IT – Entrepreneurship
User Interface (UI)

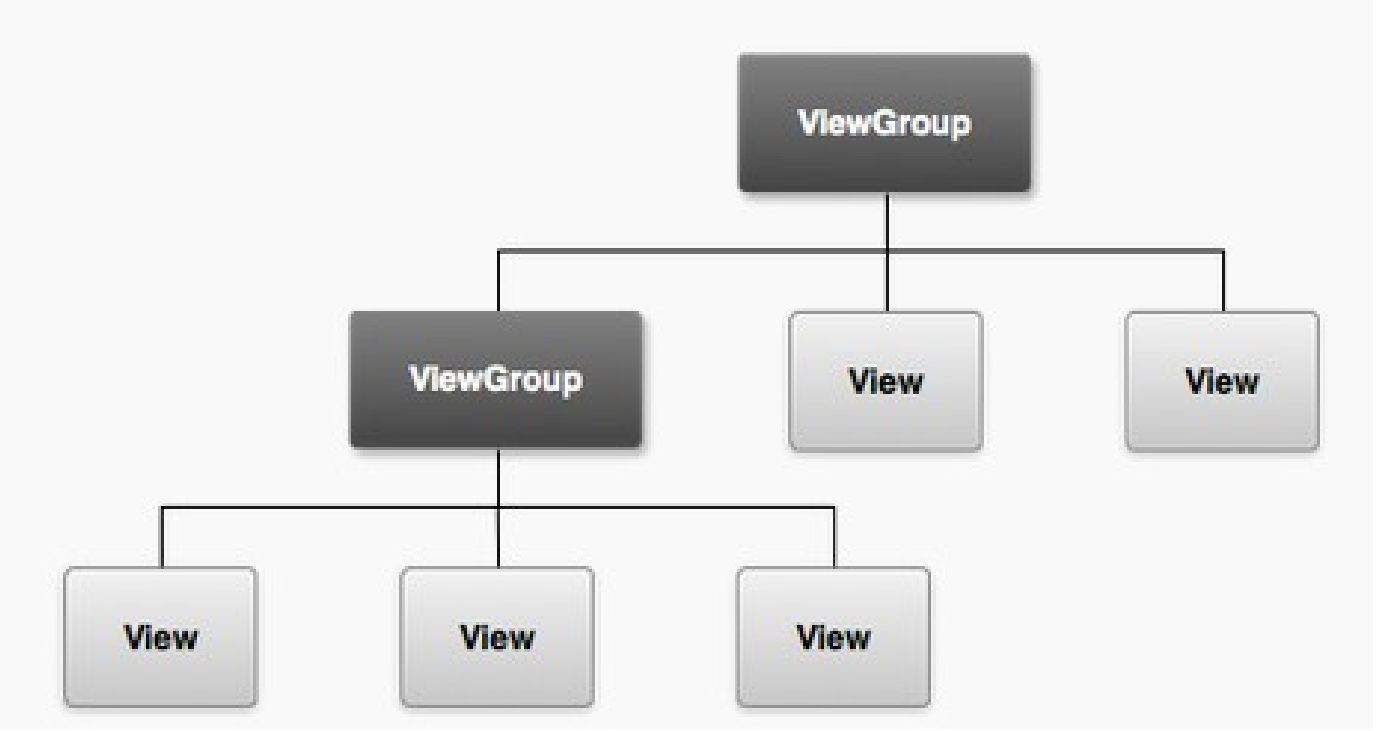
User Interface

- The user interface is implemented as collection of view objects
- A view is a class and a widget which is drawn on some part of the screen and is responsible for event handling such as when the user interacts with the UI

