Application Basics (COEN 268)

Layouts

Santa Clara University

Center for Science, Technology, and Society

The Android `View`

- The Android SDK has a Java package named `android.view`. This package contains a number of interfaces and classes related to drawing on the screen.
- However, when we refer to the `View` object, we actually refer to `android.view.View` class
- The `View` class is the basic user interface building block within Android. It represents a rectangular portion of the screen.
- The `View` class serves as the base class for nearly all the user interface controls and layouts within the Android SDK.

Organizing Your User Interface

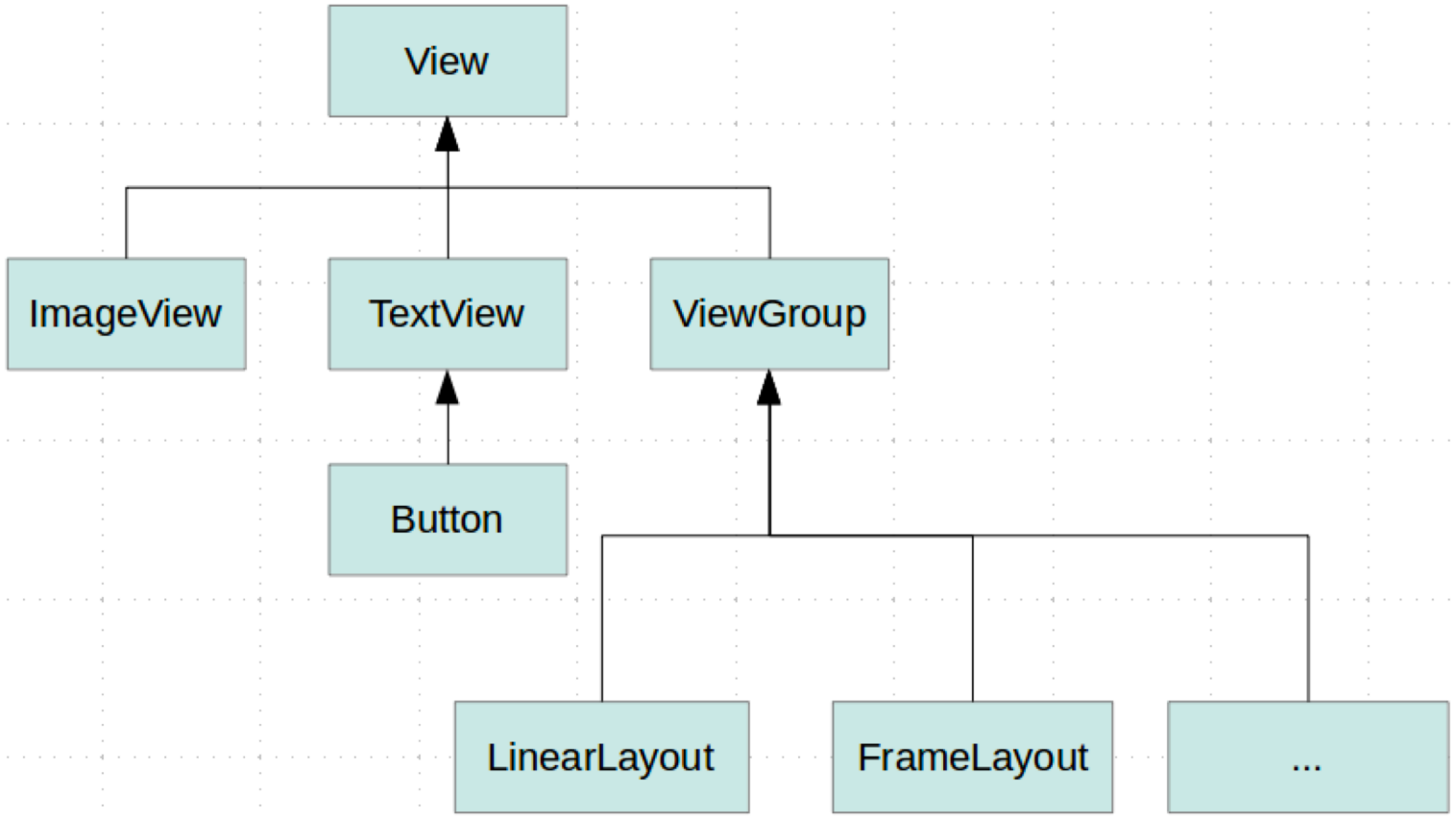
- View is a basic building block of UI (User Interface) in android. A view is a small rectangular box which responds to user inputs. Eg: EditText, Button, CheckBox, etc..
- All user interface controls, such as `Button`, `Spinner`, and `EditText`, derive from the `View` class.
- ViewGroup is a special kind of view which is extended from View as its base class. ViewGroup is the base class for layouts.
- ViewGroup is a invisible container of other views (child views) and other viewgroups. Eg: LinearLayout is a viewgroup which can contain other views in it.

