# User Interface (UI)

## What this lecture will teach you

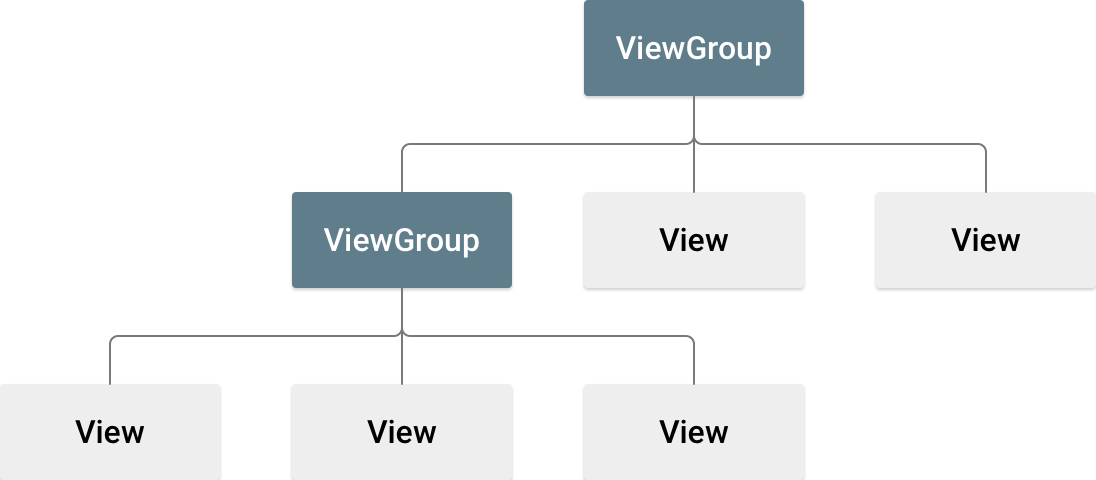
* Understanding how to create a number of common UI layouts
* Using XML and the design tool for specifying layouts
* Styles and themes
* Phone dimensions
* Operations and attributes of views

## Views and View Groups

Every item in a user interface is a subclass of the Android **View** class. The Android SDK provides a set of pre-built views that can be used to construct a user interface. Typical examples include standard items such as the Button, CheckBox, ProgressBar and TextView classes. Such views are also referred to as **widgets**.

For requirements that are not met by the widgets supplied with the SDK, new views may be created either by subclassing and extending an existing class, or creating an entirely new component by building directly on top of the View class.

A view can also be comprised of multiple other views (otherwise known as a composite view). Such views are subclassed from the Android ViewGroup class (android.view.ViewGroup) which is itself a subclass of View. An example of such a view is the RadioGroup, which is intended to contain multiple RadioButton objects such that only one can be in the “on” position at any one time. In terms of structure, composite views consist of a single parent view (derived from the ViewGroup class and otherwise known as a container view or root element) that is capable of containing other views (known as child views).



**Figure 1.** Illustration of a view hierarchy, which defines a UI layout.

When you compile your application, each XML layout file is compiled into a View resource. You should load the layout resource from your application code, in your Activity.onCreate() callback implementation. Do so by calling setContentView(), passing it the reference to your layout resource in the form of: R.layout.layout\_file\_name. [[1]](#footnote-1)For example, if your XML layout is saved as activity\_main.xml, you would load it for your Activity like so:

public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity\_main);  
}

The onCreate() callback method in your Activity is called by the Android framework when your Activity is launched .

Let consider a main activity with a very simple layout, just a button, and from this activity we activate other activities with more complicated layouts. Each of these layouts will have some specific features.

|  |
| --- |
| res/layout/values/strings.xml |
| <**resources**>  <**string name="app\_name"**>MyLayoutApplication</**string**>   <**string name="ui\_button\_linear\_layout\_title"**>Linear Layout</**string**>  .. . </**resources**> |
| res/layout/activity\_man.xml |
| <**LinearLayout 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"  android:orientation="vertical"  tools:context=".MainActivity"** >   <**Button  android:id="@+id/btn\_linear\_layout"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_margin="10dp"  android:onClick="onLinearLayoutClicked"  android:text="@string/ui\_button\_linear\_layout\_title"** /> </**LinearLayout**> |
| MainActivity |
| **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main***);  }  **public void** onLinearLayoutClicked(View v) {  Intent intent = **new** Intent(MainActivity.**this**,LinearLayoutActivity.**class**);  startActivity(intent); } |

In the code snippet [MainActivity.java](http://www.cs.dartmouth.edu/~campbell/cs65/lecture06/lecture06.txt) loads the [activity\_main\_.xml](http://www.cs.dartmouth.edu/~campbell/cs65/lecture06/lecture06.txt) in the onCreate() callback.