User Interface

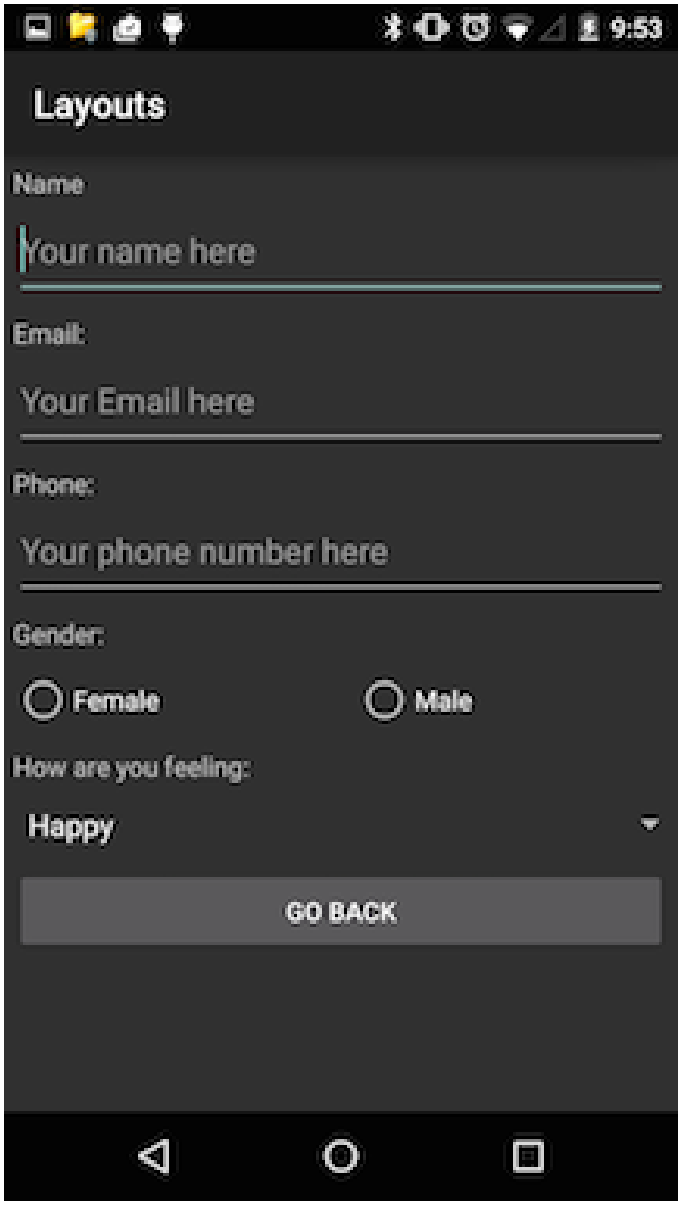
- User's can construct sophisticated UIs by bundling views together using layouts (or ViewGroups) which can be considered as invisible containers.
- These containers can hold child containers.
- Each container defines its views (or other ViewGroups) and their layout properties.

