ANDROID FUNDAMENTALS

Seven Advanced Academy

Layouts, Views and Resources

Lesson 2

Contents

- Views, view groups, and view hierarchy
- Layouts in XML and Java code
- Event Handling
- Resources
- Screen Measurements

Contents

- Use the layout editor in Android Studio
- Position views within a RelativeLayout
- Position views within a ConstraintLayout
- Create variants of the layout for landscape orientation and larger displays

Tasks

- Create an app and add user interface elements such as buttons in the Layout Editor
- Edit the app's layout in XML
- Add a button to the app. Use a string resource for the label
- Implement a click handler method for the button to display a message on the screen when the user clicks
- Change the click handler method to change the message shown on the screen

Tasks

- Create the Hello toast app
- Copy and refactor the Hello Relative app to create Hello Constraint
- Experiment with using RelativeLayout and ConstraintLayout
- Copy and refactor the Hello Toast app to create the Hello Relative app
- Change the root view group in the main layout to be a RelativeLayout

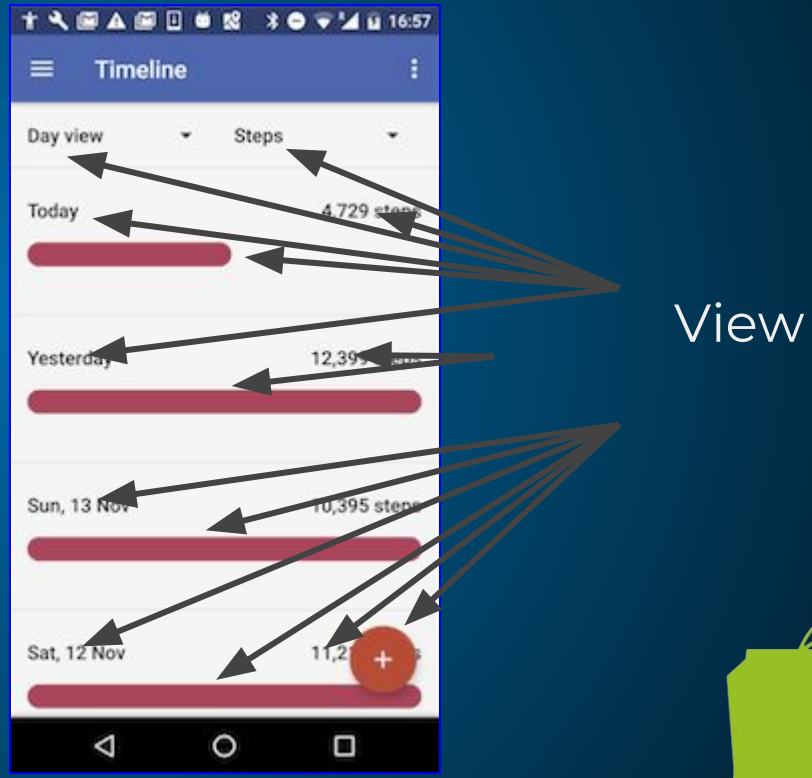
Tasks

- Rearrange the views in the main layout to be relative to each other
- Change the root view group in the main layout to be ConstraintLayout
- Modify the layout to add constraints to the views
- Modify the views for layout variants for landscape orientation and larger displays

Views

Everything you see is a View

If you look at your mobile device, every user interface element that you see is a View



What's a View ?

Views are Android's basic user interface building blocks.

- display text (TextView class), edit text (EditText class)
- buttons (Button class), menus, other controls
- scrollable (ScrollView, RecyclerView)
- show images (ImageView)
- subclass of View class

Views have Properties

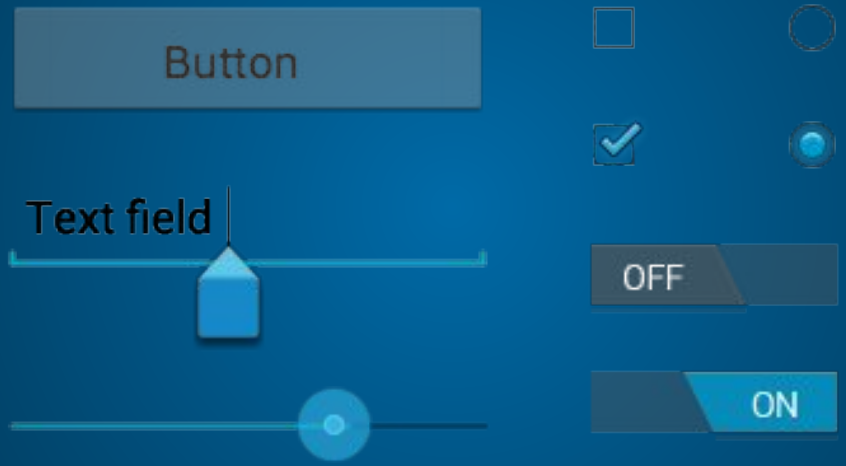
- Have properties (e.g., color, dimensions, positioning)
- May have focus (e.g., selected to receive user input)
- May be interactive (respond to user clicks)
- May be visible or not
- Have relationships to other views

