Android Development

Chapter 3 – Getting to know the Android User Interface

katholieke hogeschool
associatie KU Leuven
vives

Android Development
Chapter 3 – Getting to know the Android User Interface

The User Interface

katholieke hogeschool
associatie KU Leuven
vives

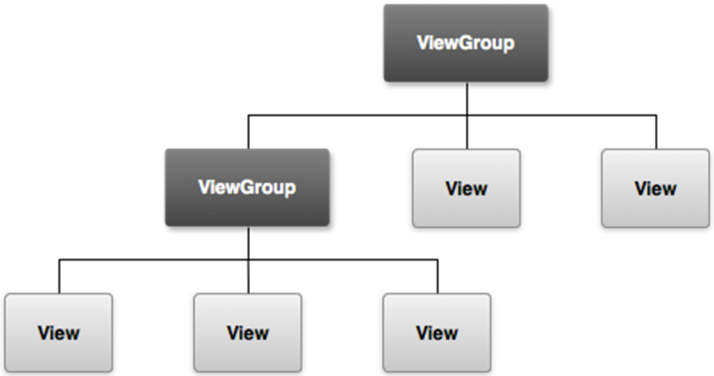
The User Interface

- All user interface elements in an Android app are built using View and ViewGroup objects.
  - A View is an object that draws something on the screen that the user can interact with.
  - A ViewGroup is an object that holds other View (and ViewGroup) objects in order to define the layout of the interface.

- Android provides a collection of both View and ViewGroup subclasses that offer you common input controls (such as buttons and text fields) and various layout models (such as a linear or relative layout).

User Interface Layout

- The user interface for each component of your app is defined using a hierarchy of View and ViewGroup objects.



- Each view group is an invisible container that organizes child views
  - The child views may be input controls or other widgets that draw parts of the UI
- This hierarchy tree can be as simple or complex as you need it to be
  - Simplicity is best for performance
- When you load a layout resource in your app, Android initializes each node of the layout into a runtime object
  - Can be used to define additional behaviors, query the object state or modify the layout

Android Development
Chapter 3 – Getting to know the Android User Interface

Layouts

katholieke hogeschool
associatie KU Leuven
vives

Layouts

- A layout defines the visual structure for a user interface, such as the UI for an activity or app widget.

- You can declare a layout in two ways:
  - Declare UI elements in XML
    - Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
  - Instantiate layout elements at runtime.
    - Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

- The Android framework gives you the flexibility to use either or both of these methods for declaring and managing your application's UI.