We know from the previous lesson how to activate an activity from another activity using an intent. The parameter v of the type View represents a button instance of the widget button that fires the method onLinearLayoutClicked.It is used the class View as type because this is the base class of all the widgets.

Then let the following figure be the screen provided by the **LinearLayoutActivity**

### Linear Layout

|  |  |
| --- | --- |
| **Main Screen** | **LinearLayout Screen** |
|  |  |

We have here some widgets in this all of them are placed vertically in one direction, therefore the activity layout is a **LinearLayout.**

[LinearLayout](https://developer.android.com/reference/android/widget/LinearLayout.html) is a view group that aligns all children in a single direction, vertically or horizontally. You can specify the layout direction with the [android:orientation](https://developer.android.com/reference/android/widget/LinearLayout.html#attr_android:orientation) attribute.

Let see one by one all the widget in the layout of the second activity

|  |  |
| --- | --- |
| <**LinearLayout 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"  android:orientation="vertical"  tools:context=".MainActivity"** >   <**TextView  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_margin="5dp"  android:text="@string/ui\_profile\_name\_title"** >  </**TextView**>  <**string name="ui\_profile\_name\_title"**>Name</**string**> |  |
| <**EditText  android:id="@+id/editName"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_margin="5dp"  android:hint="@string**/**ui\_profile\_name\_hint"  android:inputType="textCapWords"  android:singleLine="true"** > </**EditText**>  <**string name="ui\_profile\_name\_hint"**>Your name here</**string**> (**this is in strings.xml!!!)** |  |
| <**TextView  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_margin="5dp"  android:text="@string/ui\_profile\_email\_title"** > </**TextView**>  <**string name="ui\_profile\_email\_title"**>Email:</**string**> (**this is in strings.xml!!!)** |  |
| <**EditText  android:id="@+id/editEmail"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_margin="5dp"  android:hint="@string/ui\_profile\_email\_hint"  android:inputType="textEmailAddress"  android:singleLine="true"** > </**EditText**>  <**string name="ui\_profile\_email\_hint"**>Your Email here</**string**> (**this is in strings.xml!!!)** |  |
| <**TextView  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_margin="5dp"  android:text="@string/ui\_profile\_phone\_title"** > </**TextView**>  <**string name="ui\_profile\_phone\_title"**>Phone:</**string**>(**this is in strings.xml!!!)** |  |
| <**EditText  android:id="@+id/editPhone"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_margin="5dp"  android:hint="@string/ui\_profile\_phone\_hint"  android:inputType="phone"  android:singleLine="true"** > </**EditText**>  <**string name="ui\_profile\_phone\_hint"**>Your phone number here</**string**> (**this is in strings.xml!!!)** |  |
| <**TextView  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_margin="5dp"  android:text="@string/ui\_profile\_gender\_title"** > </**TextView**>  <**string name="ui\_profile\_gender\_title"**>Gender:</**string**>(**this is in strings.xml!!!)** |  |
| <**RadioGroup  android:id="@+id/radioGender"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_margin="5dp"  android:orientation="horizontal"** >   <**RadioButton  android:id="@+id/radioGenderF"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_weight="1"  android:text="@string/ui\_profile\_gender\_female"** />   <**RadioButton  android:id="@+id/radioGenderM"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_weight="1"  android:text="@string/ui\_profile\_gender\_male"** >  </**RadioButton**> </**RadioGroup**>  <**string name="ui\_profile\_gender\_male"**>Male</**string**> (**this is in strings.xml!!!)**  <**string name="ui\_profile\_gender\_female"**>Female</**string**>(**this is in strings.xml!!!)** |  |
| <**TextView  android:id="@+id/textInputType"  android:layout\_width="fill\_parent"  android:layout\_height="wrap\_content"  android:layout\_margin="5dp"  android:text="@string/ui\_mood\_spinner\_title"** > </**TextView**>  <**string name="ui\_mood\_spinner\_title"**>How are you feeling:</**string**> (**this is in strings.xml!!!)** |  |
| <**Spinner  android:id="@+id/spinnerInputType"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_margin="5dp"  android:entries="@array/ui\_mood\_spinner\_entries"** > </**Spinner**>  (**the lines below are in strings.xml!!!)**  <**string-array name="ui\_mood\_spinner\_entries"**>  <**item**>Happy</**item**>  <**item**>Sad</**item**>  <**item**>Tired</**item**>  <**item**>Rested</**item**>  <**item**>Stressed</**item**>  <**item**>Fair to middling</**item**>  <**item**>Great</**item**> </**string-array**> |  |
| <**Button  android:id="@+id/btn\_linear\_layout"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_margin="10dp"  android:onClick="onGoBackClicked"  android:text="@string/ui\_button\_goback\_title"** />    </**LinearLayout**>    <**string name="ui\_button\_goback\_title"**>go back</**string**> (**this is in strings.xml!!!)** |  |

