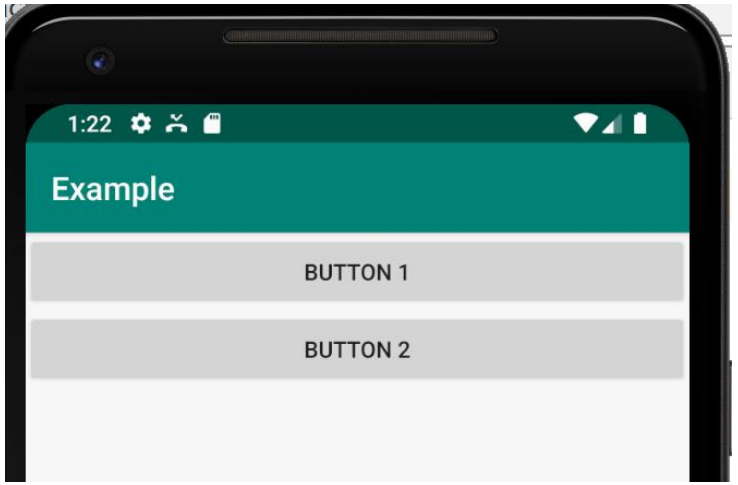
```
<WebView
    android:id="@+id/my_wv"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```



Layouts

- Containers of widgets, define their position onto the screen
- Can be nested to create complex UIs
- Layouts are XML files stored in *res/layout*
- Android provides many layouts
  - *LinearLayout*
  - *ConstraintLayout*
  - *TableLayout*
  - *FrameLayout*
  - *TabLayout*
  - *AppBarLayout*
  - …

LinearLayout

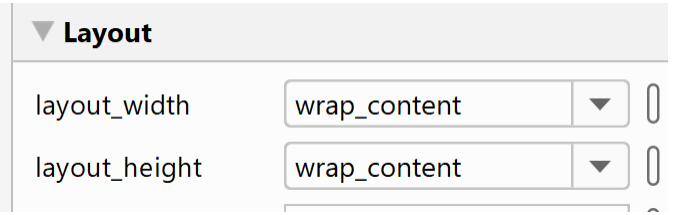


- Child elements (e.g. buttons, text fields, images, etc.) aligned in one direction
- One of the attributes determines orientation *vertical/horizontal*

```
<LinearLayout 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"
    android:orientation="vertical">

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button 2" />
</LinearLayout>
```

Width and height

- **wrap_content**: widget/layout as wide/high as its content (e.g. text)
- **match_parent**: widget/layout as wide/high as its parent layout box
- Can be applied to almost all widgets and layouts
- Manually or by using AndroidStudio's attribute editors



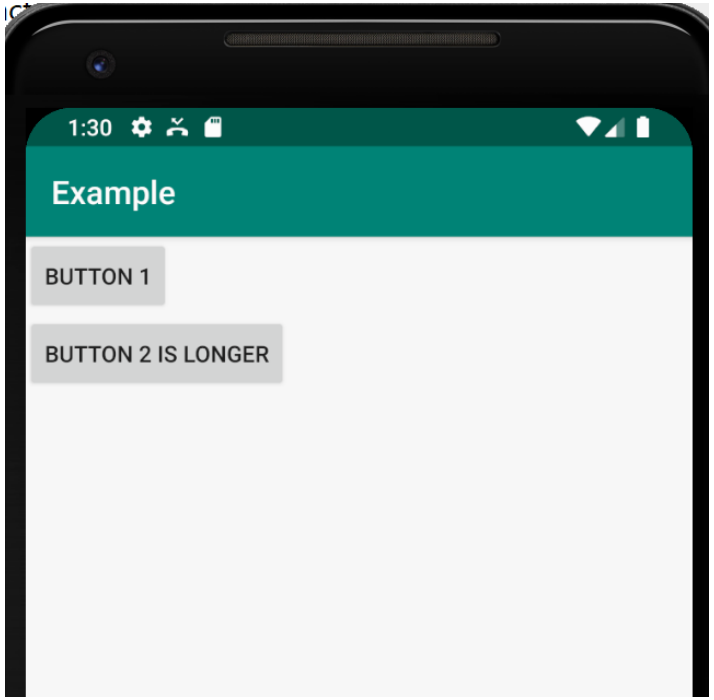

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"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

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2 is longer" />
</LinearLayout>
```