XML

- Each layout file must contain exactly one root element, which must be a View or ViewGroup object.
- Once you've defined the root element, you can add additional layout objects or widgets as child elements to gradually build a View hierarchy that defines your layout.

```
<?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" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Instantiate layout elements at runtime

- Create View dynamically
  - You should probable save it as an attribute of the class
- Set properties of the view
- Set layout parameters
- Add it to the necessary container

Absolute and Relative Measurements

- In general, specifying a layout width and height using absolute units such as pixels is not recommended.
- Instead, using relative measurements such as density-independent pixel units (dp), wrap_content, or match_parent, is a better approach, because it helps ensure that your application will display properly across a variety of device screen sizes.

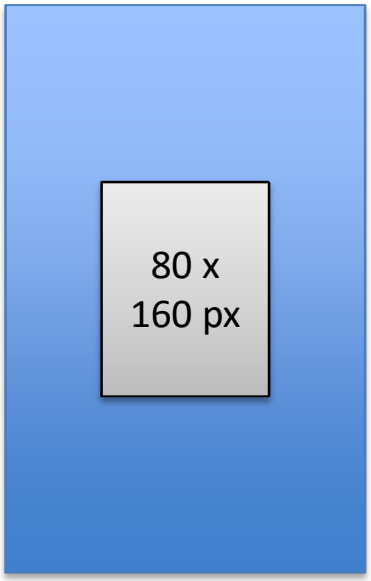
- Absolute measurements
  - px (Pixels)
    - Corresponds to actual pixels on the screen.
  - in (Inches)
    - Based on the physical size of the screen.
  - mm (Millimeters)
    - Based on the physical size of the screen.
  - pt (Points)
    - 1/72 of an inch based on the physical size of the screen.

Absolute and Relative Measurements