Examples of View

Button

EditText

SeekBar



CheckBox

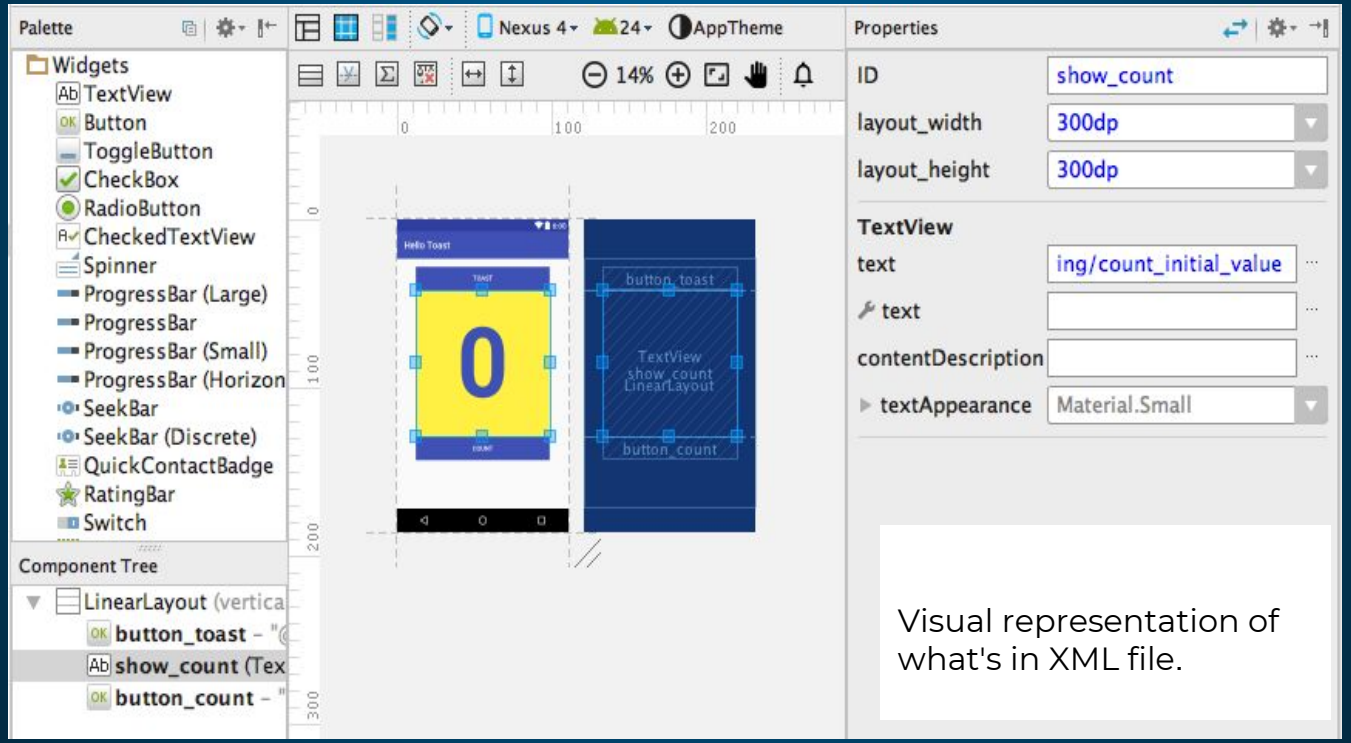
RadioButton

Switch

Creating and laying out views

- Graphically within Android Studio
- XML Files
- Programmatically

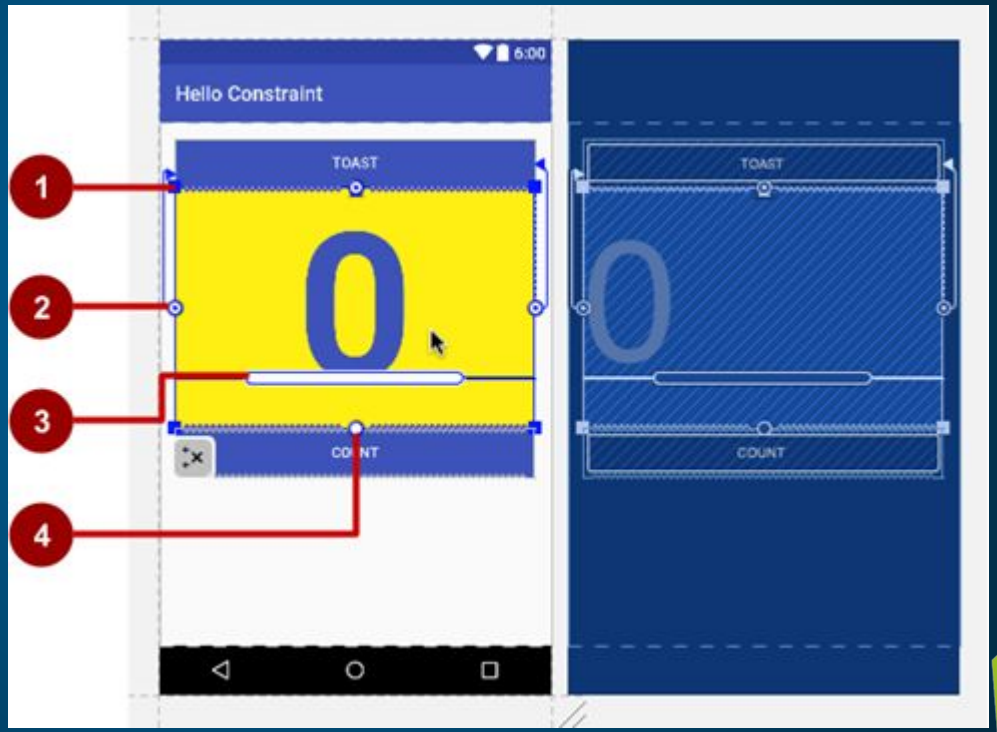
Views defined in Layout Editor



Visual representation of what's in XML file.

Using the Layout Editor

1. Resizing handle
2. Constraint line and handle
3. Baseline handle
4. Constraint handle



Views defined in XML

```
<TextView
    android:id="@+id/show_count"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/myBackgroundColor"
    android:text="@string/count_initial_value"
    android:textColor="@color/colorPrimary"
    android:textSize="@dimen/count_text_size"
    android:textStyle="bold"
/>
```

View Properties in XML

android:<property_name>="<property_value>"
Example: android:layout_width="match_parent"