User Interface



User Interface

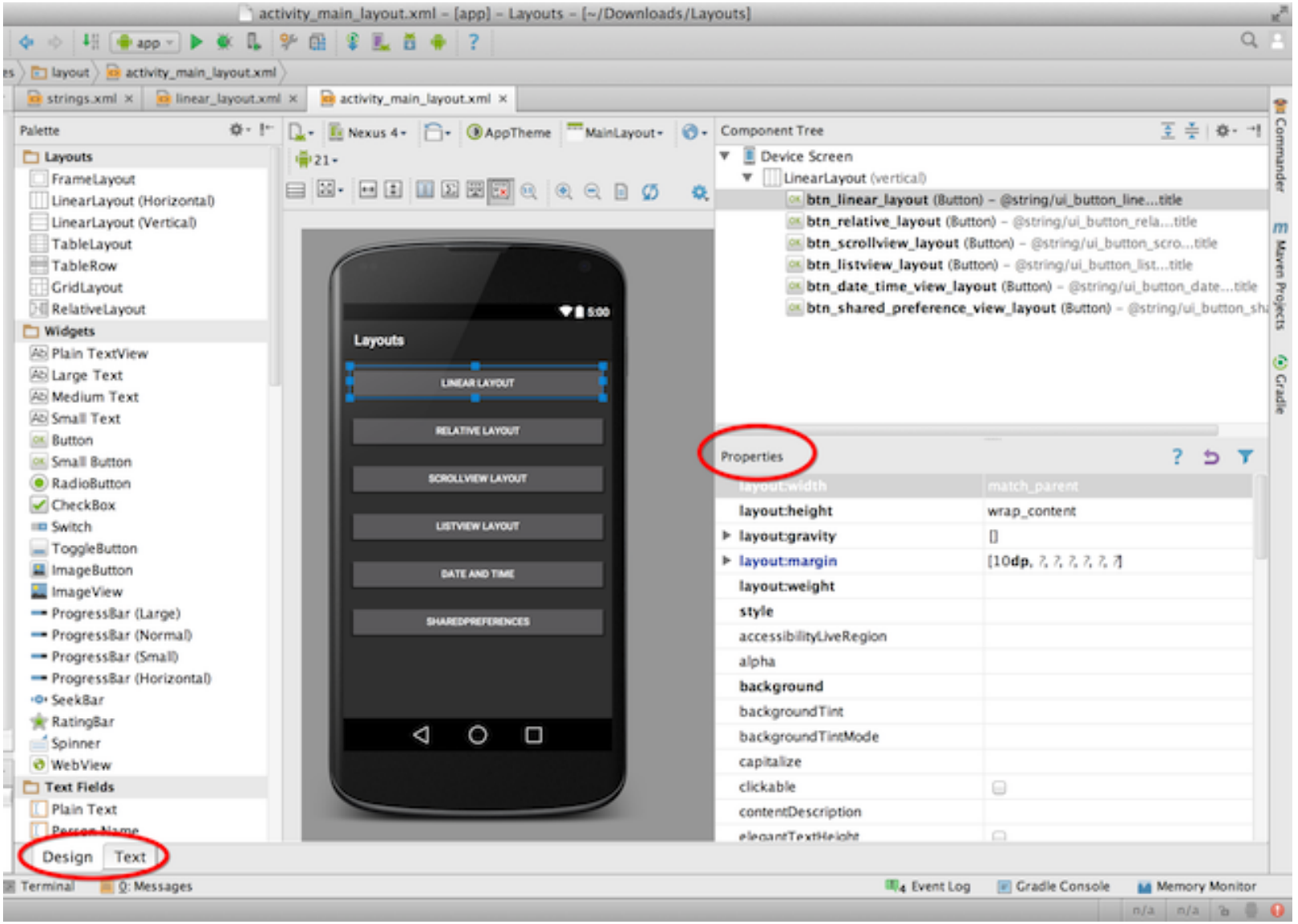
- Take a look at this simple UI
- It consists of a simple linear vertical layout of widgets including TextViews, EditTexts, RadioButtons, a Spinner and a button.



Sample Project

- Import the sample project ‘Layouts’
- Start the app and try to understand how things are done.

Writing XML layouts and/or using the design tool



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.