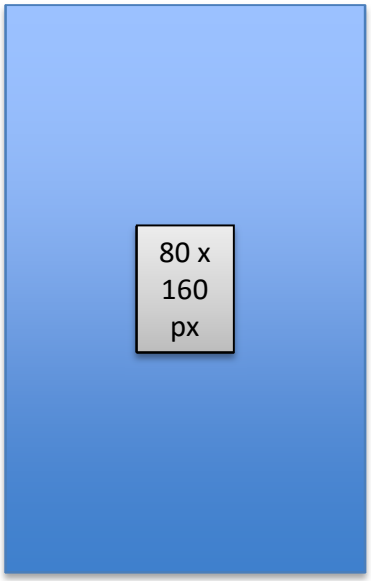
- Relative measurements
  - dp (Density-independent Pixels)
    - An abstract unit that is based on the physical density of the screen. These units are relative to a 160 dpi screen, so one dp is one pixel on a 160 dpi screen.
    - The ratio of dp-to-pixel will change with the screen density, but not necessarily in direct proportion.
    - Note: The compiler accepts both "dip" and "dp", though "dp" is more consistent with "sp".
  - sp (Scale-independent Pixels)
    - This is like the dp unit, but it is also scaled by the user's font size preference.
    - It is recommend you use this unit when specifying font sizes, so they will be adjusted for both the screen density and user's preference.

Density Support

- Example
  - Using px for dimensions (bad practice)



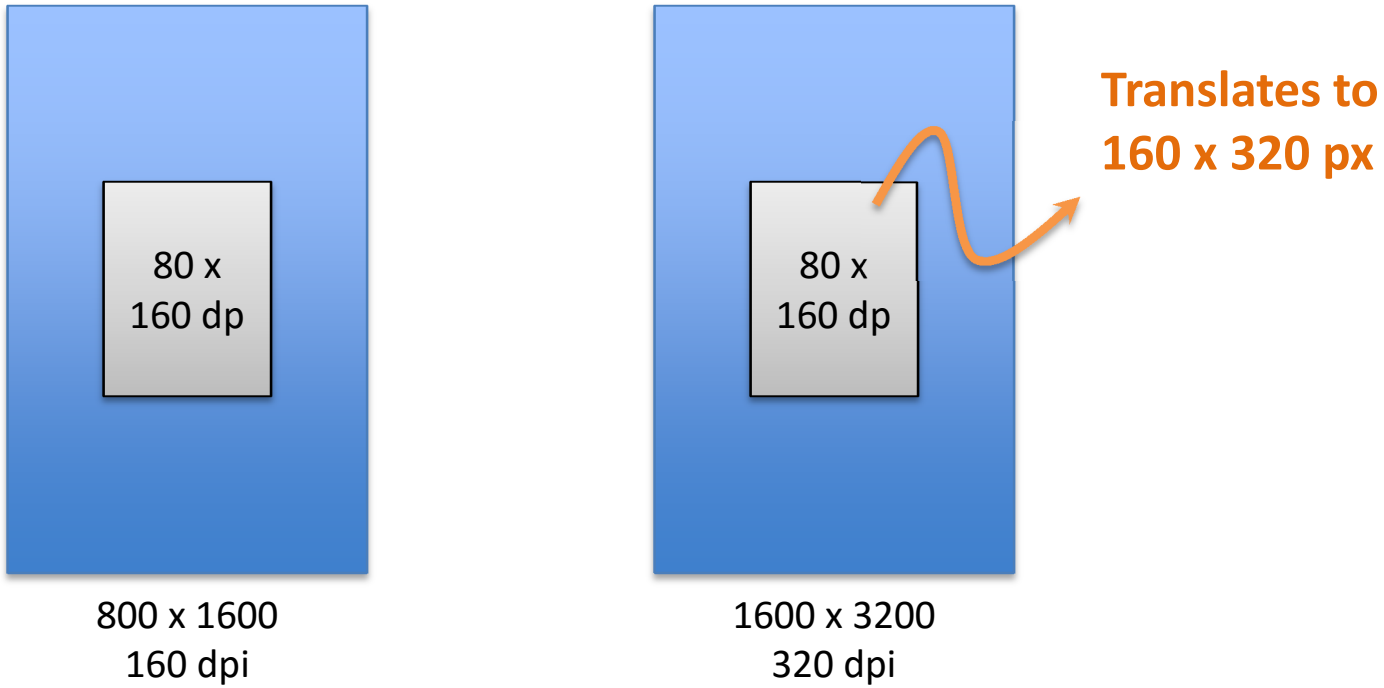
800 x 1600
160 dpi



1600 x 3200
320 dpi

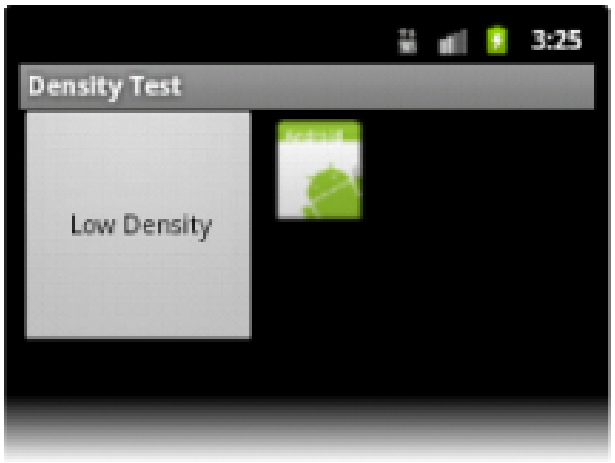
Density Support

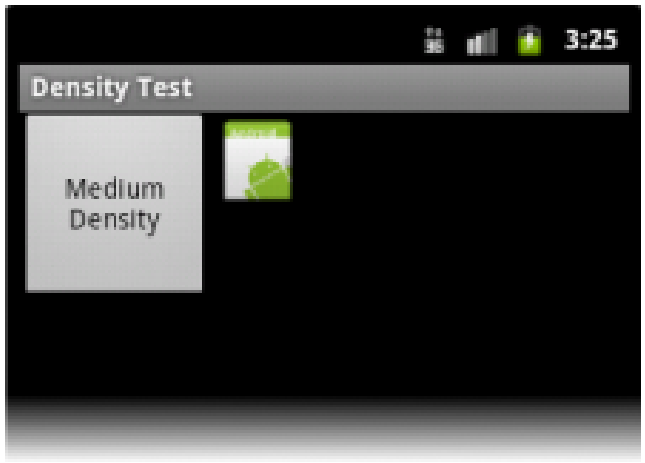
- Example
  - Using dp for dimensions (good practice)

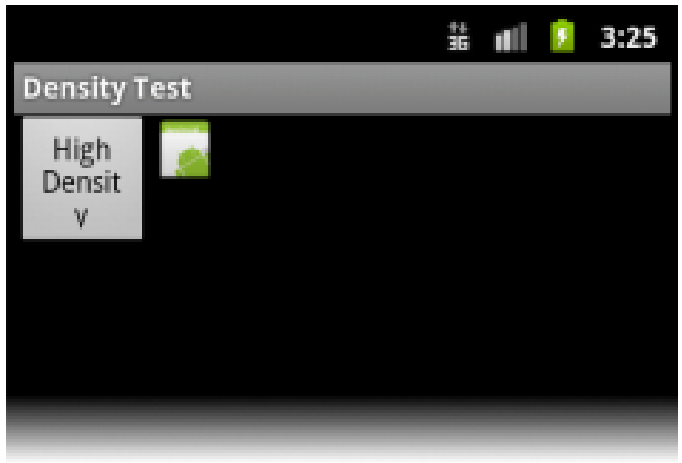


Density Support

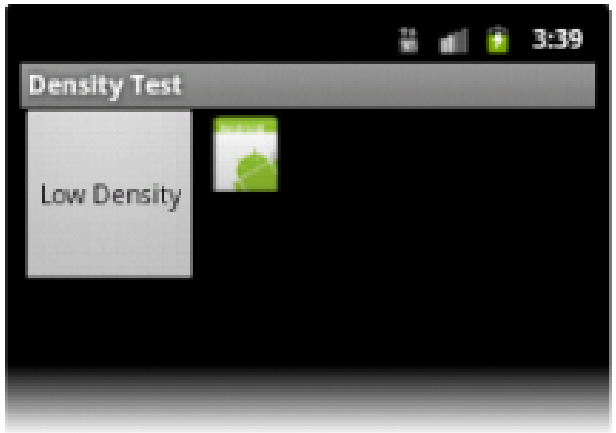
- **Bad:**

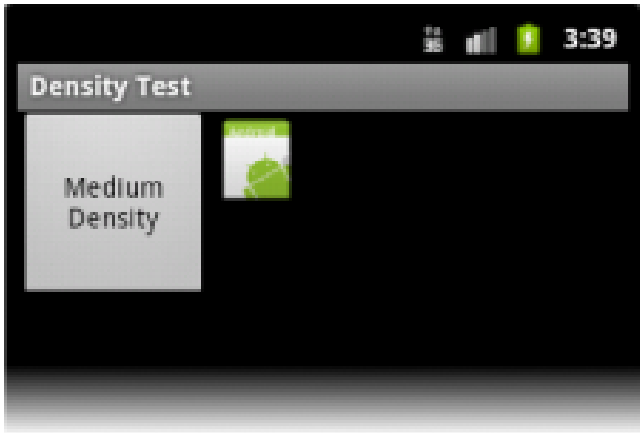


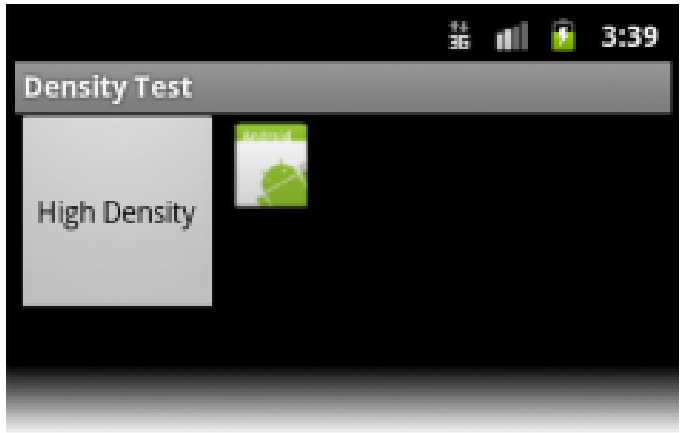




- **Good:**





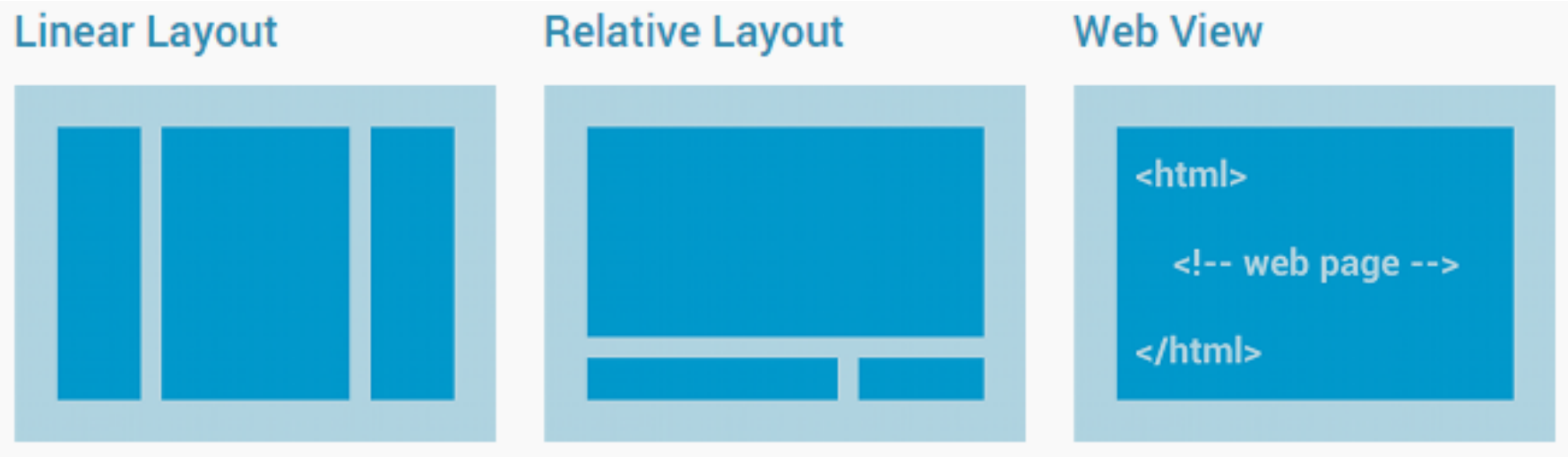


Best Practices

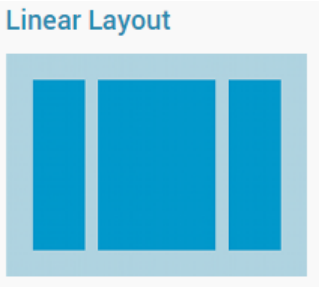
- Use wrap_content, fill_parent, or the dp unit for layout dimensions