Creating User Interface In Android

The term *layout* is used for two different but related purposes in Android user interface design:

- For resources, the `res/layout` directory contains XML resource definitions often called layout resource files.
  - These XML files provide a template for how to draw controls on the screen.
  - Layout resource files may contain any number of controls.
- The term *layout* is also used to refer to a set of `ViewGroup` classes, such as `LinearLayout`, `FrameLayout`, `TableLayout`, `RelativeLayout`, and `GridLayout`.
  - These controls are used to organize other `View` controls.

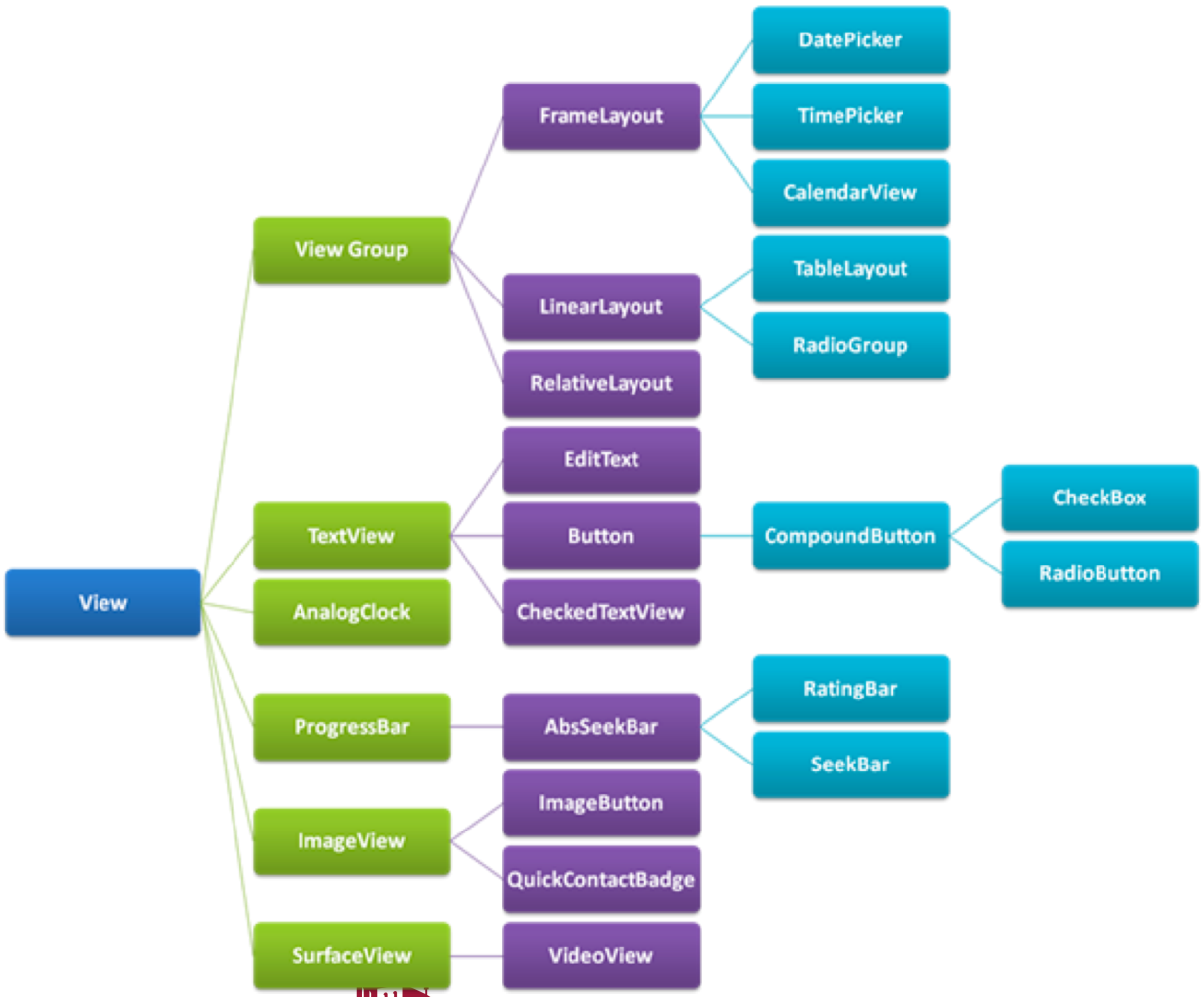
View Class Hierarchy



ViewGroup Subclasses for Layout Design

- The types of layouts built into the Android SDK framework include:
  - `LinearLayout`
  - `RelativeLayout`
  - `FrameLayout`
  - `TableLayout`
  - `GridLayout`
  - `ConstraintLayout`
- These layouts are derived from:
  - `android.view.ViewGroup`

View Hierarchy



Defining Layout Resource

```
<?xml version="1.0" encoding="utf-8"?>
<ViewGroup
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@[+][package:]id/resource_name"
    android:layout_height=["dimension" | "match_parent" | "wrap_content"]
    android:layout_width=["dimension" | "match_parent" | "wrap_content"]
    [ViewGroup-specific attributes] >
    <View
        android:id="@[+][package:]id/resource_name"
        android:layout_height=["dimension" | "match_parent" | "wrap_content"]
        android:layout_width=["dimension" | "match_parent" | "wrap_content"]
        [View-specific attributes] >
        <requestFocus/>
    </View>
    <ViewGroup >
        <View />
    </ViewGroup>
    <include layout="@layout/layout_resource"/>
</ViewGroup>
```

Using Built-in Layout Classes

- A layout element can cover any rectangular space on the screen. It doesn't need to fill the entire screen.
- Layouts can be nested within one another. This provides great flexibility when developers need to organize screen elements.
- Android default is ConstraintLayout
- However, you can start with a `RelativeLayout`, `FrameLayout`, or `LinearLayout` as the parent layout for the entire screen and then organize individual screen elements inside the parent layout using whichever layout type is most appropriate.

Using `LinearLayout`

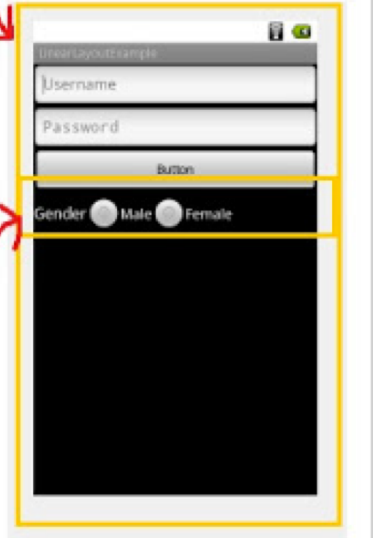
- A `LinearLayout` view organizes its child `View` controls in a single row, or a single column, depending on whether its orientation attribute is set to horizontal or vertical.
- This is a very handy layout method.
- You can find the layout attributes available for `LinearLayout` child `View` controls in `android.widget.LinearLayout.LayoutParams`.