android:<property_name>="@<resource_type>/resource_id"
Example: android:text="@string/button_label_next"

android:<property_name>="@+id/view_id"
Example: android:id="@+id/show_count"

Create View in Java Code

In an Activity:

context

```
TextView myText = new TextView(this);
myText.setText("Display this text!");
```

What’s the Context ?

- Context is an interface to global information about an application environment
- Get the context:
  Context context = getApplicationContext();
- An activity is its own context:
  TextView myText = new TextView(this);

Custom View

- Over 100 (!) different types of views available from the Android system, all children of the View class
- If necessary, create custom views by subclassing existing views or the View class

ViewGroup & View Hierarchy

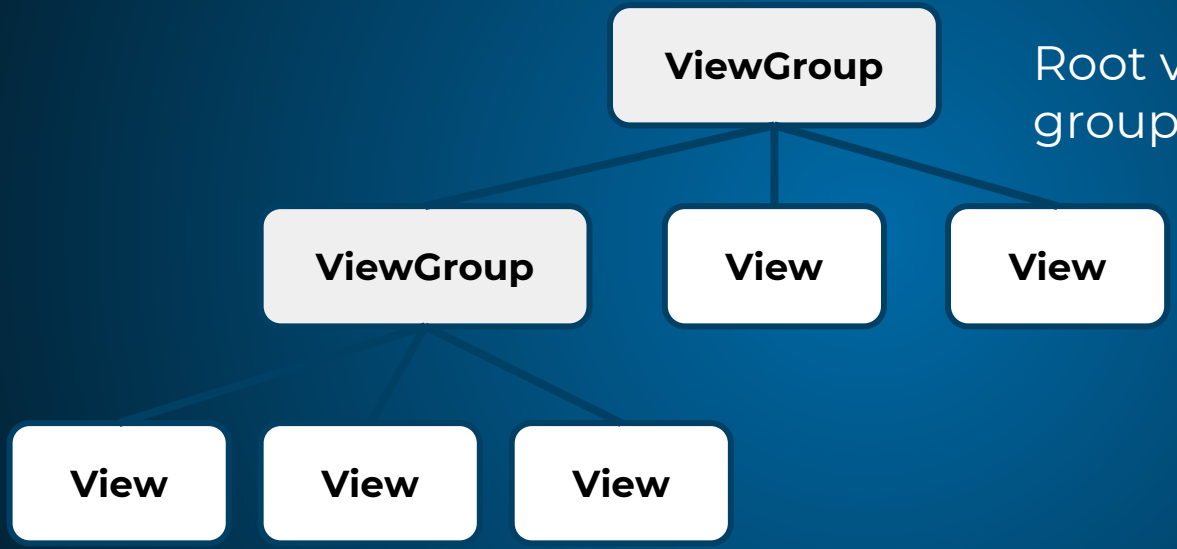
ViewGroup Views

A ViewGroup (parent) is a type of view that can contain other views (children)