- Do not use hard-coded pixel values in your application code
- Do not use AbsoluteLayout
  - Deprecated
- Use size and density-specific resources
  - res/drawable-mdpi/icon.png //for medium-density screens
  - res/drawable-hdpi/icon.png //for high-density screens

Some Common Layouts

- Although you can nest one or more layouts within another layout to achieve your UI design, you should strive to keep your layout hierarchy as shallow as possible.
  - Your layout draws faster if it has fewer nested layouts (a wide view hierarchy is better than a deep view hierarchy).



Linear Layout



- A layout that organizes its children into a single horizontal or vertical row.
  - It creates a scrollbar if the length of the window exceeds the length of the screen.
  - You can specify the layout direction with the android:orientation attribute.
  - All children of a LinearLayout are stacked one after the other
    - so a vertical list will only have one child per row, no matter how wide they are
    - a horizontal list will only be one row high (the height of the tallest child, plus padding).
  - A LinearLayout respects margins between children and the gravity (right, center, or left alignment) of each child.

The Weight Attribute

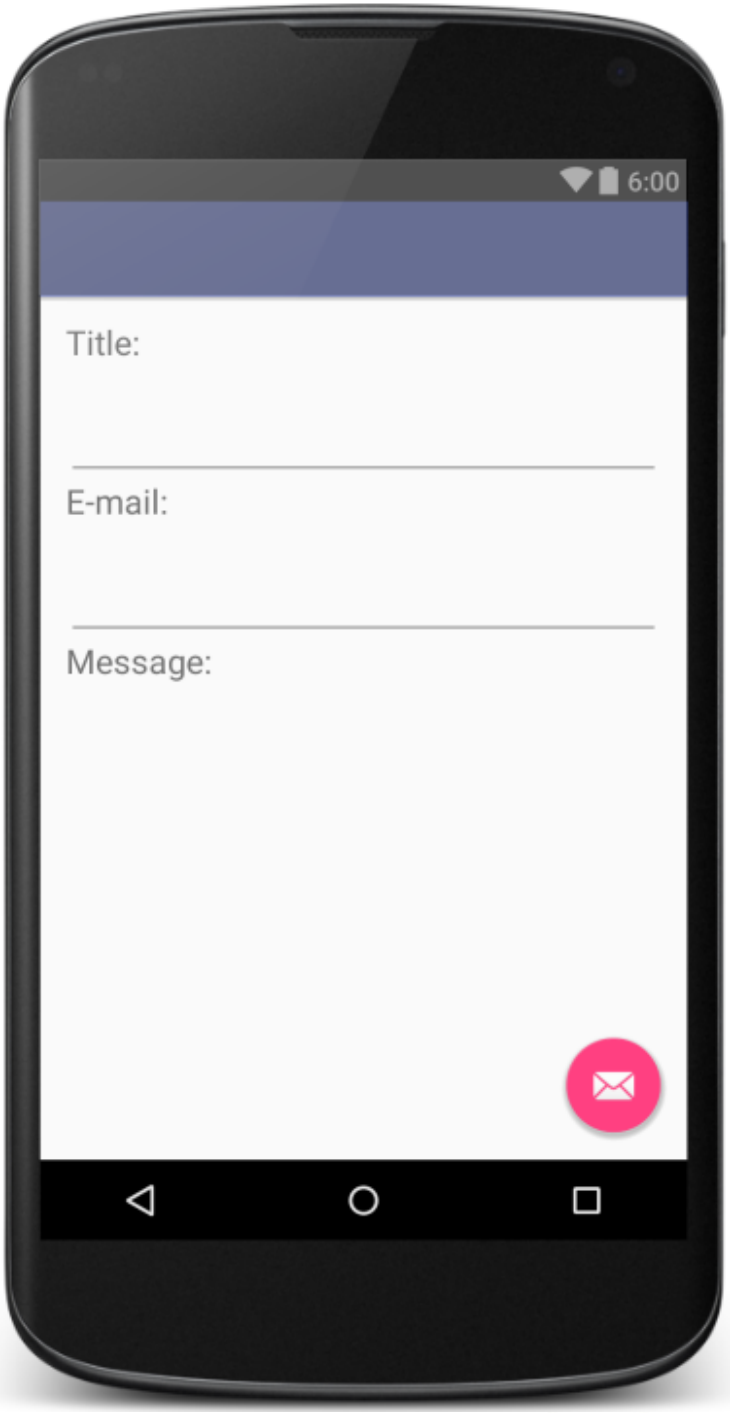
- LinearLayout also supports assigning a weight to individual children with the android:layout_weight attribute.