Layout is as wide as its parent (the screen)

Layout is as high as its parent (the screen)

Button is as wide as its content (the text)

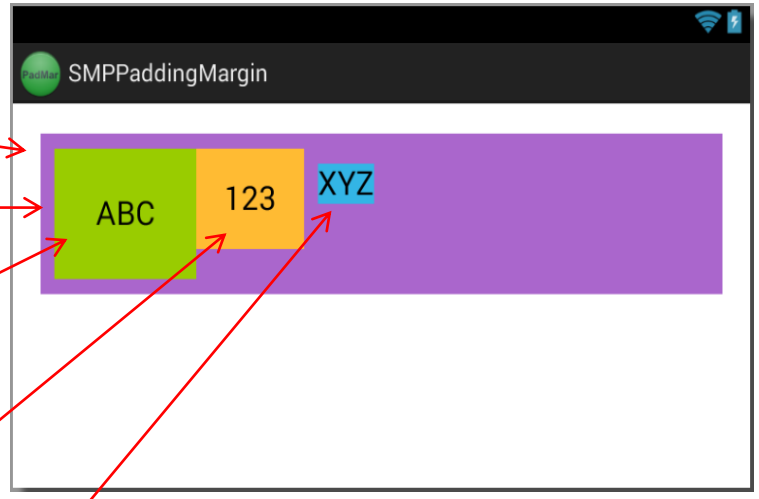
Button is as high as its content (the text)

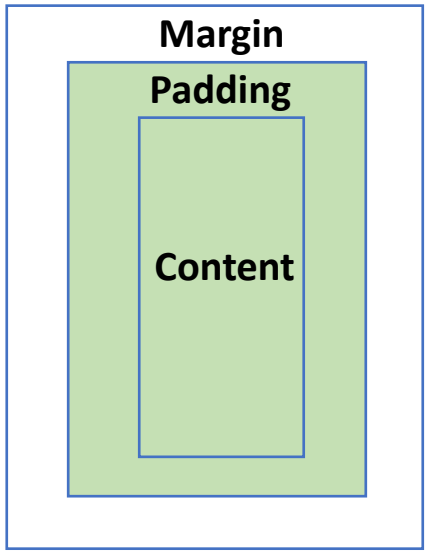


Padding, margin

- **Padding**: space inside the view's border
- **Margin**: space outside the view's border

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:layout_margin="20dp"
  android:background="@android:color/holo_purple"
  android:orientation="horizontal"
  android:padding="10dp"
  android:baselineAligned="false"
  >
  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@android:color/holo_green_light"
    android:padding="30dp"
    android:text="ABC"
    android:textAppearance="?android:attr/textAppearanceLarge" />
  <TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@android:color/holo_orange_light"
    android:padding="20dp"
    android:text="123"
    android:textAppearance="?android:attr/textAppearanceLarge" />
  <TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@android:color/holo_blue_light"
    android:layout_margin="10dp"
    android:text="XYZ"
    android:textAppearance="?android:attr/textAppearanceLarge" />
</LinearLayout>
```

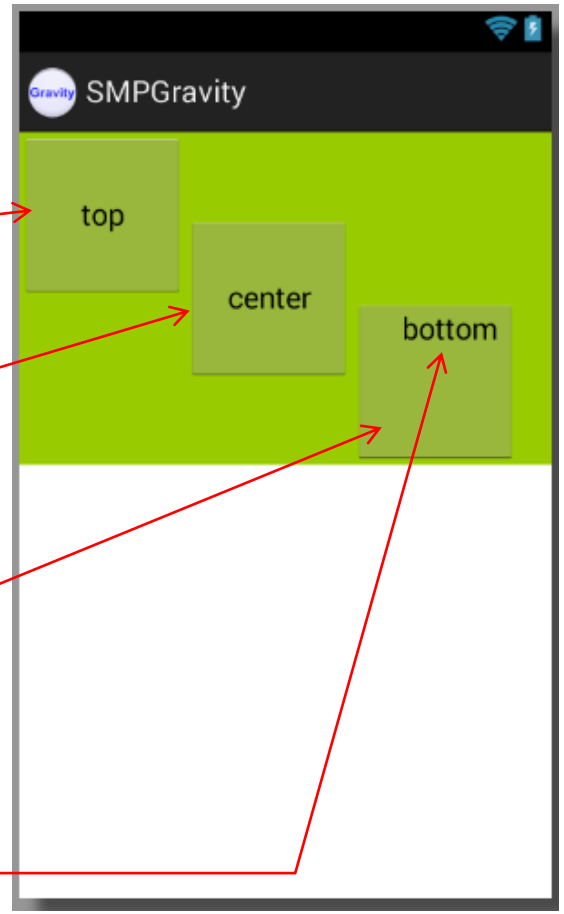




Gravity

- ***gravity:*** how content is aligned within a view
- ***layout_gravity:*** how a view is aligned within a container

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="200dp"
  android:background="@android:color/holo_green_light"
  android:baselineAligned="false"
  android:gravity="left"
  android:orientation="horizontal"
  >
  <Button
    android:id="@+id/Button01"
    android:layout_width="100dp"
    andr        oid:layout_height="100dp"
    android:layout_gravity="top"
    android:text="top" >
  </Button>
  <Button
    android:id="@+id/Button02"
    android:layout_width="100dp"
    android:layout_height="100dp"
    android:layout_gravity="center"
    android:text="center" >
  </Button>
  <Button
    android:id="@+id/Button03"
    android:layout_width="100dp"
    android:layout_height="100dp"
    android:layout_gravity="bottom"
    android:gravity="top|right"
    android:text="bottom" >
  </Button>
</LinearLayout>
```