ViewGroup is the base class for layouts and view containers

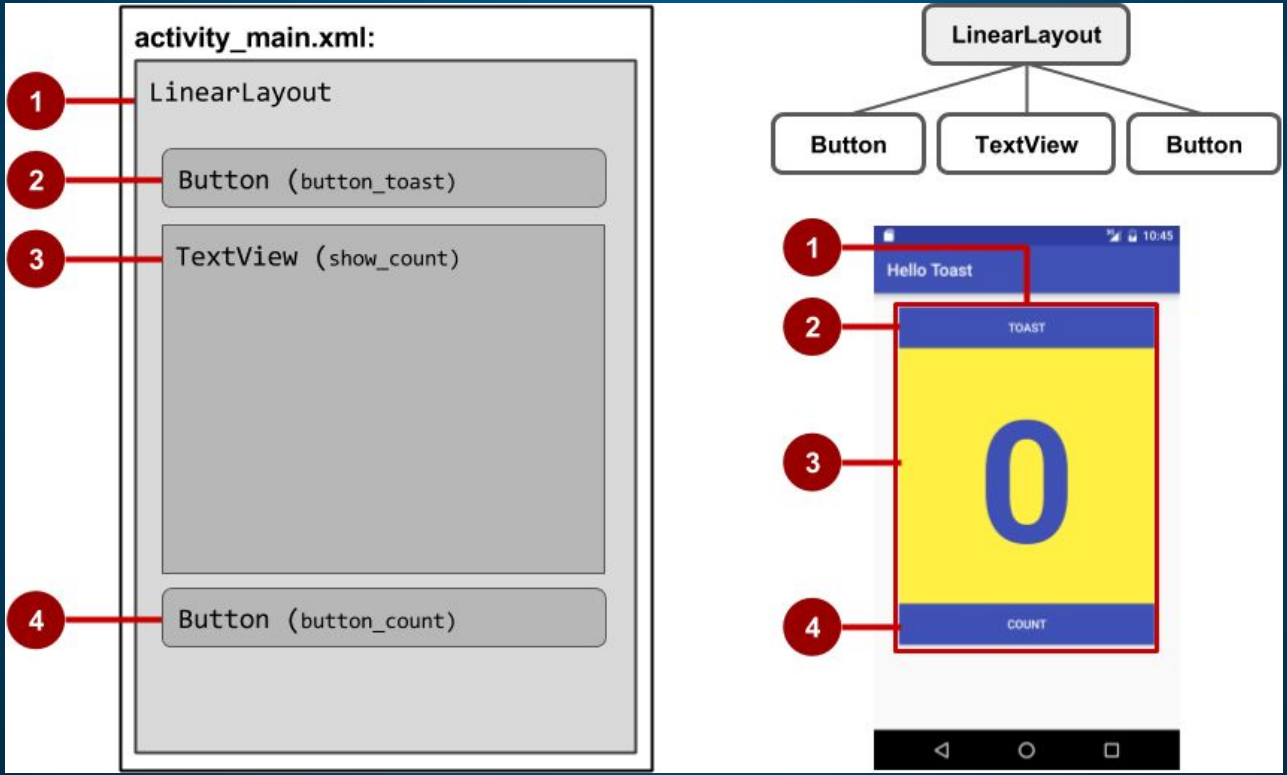
- ScrollView—scrollable view that contains one child view
- LinearLayout—arrange views in horizontal/vertical row
- RecyclerView—scrollable "list" of views or view groups