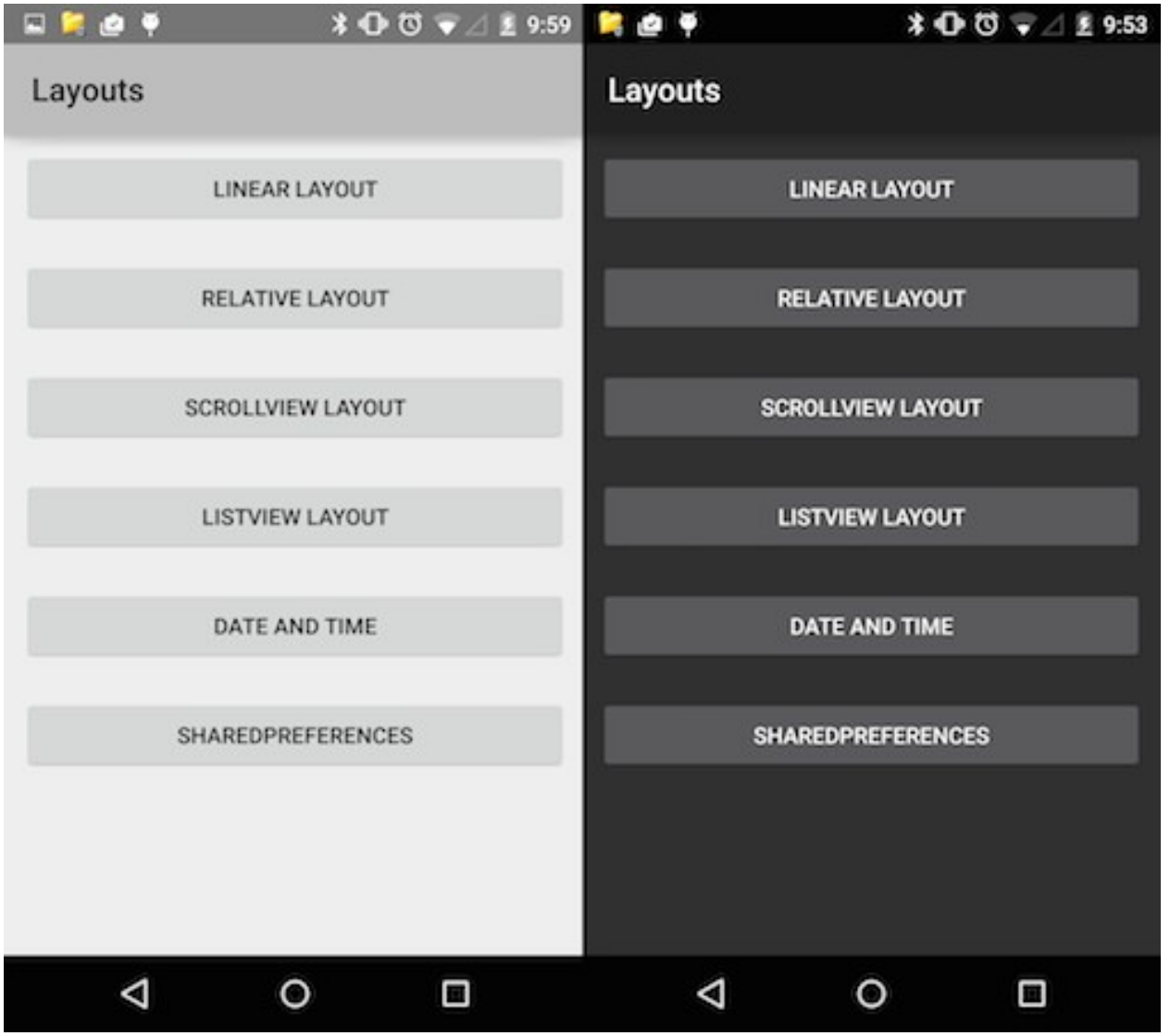
Styles and themes

- The manifest specifies android:theme="@style/AppTheme"
- Android provides three system themes that you can choose from when building apps for Lollipop (Android 5.0+)
- Material (dark version)
- Material Light (light version)
- Material Light with dark action bars

Styles and themes

- For backward compatibility
- Theme.AppCompat (Dark version)
- Theme.AppCompat.Light (Light version)
- Theme.AppCompat.Light.DarkActionBar

Styles and themes



LinearLayout

- LinearLayout 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 attribute.
- All children of a LinearLayout are stacked one after the other

LinearLayout

- LinearLayout also supports assigning a weight to individual children with the android:layout_weight attribute.
- A larger weight value allows it to expand to fill any remaining space in the parent view.
- 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.

LinearLayout

- For example, if there are three text fields and two of them declare a weight of 1, while the other is given no weight, the third text field without weight will not grow and will only occupy the area required by its content.
- The other two will expand equally to fill the space remaining after all three fields are measured.
- If the third field is then given a weight of 2 (instead of 0), then it is now declared more important than both the others, so it gets half the total remaining space, while the first two share the rest equally.