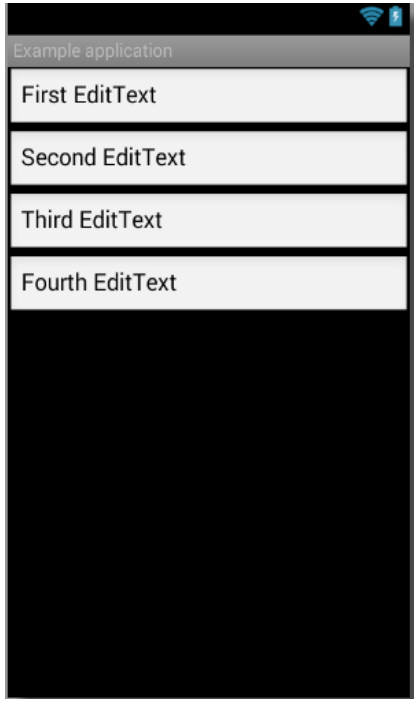


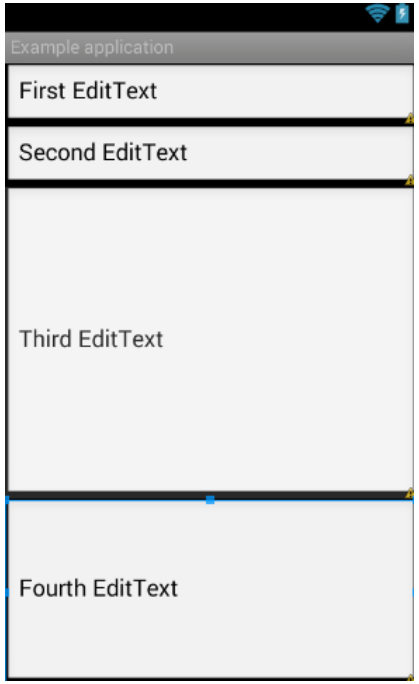
LinearLayout

- A *weight* can be assigned to contained elements
  - default weight is 0
  - remaining space is assigned to children in the proportion of their weight