Hierarchy of View Groups and Views

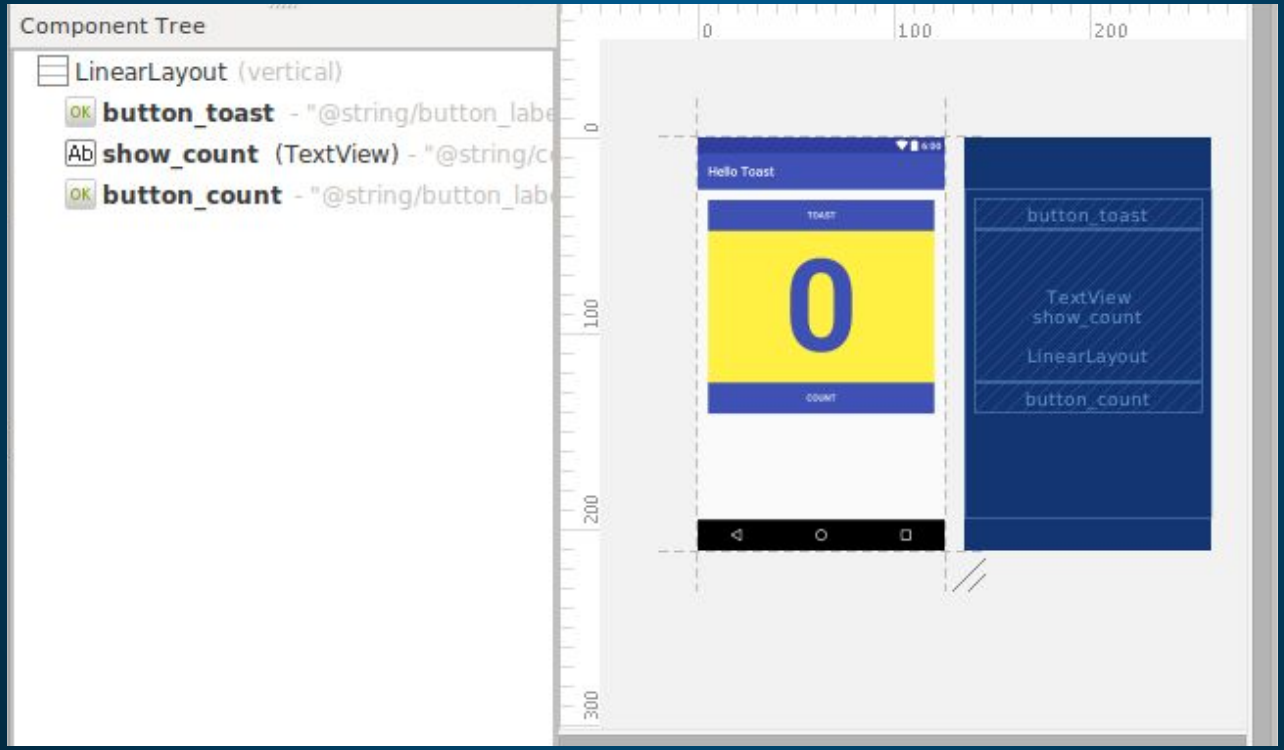


Root view is always a view group

View Hierarchy and Screen Layout



View Hierarchy in the Component Tree



Best Practices for View Hierarchies

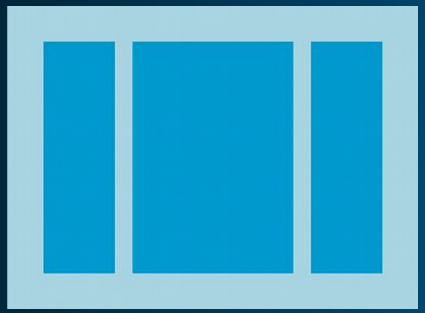
- Arrangement of view hierarchy affects app performance
- Use smallest number of simplest views possible
- Keep the hierarchy flat—limit nesting of views and view groups

Layout

Layout View

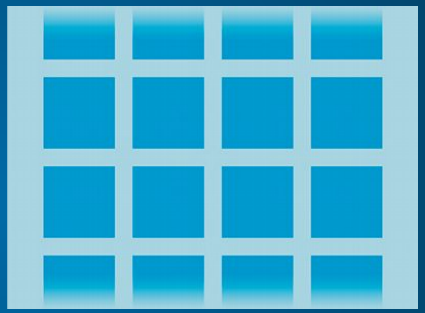
- Layouts
  - are specific types of view groups
  - are subclasses of ViewGroup
  - contain child views
  - can be in a row, column, grid, table, absolute

Common Layout Class

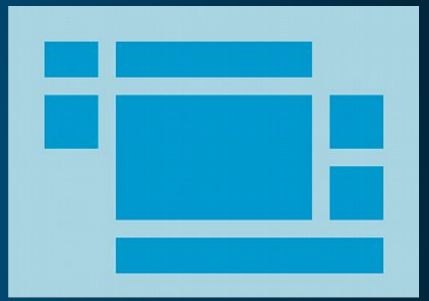


LinearLayout

RelativeLayout



GridLayout



TableLayout

Common Layout Classes

- **ConstraintLayout -** connect views with constraints
- **LinearLayout -** horizontal or vertical row
- **RelativeLayout -** child views relative to each other
- **TableLayout -** rows and columns
- **FrameLayout -** shows one child of a stack of children
- **GridView -** 2D scrollable grid