Using `LinearLayout` (2)

```xml
<?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" >
    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Username" >
        <requestFocus />
    </EditText>
    <EditText
        android:id="@+id/editText2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Password"
        android:inputType="textPassword" />
    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button" />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
        <TextView
            android:id="@+id/textView1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Gender"
            android:textAppearance="?android:attr/textAppearanceMedium" />
        <RadioButton
            android:id="@+id/radioButton1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Male" />
        <RadioButton
            android:id="@+id/radioButton2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Female" />
    </LinearLayout>
</LinearLayout>
```

Vertical Layout

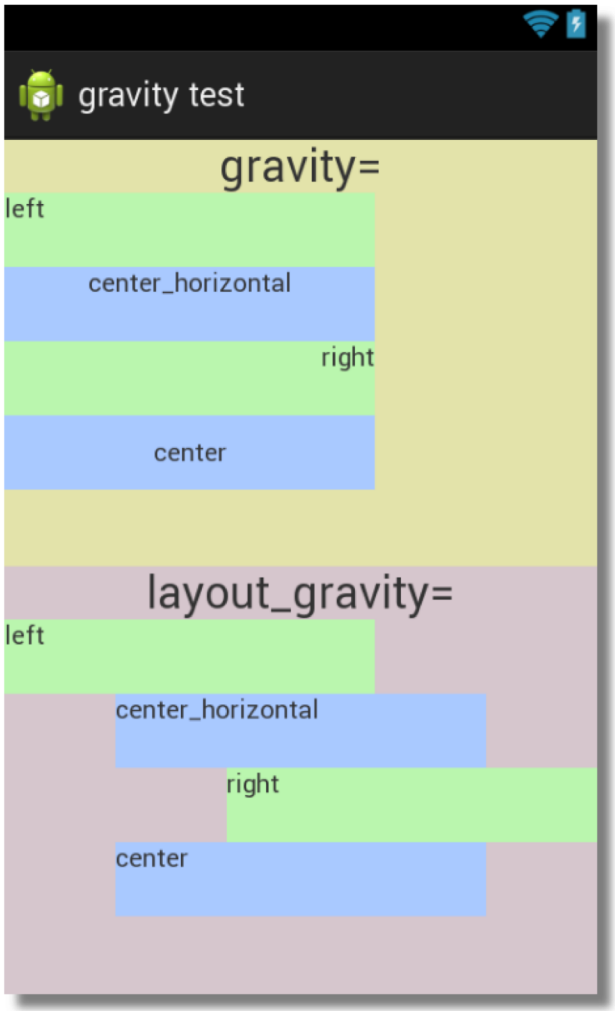
Horizontal Layout



LinearLayout View Attributes

Attribute Name	Value
android: orientation	Either horizontal or vertical
android: gravity	One or more constants separated by "|" such as top, bottom, left, right, and many others
weightSum	A number that defines the sum of all child control weights; default is 1
android: layout_gravity	One or more constants separated by "|" such as top, bottom, left, right, and many others
android:layout_weight	The sum of values across all child views in a parent View must equal the weightSum attribute of the parent LinearLayout control, e.g., .3 or .7