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">
  <EditText
    android:id="@+id/editText1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="First EditText" />
  <EditText
    android:id="@+id/editText2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="0"
    android:text="Second EditText" />
  <EditText
    android:id="@+id/editText3"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="2"
    android:text="Third EditText" />
  <EditText
    android:id="@+id/editText4"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Fourth EditText" />
</LinearLayout>
```

Same XML code but without weights

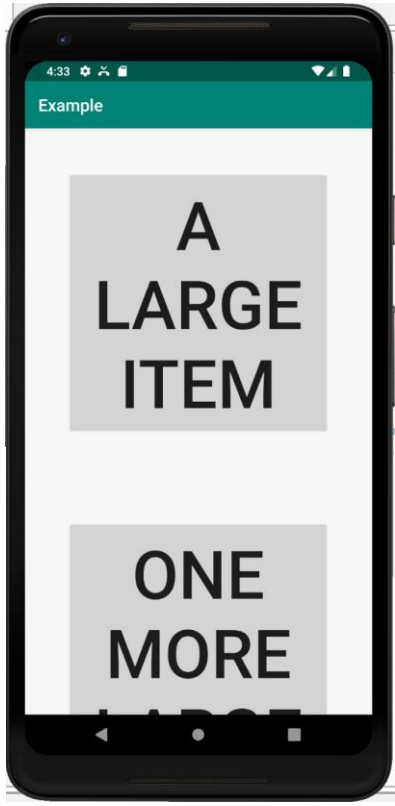




Scrolling

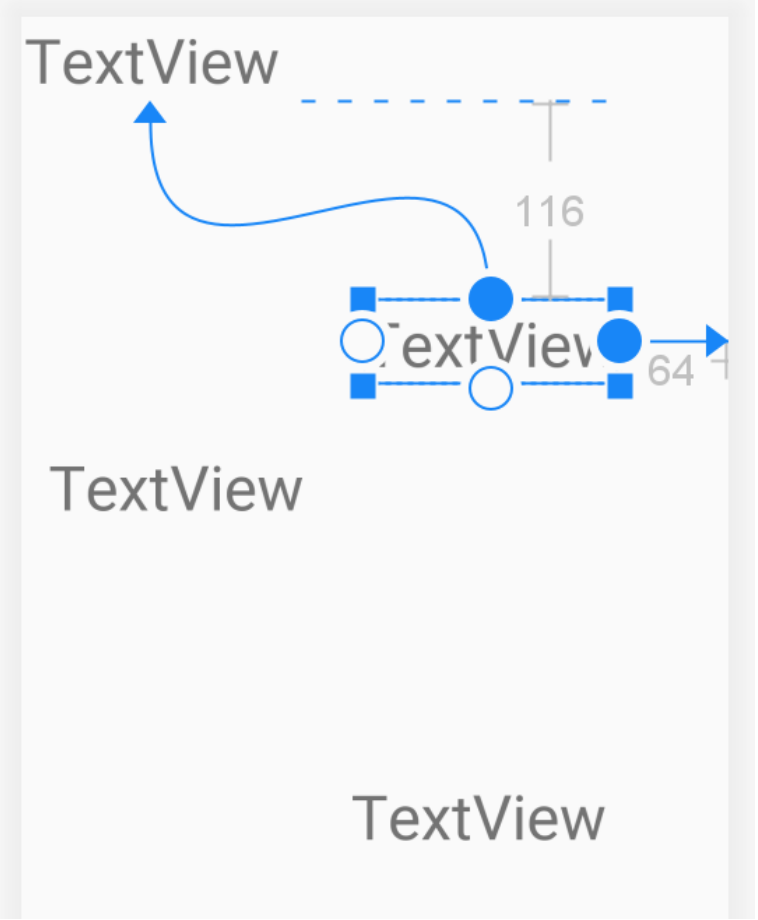
- Phone screen is small… scrolling!
- Views for Scrolling:
  - *ScrollView* for vertical scrolling
  - *HorizontalScrollView*
- Rules:
  - Only one direct child View
  - Child could have many children of its own

```
<ScrollView
  . . .>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >

        <Button
            android:id="@+id/button3"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_margin="50dp"
            android:text="A large item"
            android:textSize="80sp" />