  - This attribute assigns an "importance" value to a view in terms of how much space it should occupy on the screen.
  - A larger weight value allows it to expand to fill any remaining space in the parent
  - Child views can specify a weight value, and then any remaining space in the view group is assigned to children in the proportion of their declared weight.
  - Default weight is zero.

- Example:
  - You have a MapView and a Table which should show some additional information to the map. The map should use 3/4 of the screen and table should use 1/4 of the screen. Then you will set the layout_weight of the map to 3 and the layout_weight of the table to 1.

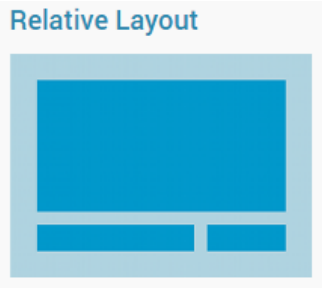
- To get it work you also have to set the height or width (depending on your orientation) to 0px.

Example

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Title:"
        android:textSize="20sp"
        android:paddingEnd="8dp"/>

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="E-mail:"
        android:textSize="20sp"
        android:paddingEnd="8dp" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Message:"
        android:textSize="20sp"
        android:paddingEnd="8dp" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textMultiLine"
        android:ems="10"
        android:layout_weight="10" />
</LinearLayout>
```