Class Hierarchy vs. Layout Hierarchy

- View class-hierarchy is standard object-oriented class inheritance
  - For example, Button is-a TextView is-a View is-a Object
  - Superclass-subclass relationship

- Layout hierarchy is how Views are visually arranged
  - For example, LinearLayout can contain Buttons arranged in a row
  - Parent-child relationship

Layout created in XML

```
<LinearLayout
  android:orientation= "vertical"
  android:layout_width= "match_parent"
  android:layout_height= "match_parent">
    <EditText
      ... />
    <Button
      ... />
</LinearLayout
```

Layout created in Java Activity code

```
LinearLayout linearL = new LinearLayout(this);
linearL.setOrientation(LinearLayout.VERTICAL);

TextView myText = new TextView(this);
myText.setText("Display this text!");

linearL.addView(myText);
setContentView(linearL);
```

Setting width and height in Java code

- Set the width and height of a view:

```
LinearLayout.LayoutParams layoutParams =
  new Linear.LayoutParams(
     LayoutParams.MATCH_PARENT,
     LayoutParams.WRAP_CONTENT);

myView.setLayoutParams(layoutParams);
```

Event Handling

Events

Something that happens

- In UI: Click, tap, drag
- Device: DetectedActivity such as walking, driving, tilting
- Events are "noticed" by the Android system

Event Handlers

- Methods that do something in response to a click
- A method, called an **event handler**, is triggered by a specific event and does something in response to the event

Handling clicks in XML & Java

Attach handler to view in layout:

```
android:onClick="showToast"
```

Implement Handler in Activity :

```
public void showToast(View view) {
  String msg = "Hello Toast!";
  Toast toast = Toast.makeText(
       this, msg, duration);
  toast.show();
  }
}
```

Setting click handlers in Java

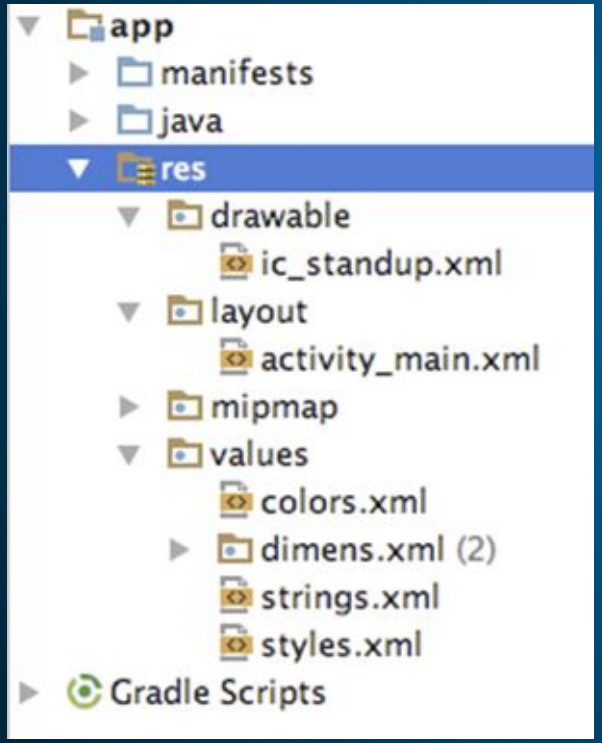
```
final Button button = (Button)
findViewById(R.id.button_id);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        String msg = "Hello Toast!";
        Toast toast = Toast.makeText(this, msg, duration);
        toast.show();
        }
    });
```

Resources

Resources

- Separate static data from code in your layouts.
- Strings, dimensions, images, menu text, colors, styles
- Useful for localization

Where are the resource in your project?



Resources and resource files stored in **res** folder

Refer to resources in code

Layout:

```
R.layout.activity_main
setContentView(R.layout.activity_main);
```

View:

```
R.id.recyclerview
rv = (RecyclerView) findViewById(R.id.recyclerview);
```

String:

```
In Java: R.string.title
In XML: android:text="@string/title"
```

Measurements

- Device Independent Pixels (dp) - for Views
- Scale Independent Pixels (sp) - for text

Don't use device-dependent units:

- ~~Actual Pixels (px)~~
- ~~Actual Measurement (in, mm)~~
- ~~Points - typography 1/72 inch (pt)~~