The code of the LinearLayoutActivity. We have just a message that shows that we are in the second activity and a method that sends us back in the first activity:

**protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
  
 Toast.*makeText*(getApplicationContext(),  
 getString(R.string.***i\_am\_here\_message***), Toast.***LENGTH\_SHORT***).show();  
  
 setContentView(R.layout.***activity\_linear\_layout***);  
}  
  
**public void** onGoBackClicked(View v) {  
  
 Intent intent = **new** Intent(LinearLayoutActivity.**this**,  
 MainActivity.**class**);  
 startActivity(intent);  
}

### Relative Layout

Now it is easy to create some other Activities, that are activated from the MainActivity in the same way that we used for the LinearLayoutActivity, and with almost the same content. First we add some new buttons in the layout of the MainActivity one for each Activity, that will fire the other activities. The only difference is the layout of each of them, therefore we are presenting here their layouts, and some differences, if we have, in the code of the Activities.

Let create the Activity with the name RelativeLayoutActivity with the respective layout with the name activity\_relative\_layout. The layout of this activity will be a RelativeLayout one.

RelativeLayout is a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent RelativeLayout area (such as aligned to the bottom, left or center). And we will have the screens now:

|  |  |
| --- | --- |
| **Main Screen** | **RelativeLayout Screen** |
|  |  |

The details of the RelativeLayoutActivity layout are as follows

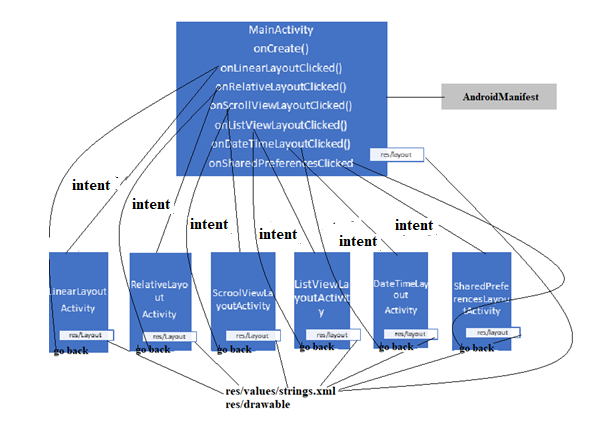
|  |  |
| --- | --- |
| **<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"**  **android:layout\_width="match\_parent"**  **android:layout\_height="match\_parent"**  **android:paddingLeft="16dp"**  **android:paddingRight="16dp" >**  <**AnalogClock  android:id="@+id/analogClock1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_above="@+id/btn\_goback"  android:layout\_centerHorizontal="true"** />  <**string name="ui\_button\_goback\_title"**>go back</**string**>  (**this is in strings.xml!!!)** |  |
| **<ImageView**  **android:id="@+id/imageViewOfStudent"**  **android:layout\_width="wrap\_content"**  **android:layout\_height="wrap\_content"**  **android:layout\_alignParentTop="true"**  **android:layout\_centerHorizontal="true"**  **android:layout\_marginTop="24dp"**  **android:adjustViewBounds="true"**  **android:clickable="true"**  **android:maxHeight="150dp"**  **android:maxWidth="150dp"**  **android:minHeight="50dp"**  **android:minWidth="50dp"**  **android:onClick="onClickConan"**  **android:scaleType="fitXY"**  **android:src="@drawable/conan" />**  **conan.jpg is a picture in drawable folder**  **when you click here the message is displayed**  <**string name="ui\_conans\_message"**>Your school motto is "Vox clamantis in deserto," which means "Voice crying out in the wilderness." This is easily the most pathetic school motto I have ever heard. </**string**>  (**this is in strings.xml!!!)** |  |
| <**RatingBar  android:id="@+id/ratingBar1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_above="@+id/analogClock1"  android:layout\_centerHorizontal="true"  android:layout\_marginBottom="18dp"** /> |  |
| <**Button  android:id="@+id/btn\_goback"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_alignParentBottom="true"  android:layout\_centerHorizontal="true"  android:onClick="onGoBackClicked"  android:text="@string/ui\_button\_goback\_title"** />  </**RelativeLayout**>  <**string name="ui\_button\_goback\_title"**>go back</**string**> (**this is in strings.xml!!!)** |  |