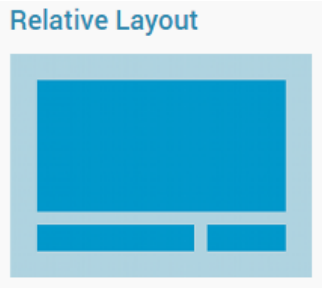
Relative Layout



- Enables you to specify the location of child objects relative to each other (specified by ID) or to the parent (aligned to the top of the parent).

- A RelativeLayout is a very powerful utility for designing a user interface because it can eliminate nested view groups and keep your layout hierarchy flat, which improves performance.
  - If you find yourself using several nested LinearLayout groups, you may be able to replace them with a single RelativeLayout.

Relative Layout

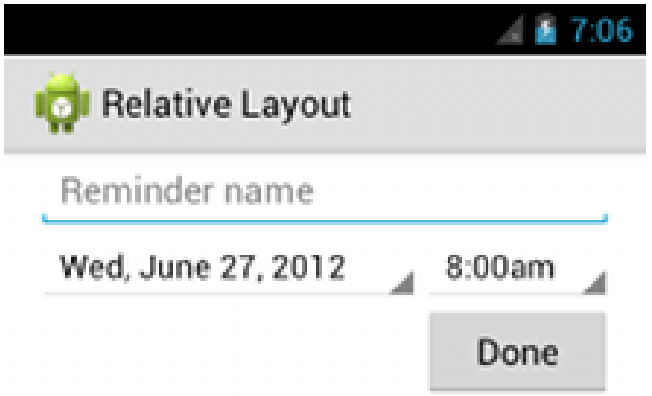


- Some of the many layout properties available to views in a RelativeLayout include:
  - android:layout_alignParentTop
    - If "true", makes the top edge of this view match the top edge of the parent.
  - android:layout_centerVertical
    - If "true", centers this child vertically within its parent.
  - 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.

- The value for each layout property is either a boolean to enable a layout position relative to the parent RelativeLayout or an ID that references another view in the layout against which the view should be positioned.

Example

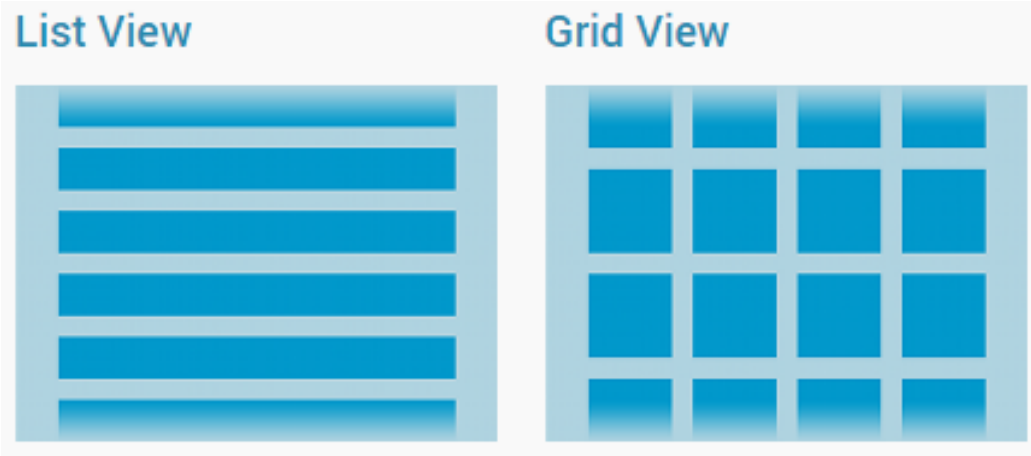
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```



Most of this can be achieved relatively easily by using the design editor

Building Layouts with an Adapter

- When the content for your layout is dynamic or not pre-determined, you can use a layout that subclasses AdapterView to populate the layout with views at runtime.
- A subclass of the AdapterView class uses an Adapter to bind data to its layout.
- The Adapter behaves as a middleman between the data source and the AdapterView layout
  - The Adapter retrieves the data (from a source such as an array or a database query) and converts each entry into a view that can be added into the AdapterView layout