Let’s code

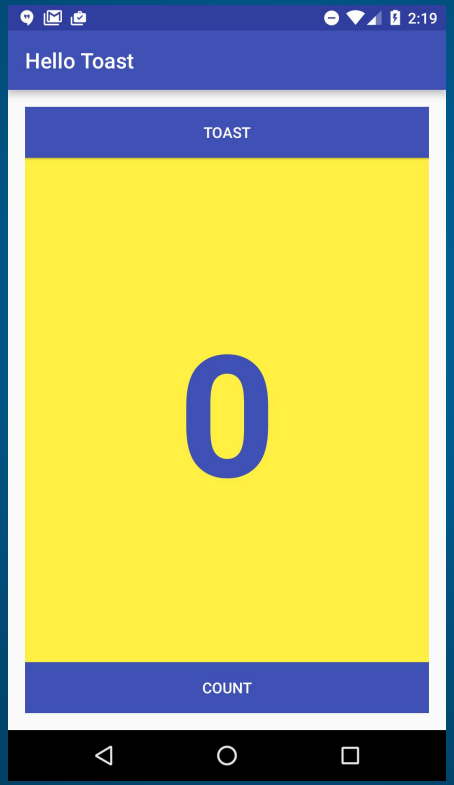
Make Your First Interactive UI

Hello toast app

App overview

The "Hello Toast" app will consist of two buttons and one text view. When you click on the first button, it will display a short message, or toast, on the screen. Clicking on the second button will increase a click counter; the total count of mouse clicks will be displayed in the text view.

The finished app



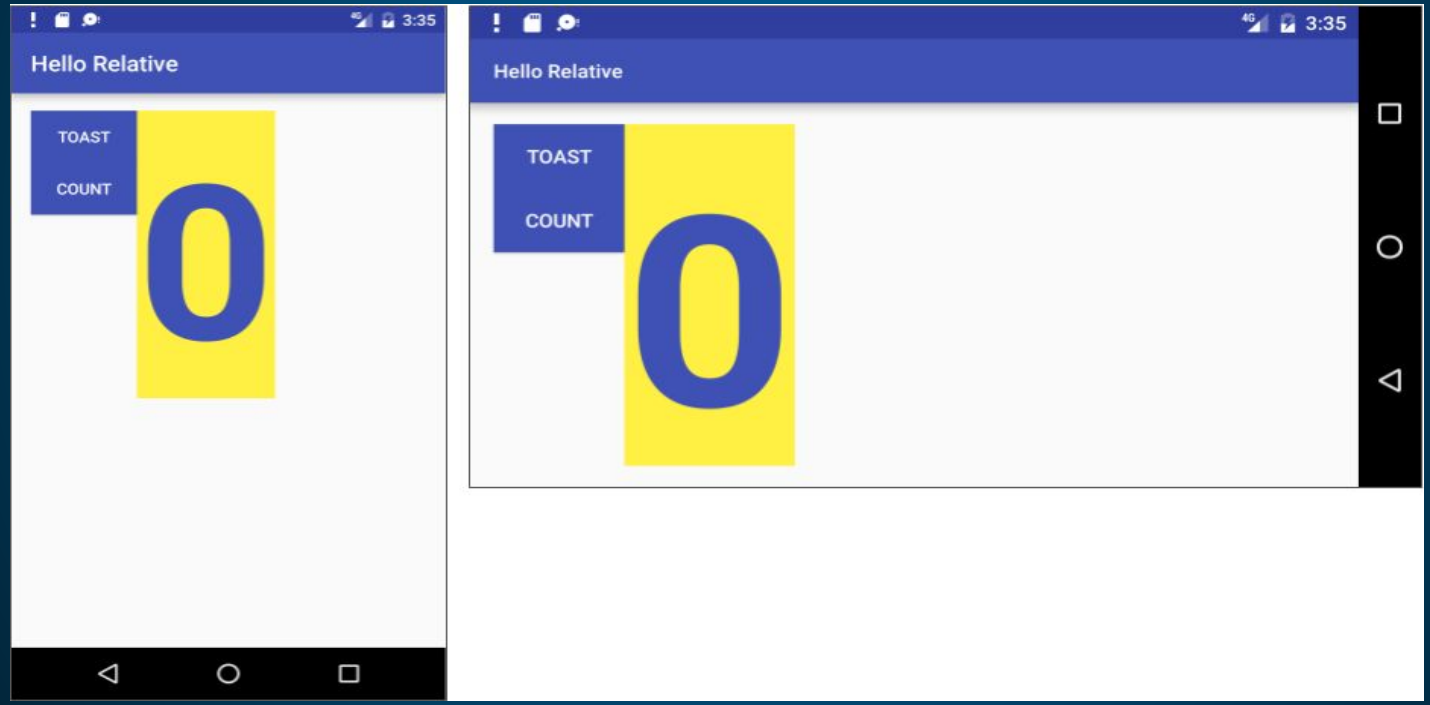
Let’s code

Hello relative app

App overview

In order to practice using the layout editor, you will copy the Hello Toast app and call the new copy Hello Relative, in order to experiment with a RelativeLayout