### Scrooll Layout

Let create a new Activity with the name ScrollViewLayoutActivity with the layout activity\_scroll\_view\_layout. The content of the ScrollViewLayoutActivity is the same as the content of LinearLayoutActivity, with the difference that we have here two buttons that play the same role like the button go back in the LinearLayoutActivity. The layout of this activity has elements that we have seen on layouts of LinearLayoutActivity, and RelativeLayoutActivity but we have a new element here , the scroll element that makes possible to scroll the screen for a longer content. There are some other small differences that we are representing here

|  |  |  |  |
| --- | --- | --- | --- |
| **Main Screen** | **ScrollViewLayout Screen(1)** | ScrollViewLayout Screen(2) | |
|  |  |  | |
|  |  |  | |
| <**ScrollView xmlns:android="http://schemas.android.com/apk/res/android"  android:id="@+id/mainLayout"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"** >   <**LinearLayout  android:layout\_width="fill\_parent"  android:layout\_height="wrap\_content"  android:orientation="vertical"** >  . . . . .  </**LinearLayout**>  </**ScrollView**> | | | The sc  the th the scroll element | |
| <**ToggleButton  android:id="@+id/toggleButton1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_margin="5dp"** /> | | |  | |
| <**CheckBox  android:id="@+id/checkBox1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="@string/ui\_profile\_demand\_title"** />  <**string name="ui\_profile\_demand\_title"**>I want an A for else ;-)</**string**>  (**this is in strings.xml!!!)** | | |  | |
| <**LinearLayout  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_margin="5dp"** >   <**Button  android:id="@+id/btnSave"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_marginLeft="40dp"  android:layout\_weight="1"  android:onClick="onSaveClicked"  android:text="@string/ui\_button\_save\_title"** >  </**Button**>   <**Button  android:id="@+id/btnCancel"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_marginRight="40dp"  android:layout\_weight="1"  android:onClick="onCancelClicked"  android:text="@string/ui\_button\_cancel\_title"** >  </**Button**> </**LinearLayout**>  <**string name="ui\_button\_save\_title"**>Saved</**string**> <**string name="ui\_button\_cancel\_title"**>Cancel</**string**>  (**these two lines is in strings.xml!!!)** | | |  | |