Android Development
Chapter 3 – Getting to know the Android User Interface

Adapting to Display Orientation

katholieke hogeschool
associatie KU Leuven
vives

Display Orientation

- One of the key features of modern smartphones is their ability to switch screen orientation, and Android is no exception.
- Android supports two screen orientations: portrait and landscape
- By default, when you change the display orientation of your Android device, the current activity that is displayed automatically redraws its content in the new orientation.
  - This is because the onCreate() method of the activity is fired whenever there is a change in display orientation.
  - When you change the orientation of your Android device, your current activity is actually destroyed and then recreated.

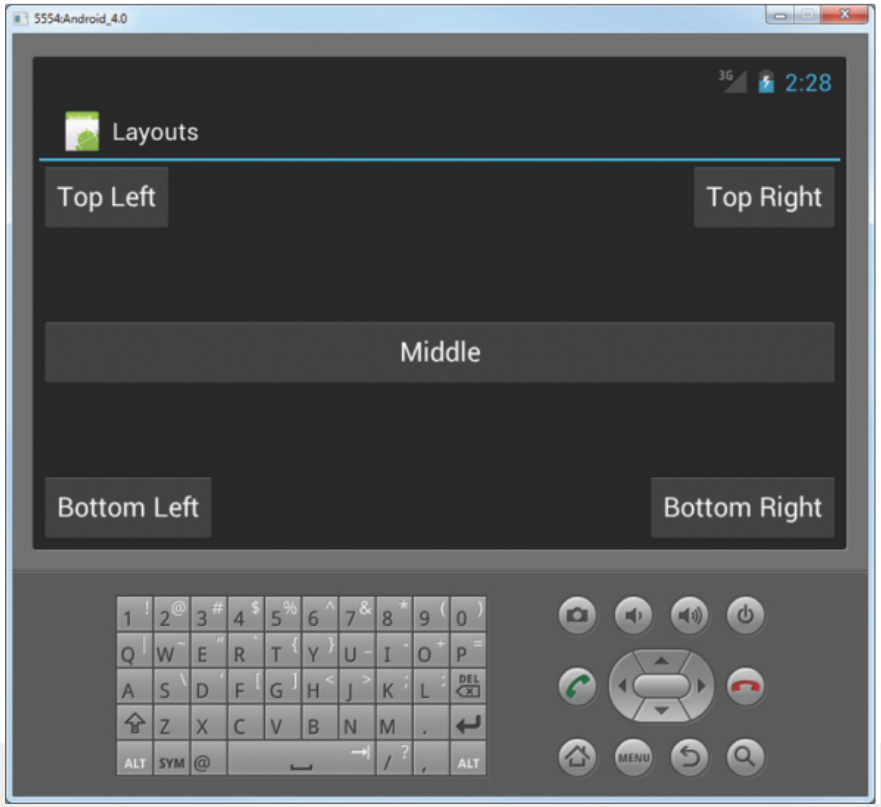
- You can change the emulator orientation by pressing CTRL-F11

Adapting to Display Orientation

- In general, you can employ two techniques to handle changes in screen orientation:
  - Anchoring
    - The easiest way is to anchor your views to the four edges of the screen. When the screen orientation changes, the views can anchor neatly to the edges.
    - Anchoring can be easily achieved by using RelativeLayout .
  - Resizing and repositioning
    - Whereas anchoring and centralizing are simple techniques to ensure that views can handle changes in screen orientation, the ultimate technique is resizing each and every view according to the current screen orientation.