LinearLayout View Attributes



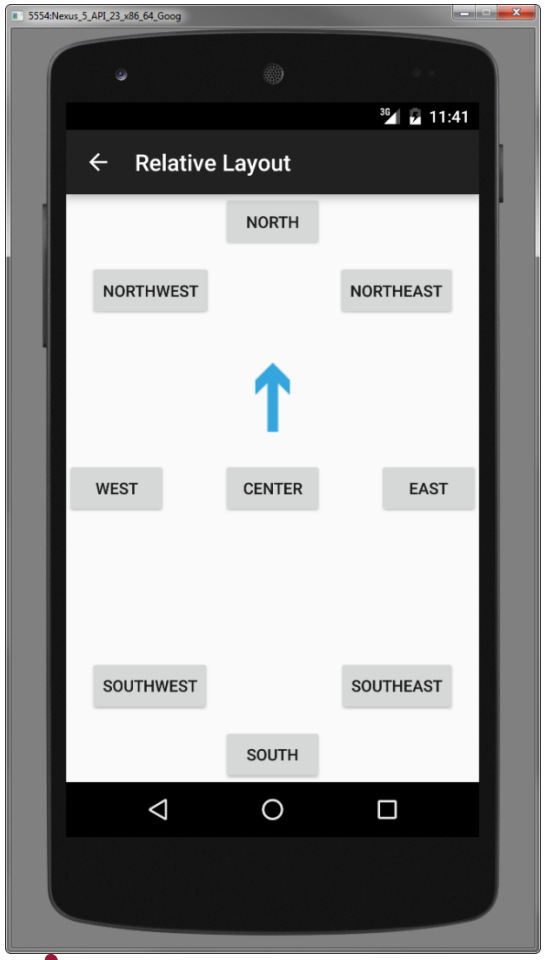
RelativeLayout

- The `RelativeLayout` view enables you to specify where the child View controls are in relation to each other.
  - For instance, you can set a child View to be positioned "above" or "below" or "to the left of" or "to the right of" another View.
  - You can also align child View controls relative to one another or the parent layout edges.
- Combining `RelativeLayout` attributes can simplify the creation of interesting user interfaces without resorting to multiple layout groups to achieve a desired effect.

RelativeLayout (2)