## Systems diagram

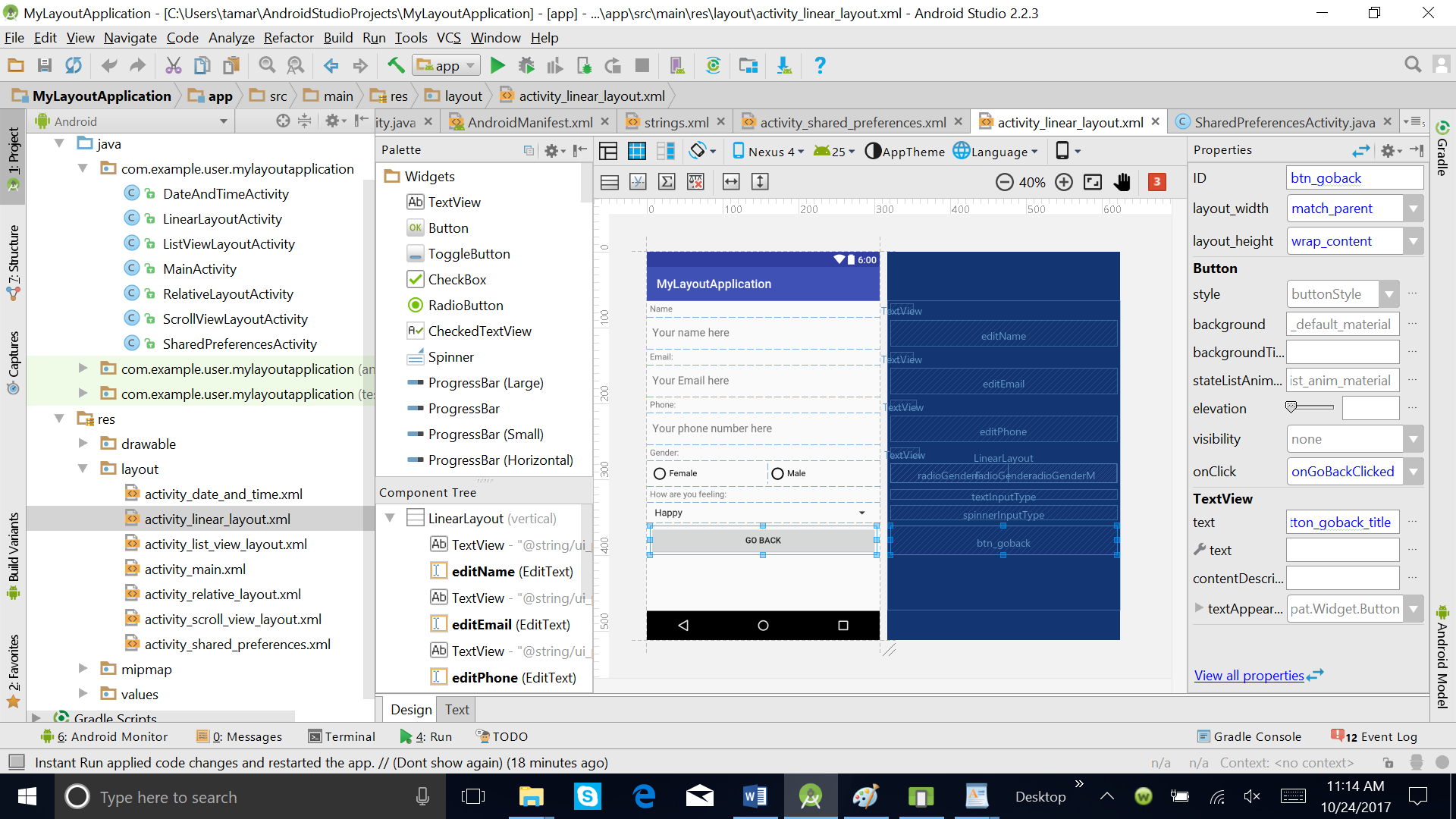
****

The diagram is a blue print for the app. It comprises the MainActivity and the other 6 activities -- one for each of the buttons (e.g., Linear Layout). Example layouts for the UIs are shown as well as the intents that fire activities, snippets of code, and importantly the AndroidManifest which *wires* together all components and resources to make the app work.

ListViewLayoutActivity, DataTimeLayoutActivity and SharedPreferencesLayoutActivity we be discussed in the next lesson

## Writing XML layouts and/or using the design tool

You can build UIs using XML and/or using the design tool. The graphical representation is shown below. On the left of the figure you can see a list of widgets that you can select, drag, drop and arrange on the screen. **Note, that the XML/graphical views can be toggled as shown in the figure**. The figure shows the type of layout and highlights the first button -- on the right side you can see the properties of the button, for example, the button id @+**="@+id/btn\_linear\_layout**, the width and height are match\_parent and wrap\_content respectively.