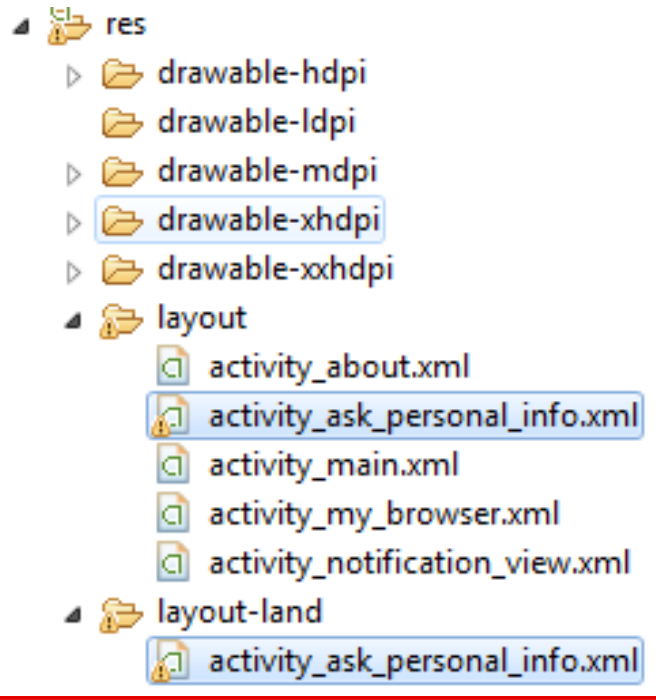
Anchoring Views

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    tools:context="${relativePackage}.${activityClass}" >
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Right"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true" />
    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Right"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button5"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Middle"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```



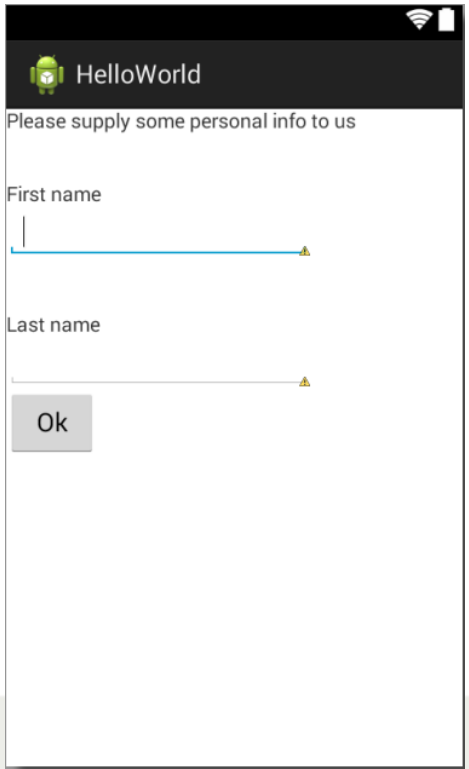
Resizing and Repositioning

- Apart from anchoring your views to the four edges of the screen, an easier way to customize the UI based on screen orientation is to create a separate res/layout folder containing the XML files for the UI of each orientation.
- To support landscape mode, you can create a new folder in the res folder and name it as layout-land (representing landscape).
- Basically, the xml file contained within the layout folder defines the UI for the activity in portrait mode, whereas the xml file in the layout-land folder defines the UI in landscape mode.



Resizing and Repositioning - Portrait

```
<RelativeLayout 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"
    tools:context="${relativePackage}.${activityClass}" >
    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ask_info" />
    <LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/textView2"
        android:layout_marginTop="56dp"
        android:orientation="vertical" >
      <TextView
          android:id="@+id/txtFirstname"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_marginTop="38dp"
          android:text="@string/firstname" />
      <EditText
          android:id="@+id/txtInputFirstname"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:ems="10" >
          <requestFocus />
      </EditText>
      ...
      <TextView
          android:id="@+id/txtLastname"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_marginTop="38dp"
          android:text="@string/lastname" />
      <EditText
          android:id="@+id/txtInputLastname"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:ems="10" />
      <Button
          android:id="@+id/btnOk"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@string/button_ok"
          android:onClick="onPersonalInfoOk" />
    </LinearLayout>
</RelativeLayout>
```



Resizing and Repositioning - Landscape

```
<RelativeLayout 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"
    tools:context="${relativePackage}.${activityClass}" >
    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ask_info" />
    <LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/textView2"
        android:layout_marginTop="56dp"
        android:orientation="vertical" >
      <LinearLayout
          android:id="@+id/linearLayout2"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:orientation="horizontal" >
          <TextView
              android:id="@+id/txtFirstname"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:layout_marginLeft="12dp"
              android:layout_marginRight="12dp"
              android:text="@string/firstname" />
          <EditText
              android:id="@+id/txtInputFirstname"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:ems="10" >
              <requestFocus />
          </EditText>
      </LinearLayout>
      ...
```

```
      ...
      <LinearLayout
          android:id="@+id/linearLayout3"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:orientation="horizontal" >
          <TextView
              android:id="@+id/txtLastname"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:layout_marginLeft="12dp"
              android:layout_marginRight="12dp"
              android:text="@string/lastname" />
          <EditText
              android:id="@+id/txtInputLastname"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:ems="10" />
      </LinearLayout>
      <Button
          android:id="@+id/btnOk"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_gravity="center"
          android:text="@string/button_ok"
          android:onClick="onPersonalInfoOk" />
    </LinearLayout>
</RelativeLayout>
```