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width=”match_parent"
    android:layout_height=”match_parent"
    android:id="@+id/iclLayout">
    <ImageView android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/background"
        android:id="@+id/countryimage"
        android:layout_above="@+id/infotext">
    </ImageView>
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/info_text"
        android:layout_alignParentBottom="true"
        android:textColor="@color/White"
        android:textSize="24dip"
        android:layout_centerHorizontal="true"
        android:id="@+id/infotext">
    </TextView>
</RelativeLayout>
```

Can You Build This?

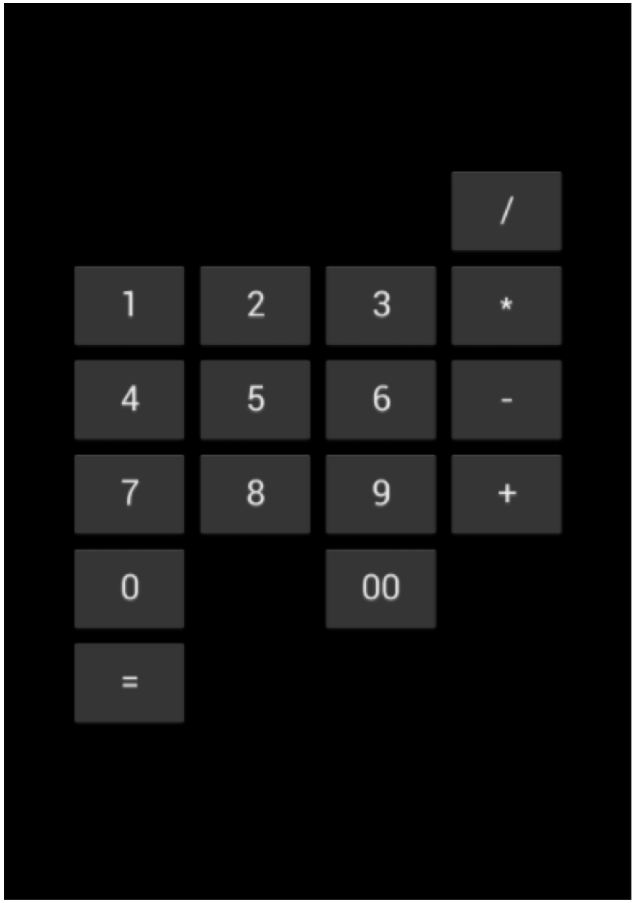


Santa Clara University

Center for Science, Technology, and Society