In fact, you can define your UI with XML or the graphical tool and programmatically in your code in you wish. It is recommended specifying your UI not in your program but using the xml presentation. Why? Because it is a good idea to separate the specification of UI from your code. It is more extensible (for example, having different layouts for different devices) rather than littering your program with lots of unnecessary UI code. By keeping things separate you can completely change your UI without changing your code -- that's cool.

## Styles and themes

Themes are Android's mechanism for applying a consistent style to an app or activity. The style specifies the visual properties of the elements that make up your user interface, such as color, height, padding and font size. Android provides three system themes that you can choose from when building apps (Android 5.0+).

* Material (dark version)
* Material Light (light version)
* Material Light with dark action bars (we will discuss action bars later)

When designing the UI you could start with selecting the theme that matches your needs and then, if you wish, customize it further. For now let's look at the layout app with different styles - Theme.Holo.Light (left) and Theme.Holo (right).

|  |  |
| --- | --- |
|  |  |

These are set in the xml file res/values/style.xml

<**style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar"**>  
 </**style**>

The manifest specifies android:theme="@style/AppTheme">

What is the difference between a theme and style?

A style is a collection of properties that specify the look and format for a view or screen. A style can specify properties such as height, padding, font color, font size, background color, and much more. A style is defined in an XML resource that is separate from the XML that specifies the layout. A theme is a style applied to an entire activity or application, rather than an individual view. When a style is applied as a theme, every view in the activity or application will apply each style property that it supports.

In our example we have

**styles.xml**

<**resources**>  
 *<!-- Base application theme. -->* <**style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar"**>  
 *<!-- Customize your theme here. -->* <**item name="colorPrimary"**>@color/colorPrimary</**item**>  
 <**item name="colorPrimaryDark"**>@color/colorPrimaryDark</**item**>  
 <**item name="colorAccent"**>@color/colorAccent</**item**>  
 </**style**>  
</**resources**>

**colors.xml**

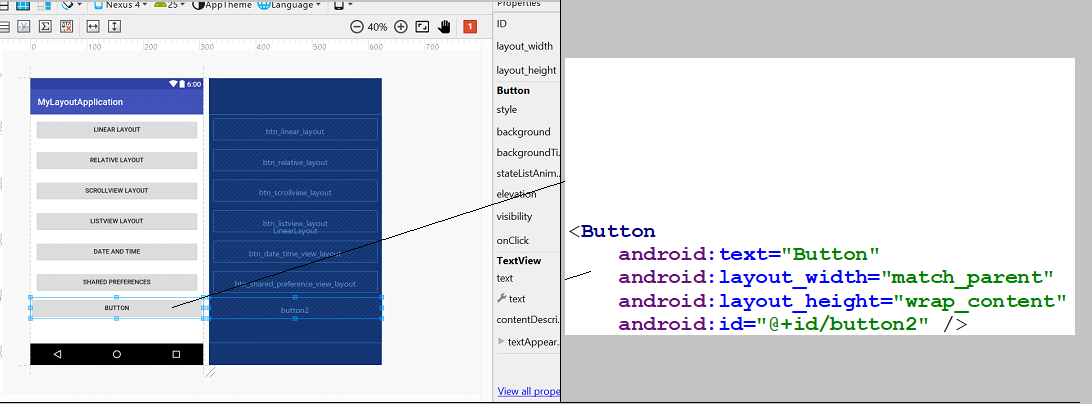
*<?***xml version="1.0" encoding="utf-8"***?>*<**resources**>  
 <**color name="colorPrimary"**>#3F51B5</**color**>  
 <**color name="colorPrimaryDark"**>#303F9F</**color**>  
 <**color name="colorAccent"**>#FF4081</**color**>  
</**resources**>

**dimens.xml**

<**resources**>  
 *<!-- Default screen margins, per the Android Design guidelines. -->* <**dimen name="activity\_horizontal\_margin"**>16dp</**dimen**>  
 <**dimen name="activity\_vertical\_margin"**>16dp</**dimen**>  
</**resources**>

## Using the design mode to implement the UI

You can implement the layout using XML or using the design mode as mentioned earlier -- or a combo. Below, the properties of the layout (e.g., layout\_width, orientation, etc.) can be set by first drop and dragging the button (in this example case) and the highlight the button and left click to set the properties of the button such as id, layout\_width, layout\_height, layout\_margin, onClick, text.