        <Button
            android:id="@+id/button4"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_margin="50dp"
            android:text="One more large item"
            android:textSize="80sp" />
    </LinearLayout>
</ScrollView>
```



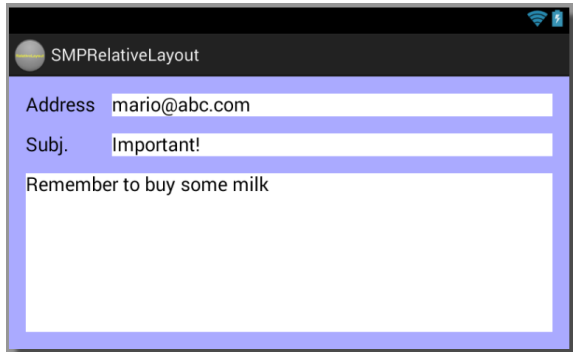
ConstraintLayout

- The position of elements is defined relative to
  - The container's border
  - Other contained elements
- Reduces the need for nested layouts and keeps the hierarchy flat (faster do draw)
- *RelativeLayout* is similar



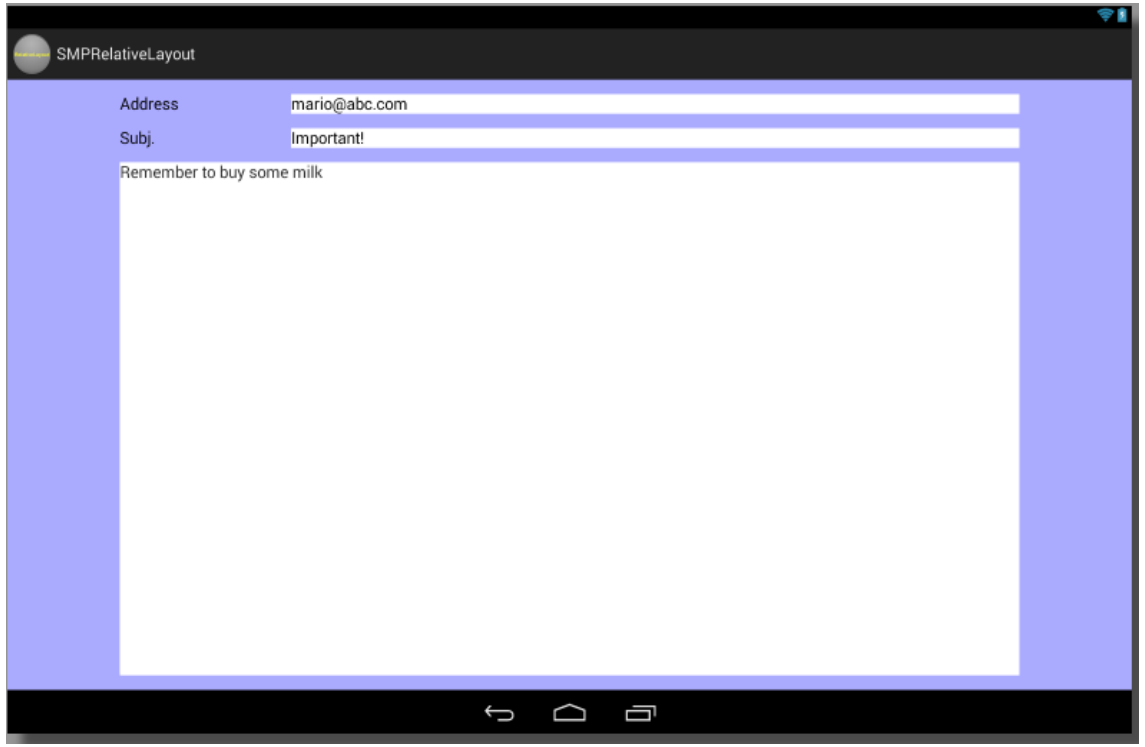
RelativeLayout

- If properly set, constraints help to adapt the UI to the different screen sizes



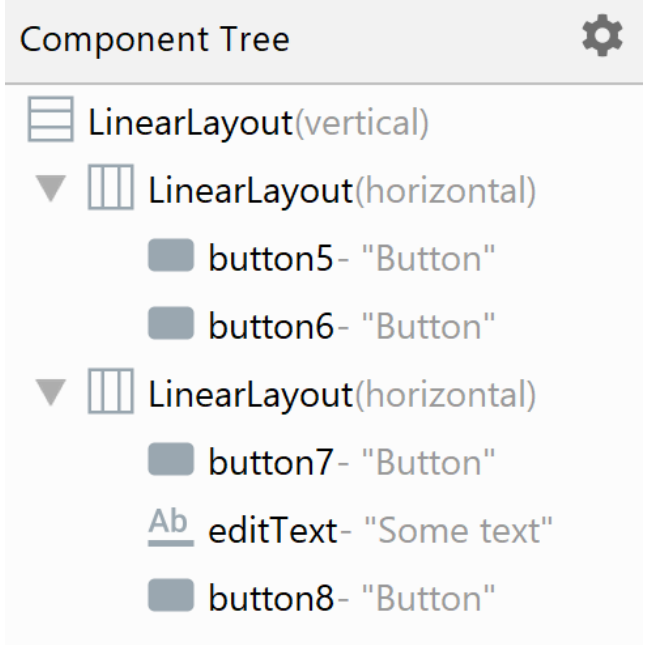
7.3” tablet

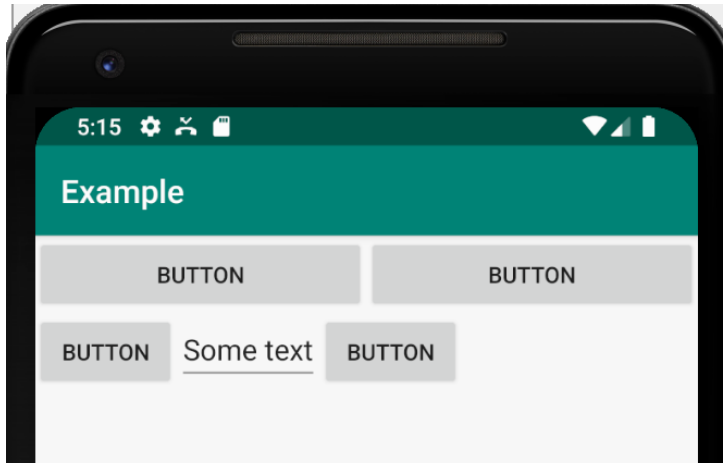
10.1” tablet



Nesting layouts

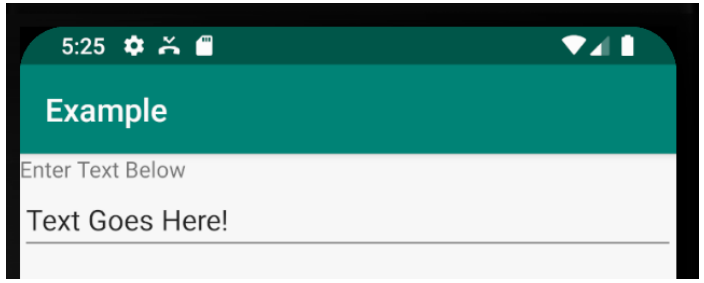
- Layouts can be nested to define more complex UIs





UIs with Java code

- Attributes and content can also be defined programmatically