LinearLayout

```
<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=".MainLayoutActivity" >

    **snippet**

    <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" />

    <Button
        android:id="@+id/btn_relative_layout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:onClick="onRelativeLayoutClicked"
        android:text="@string/ui_button_relative_layout_title" />

</LinearLayout>
```

LinearLayout

```
public class MainLayoutActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main_layout);
        Toast.makeText(getApplicationContext(),
                getString(R.string.i_am_here_message),
Toast.LENGTH_SHORT).show();

    }

    public void onLinearLayoutClicked(View v) {

        Intent intent = new Intent(MainLayoutActivity.this,
                LinearLayoutActivity.class);
        startActivity(intent);
    }
}
```

LinearLayout

```
public class LinearLayoutActivity extends Activity {

    @Override
    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.linear_layout);
    }

    public void onGoBackClicked(View v) {

        Intent intent = new Intent(LinearLayoutActivity.this,
                MainLayoutActivity.class);
        startActivity(intent);
    }

}
```

LinearLayout

```
<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>
```

LinearLayout

```
<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>
```

LinearLayout

```
<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

- 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.
- 1 dp is equivalent to one pixel on a 160 dpi screen.
- The quantity of pixels within a physical area of the screen is referred to as dpi (dots per inch).

Dimensions of a phone used in laying out the UI

- 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 some generalized densities

Dimensions of a phone used in laying out the UI

- 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
- extra-extra high density (xxhdpi) is 480 dpi
- Extra-extra-extra high density (xxxhdpi) is 640 dpi

Dimensions of a phone used in laying out the UI

- Need more clarity? StackOverflow to the rescue!

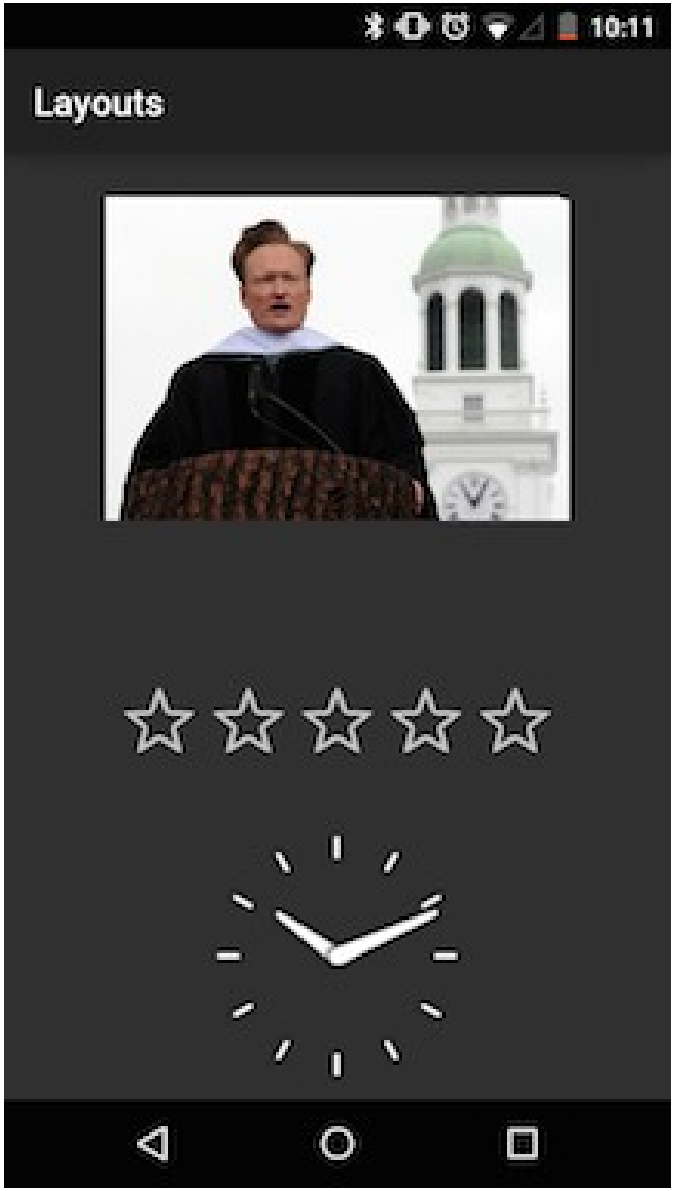
Operations and Attributes of Views

- Their are a common set of operations and attributes associated with views.