**android:layout\_width="match\_parent"**

Note, that we use labels for text and not direct text (go back). All string resources are defined in strings.xml file in the resources/values folder - a snippet relating to [srings.xml](http://www.cs.dartmouth.edu/~campbell/cs65/lecture06/lecture06.txt) for the linear layout are shown below. Again, you can use the graphical mode to define strings or just cut and paste in the strings.xml (which is preferable).

Note, that strings can hold a wide set of items: strings, arrays of strings, integer arrays, colors, styles/themes. We will mostly define strings, string arrays, integer arrays.

For example, we want people to enter their mood using a drop down/up menu selection. We use the Spinner for this and populate it with a set of moods defined by entries that is @array/ui\_mood\_spinner\_entries which is defined in strings.xml. Later in ListView layout we will add an array to the layout programmatically in the code but here we do it all in the xml -- below is a snippet from linear\_layout.xml

<Spinner

android:id="@+id/spinnerInputType"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_margin="5dp"

android:entries="@array/ui\_mood\_spinner\_entries" >

</Spinner>

And here is the corresponding string array of entries defined in strings.xml. It is better to put these strings and resources in XML and in the resource folder. It is easy to maintain -- if you change the list entries later you simply change the XML and not the code. Again, this is a nice separation of code and resources i.e., data.

<string-array name="ui\_mood\_spinner\_entries">

<item>Happy</item>

<item>Sad</item>

<item>Tired</item>

<item>Rested</item>

<item>Stressed</item>

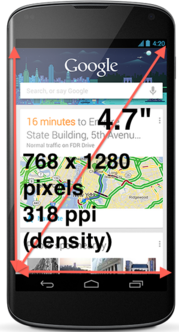
<item>Fair to middling</item>

<item>Great</item>

</string-array>

## Dimensions of a phone used in laying out the UI

How you design a layout is determined by the real estate of the screen -- essentially the size (read pixels dimensions) and pixel density. The figure below shows these display properties of the Nexus 4. The physical size of the phone is 4.7 inches diagonally, the resolution is 768 x 1280 pixels, and the pixel density is 318 ppi (where ppi is *pixels per inch*). Why does this matter. Well we have to know a little about these display characteristics when designing our views.



When specifying the UI you need to be aware of a number of scaling units. The first is dp, which stands for density-independent pixel. Most people confuse dp with ppi mentioned above. 1 dp is equivalent to one pixel on a 160 dpi screen. Android specifies four screen densities that you see reflected when you create a project. The quantity of pixels within a physical area of the screen is referred to as dpi (dots per inch). Typically, a low density screen has fewer pixels within a given physical area, compared to a medium or high density screens. Android groups all actual screen densities into four generalized densities:

* low density (ldpi) which is 120 dpi
* medium density (mdpi) is 160 dpi
* high density (hdpi) is 240 dpi
* extra high density (xhdpi) is 320 dpi

Examples of dp being used include *android:layout\_margin="5dp"* where the margin inside the border of a view is 5dp all around. Another example is using fount size. Here we use sp: the units used below for android:textSize="30sp" is based on scale-independent pixels (sp).

\*\* used to set margins in some of the layouts \*\*

android:layout\_marginRight="40dp"

android:layout\_margin="5dp"

\*\* used to specify the absolute size of Conan's image\*\*

android:maxHeight="250dp"

android:maxWidth="250dp"

android:minHeight="50dp"

android:minWidth="50dp"

\*\* used to specify font size

android:textSize="30s

## Operations and Attributes of Views

Once you have created a set of views, viewgroups, etc. there are a common set of operations and attributes associated with views.

**. properties**: for example, setting the text of a TextView such as the text, style, font size, position on the screen, how much real estate the view should take up on the screen. These properties can be set in the XML or programmatically in the code.

**. focus**: When you design a screen made up of a bunch of views and view groups you may typically want to focus on one of the views or different views from time to time. You can do this in XML with or programmatically call requestFocus().