```
public class MainActivity extends AppCompatActivity {

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

        LinearLayout ll = new LinearLayout(this);
        ll.setOrientation(LinearLayout.VERTICAL);
        TextView myTextView = new TextView(this);
        EditText myEditText = new EditText(this);
        myTextView.setText("Enter Text Below");
        myEditText.setText("Text Goes Here!");
        int lHeight = LinearLayout.LayoutParams.MATCH_PARENT;
        int lWidth = LinearLayout.LayoutParams.WRAP_CONTENT;
        ll.addView(myTextView, new LinearLayout.LayoutParams(lHeight, lWidth));
        ll.addView(myEditText, new LinearLayout.LayoutParams(lHeight, lWidth));
        setContentView(ll);
    }
}
```

Docs

- The set of attributes can be very large, use the docs

https://developer.android.com/reference/android/widget/TextView

XML attributes	
`android:allowUndo`	Whether undo should be allowed for editable text.
`android:autoLink`	Controls whether links such as urls and email addresses are automatically found and converted to clickable links.
`android:autoSizeMaxTextSize`	The maximum text size constraint to be used when auto-sizing text.
`android:autoSizeMinTextSize`	The minimum text size constraint to be used when auto-sizing text.
`android:autoSizePresetSizes`	Resource array of dimensions to be used in conjunction with **`autoSizeTextType`** set to **`uniform`**.
`android:autoSizeStepGranularity`	Specify the auto-size step size if **`autoSizeTextType`** is set to **`uniform`**.
`android:autoSizeTextType`	Specify the type of auto-size.
`android:autoText`	If set, specifies that this TextView has a textual input method and

References

- CS 528 Mobile and Ubiquitous Computing, WPI
- http://developer.android.com
- http://developer.android.com/reference