- example setting the text of a TextView such as the text, style, font size, position on the screen.
- These properties can be set in the XML or programmatically in the code.
- To focus on one of the views or different views from time to time. We can do this in XML with or programmatically call requestFocus().

Operations and Attributes of Views

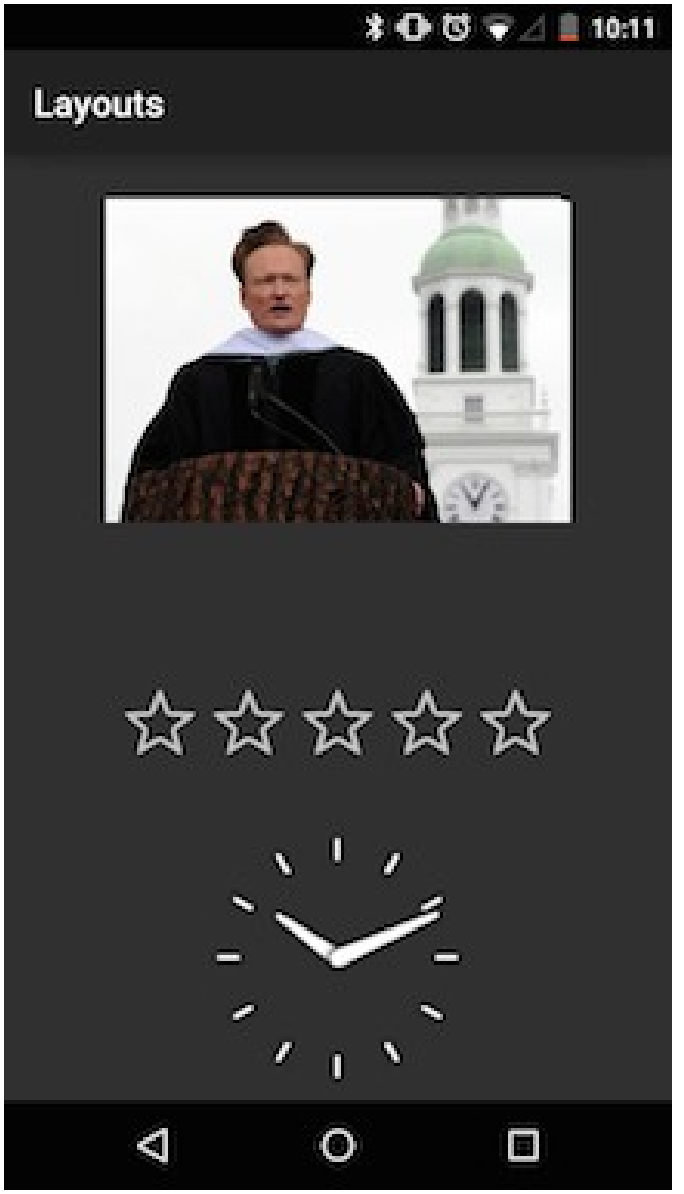
- The user can setup listeners or callbacks that will be called when an event fires and specific behavior programmed

RelativeLayout

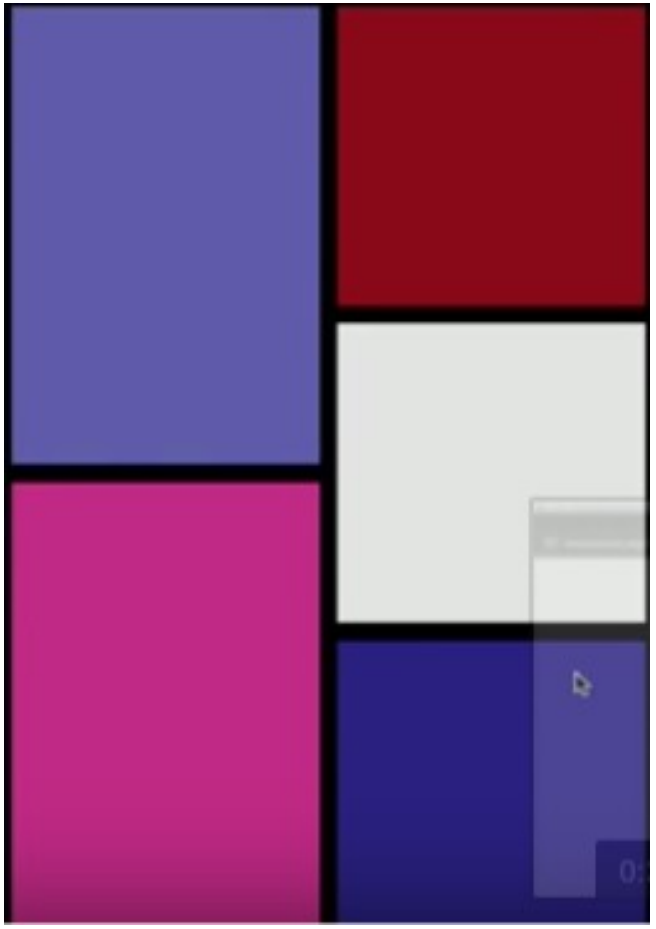


RelativeLayout

- Here we locate the AnalogClock at bottom of the layout (layout_alignParentBottom)
- The ImageView of our pal Conan at the top (layout_alignParentTop)
- The RatingBar is sandwiched between the two widgets.

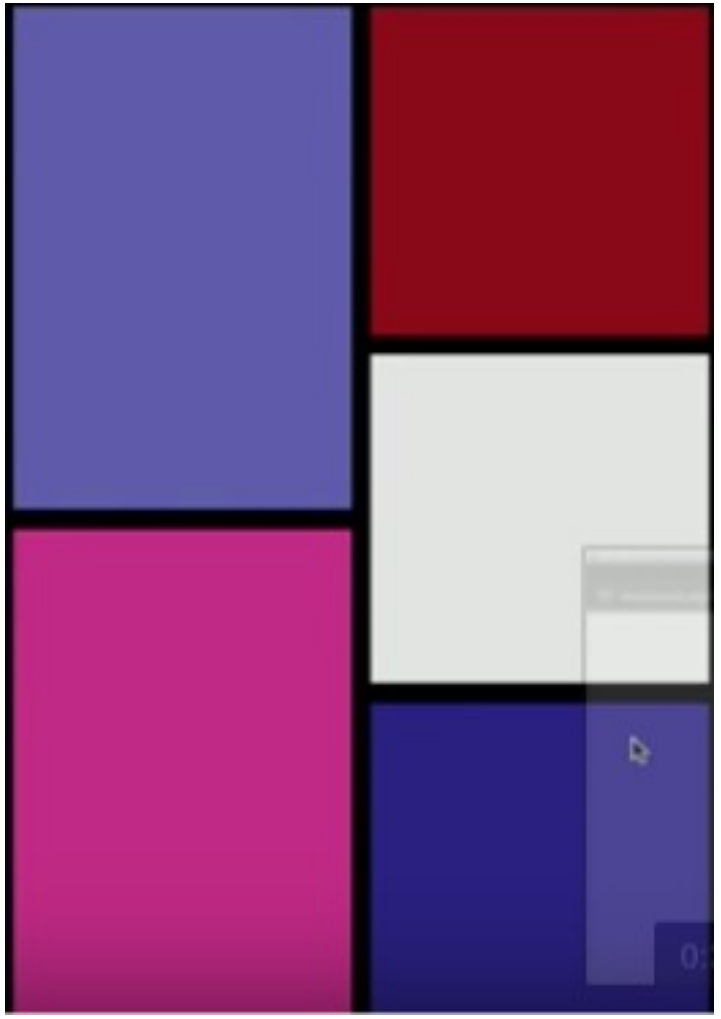


Task



- Achieve a UI similar to this with what we have learnt so far.

Home Assignment



- Achieve a similar UI to this image.
- Apply your creativity to it.
- At the bottom place a scrollbar, attach an eventlistener to it and change the colors to bright or dark based on the scroll.

References

- cs.dartmouth.edu
- Android Developers