**. listeners**: Many times, when you are design a view the user needs to interact with the object. The user can setup listeners or callbacks that will be called when an event fires and specific behavior programmed; for example, when a RadioButton is set by the user the state is saved.

If we look at the attributes associated with the Linear Layout XML (above) we will see a number of strange looking syntax -- these attributes include the following:

<EditText

android:id="@+id/editName"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_margin="5dp"

android:hint="@string/ui\_profile\_name\_hint"

android:inputType="textCapWords"

android:singleLine="true" >

</EditText>

* android:id = "@+id/editName"; represents the view object ID which is typically a string such as "@+id/editName"
* android:layout\_width="match\_parent"; match\_parent, which means that the view wants to be as big as its parent minus any padding).
* android:layout\_height="wrap\_content"; which means that the view wants to be just big enough to enclose its content plus any padding.
* android:layout\_margin="5dp"; this is the space outside of the border of the view (e.g., a button) and between what is next to or around the view.
* android:hint="@string/ui\_profile\_name\_hint"; when associated in this case with a EditText it is a hint of what the user should enter (e.g., your email).
* android:inputType="textCapWords"; The user input where the letter of the each word will begin with a capital by default -- for example in this case the users name Andrew Thomas Campbell. There are many other types of input such textEmailAddress and textAutoComplete.
* android:singleLine="true"; this restricts the input by the user to a single line.

There are many other attributes associated with views.

First, here are a bunch of margin based attributes. BTW, padding is the space inside the border between the border and the actual view's content. There is padding all around the content. Margins is outside the border between the border and the other elements next to this view.

* android:layout\_marginBottom: specifies extra space on the bottom side of this view.
* android:layout\_marginEnd; specifies extra space on the end side of this view.
* android:layout\_marginLeft; specifies extra space on the left side of this view.
* android:layout\_marginRight; sSpecifies extra space on the right side of this view.
* android:layout\_marginStart; specifies extra space on the start side of this view.
* android:layout\_marginTop; specifies extra space on the top side of this view.

Some other important attributes include:

* android:layout\_gravity: Standard gravity constant that a child supplies to its parent. Defines how the child view should be positioned, on both the X and Y axes, within its enclosing layout; some positions include top, botton, left and right; start, center and end.
* android:layout\_weight; specifies how child views are specified, specifically, how much of the extra space will be allocated to the view: 0 if the view should not be stretched. Otherwise the extra pixels will be pro-rated among all views whose weight is greater than 0. Note, how layout\_weight is used within the RadioGroup for the Linear Layout snippet below -- each of the RadioButtons is equally weighted in the RadioGroup.

<RadioButton

android:id="@+id/radioGenderF"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_weight="1"

android:text="@string/ui\_profile\_gender\_female" />

<RadioButton

android:id="@+id/radioGenderM"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_weight="1"

android:text="@string/ui\_profile\_gender\_male" >

</RadioButton>

The RelativeLayout has a number of properties that all makes good sense; match\_parent will fill the screen and the padding left and right positions the other widgets in the container.

RelativeLayoutActivity is simple code with two callbacks - one for onGoBackClicked and onClickConan. Note that in both cases the view object is passed to the callback method.

public class RelativeLayoutActivity extends Activity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.relative\_layout);

}

public void onGoBackClicked(View v) {

Intent intent = new Intent(RelativeLayoutActivity.this,

MainActivity.class);

startActivity(intent);

}

public void onClickConan(View v) {

Toast.makeText(this, getString(R.string.ui\_conans\_message),

Toast.LENGTH\_LONG).show();

}

}

## References

https://developer.android.com/guide/topics/ui/index.html

<https://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html>

<https://developer.android.com/guide/topics/ui/look-and-feel/themes.html>

## Assignment

* Implement the above project (partially, the second part will be in the next class) completing the code that it is missing. You can change some of attributes, for example the colors, content of the fields.

1. **Android** R.java is an auto-generated **file** by aapt (**Android** Asset Packaging Tool) that contains resource IDs for all the resources of res/ directory. If you create any component in the activity\_main.xml **file**, id for the corresponding component is automatically created in this **file**. [↑](#footnote-ref-1)