The finished app



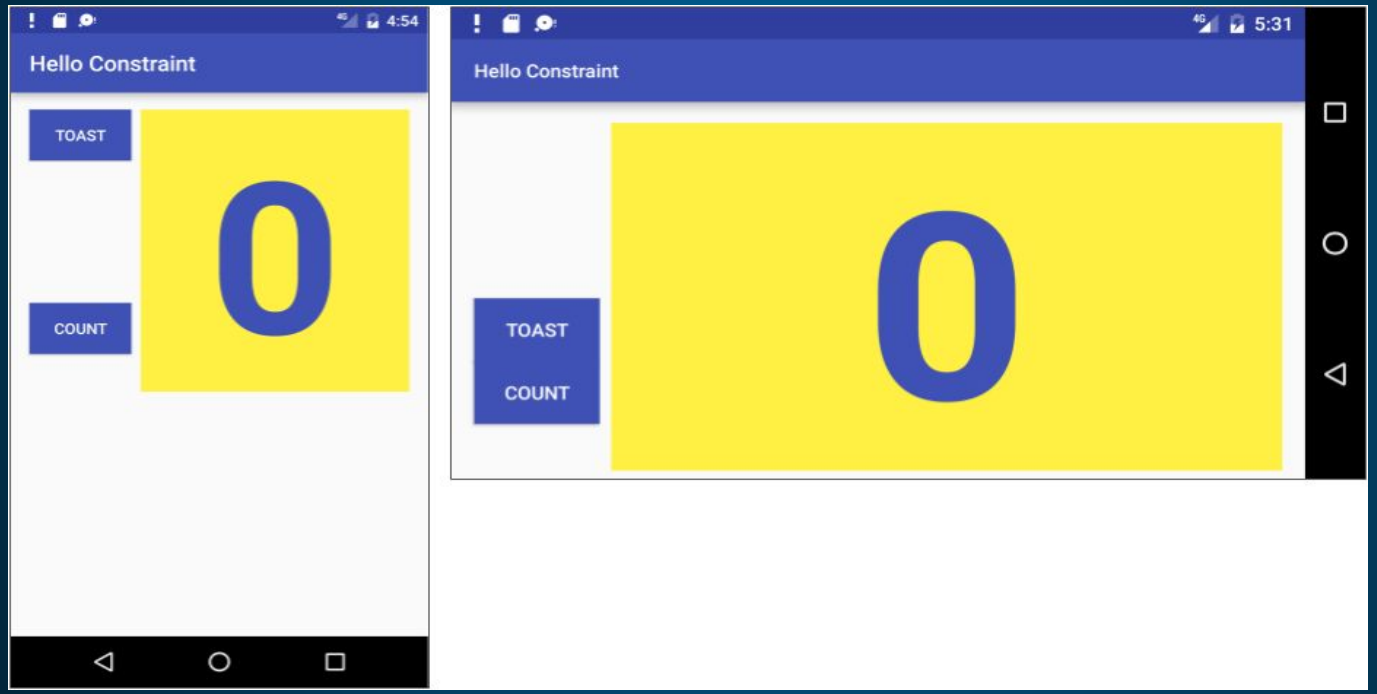
Let’s code

Hello constraint app

App overview

- Finally, you will make another copy of the app and call it Hello Constraint, and replace LinearLayout with ConstraintLayout.
- ConstraintLayout offers more visual aids and positioning features in the layout editor.
- You will create an entirely different UI layout, and also layout variants for landscape orientation and larger displays

The finished app



Let’s code

Source code

- HelloToast
- HelloRelative
- HelloConstraint

Learn More

Views:

- View class documentation
- device independent pixels
- Button class documentation
- TextView class documentation
- Hierarchy Viewer for visualizing the view hierarchy

Layouts:

- developer.android.com Layouts
- Common Layout Objects

Learn More

Developer Documentation:

- View
- Relative Layout
- Build a UI with Layout Editor
- Build a Responsive UI with ConstraintLayout

Other:

Codelabs: Using ConstraintLayout to design your views

Learn even more

Resources:

- Android resources
- Color class definition
- R.color resources
- Supporting Different Densities
- Color Hex Color Codes

Other:

- Android Studio doc
- Image Asset Studio
- UI Overview
- concepts glossary
- Model-View-Presenter (MVP) architecture pattern
- Architectural patterns