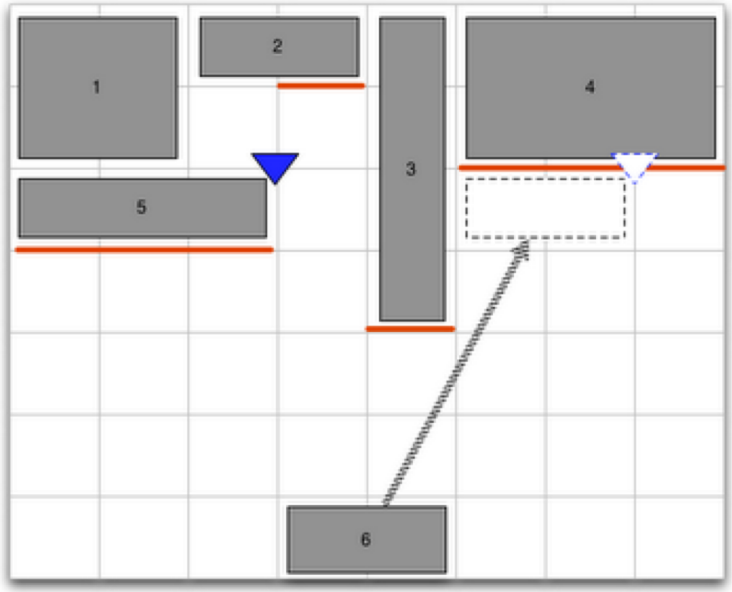
Table Layout

Row 1 Column 1	Row 1 Column 2	Row 1 Column 3
Row 2 Column 1		Row 2 Column 2
Row 3 Column 1		



Using `GridLayout`

- `GridLayout` was introduced in Android 4.0 (API Level 14).
- The `GridLayout` organizes its children inside a grid.
- Unlike a `TableLayout`, child `View` controls in a `GridLayout` can span rows and columns, and are flatter and more efficient in terms of layo



Using `GridLayout` (2)

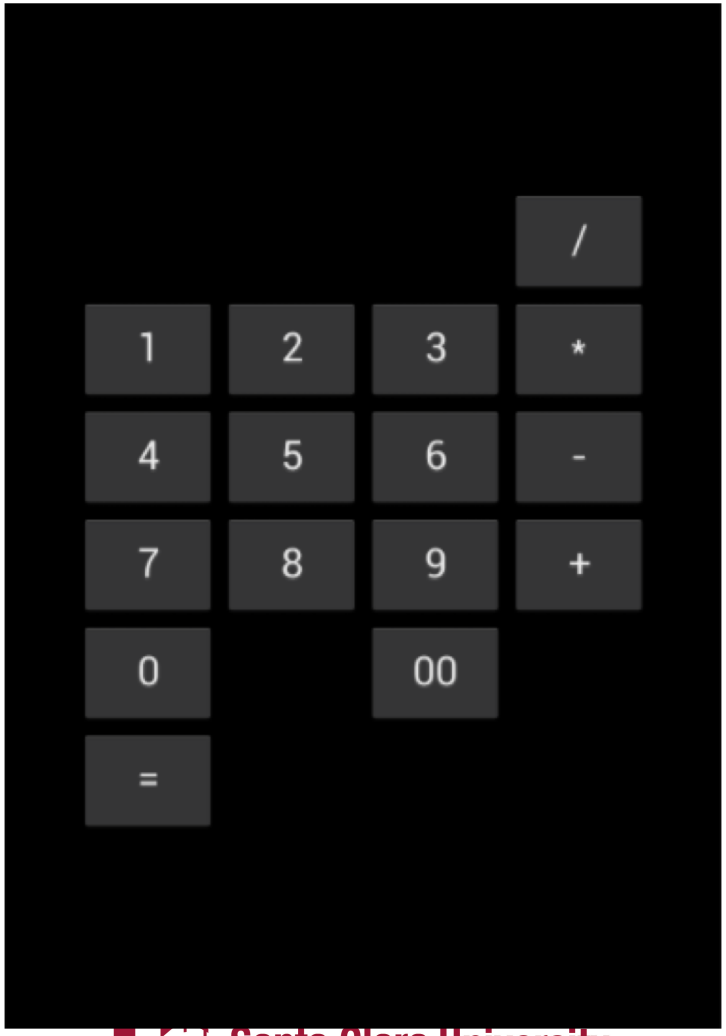
- You can find the layout attributes available for `GridLayout` child `View` controls in

`android.widget.GridLayout.LayoutParams`.

Using `GridLayout` (3)

Attribute Name	Applies to	Value
columnCount	GridLayout	A whole number; e.g., 4
rowCount	GridLayout	A whole number; e.g., 3
orientation	GridLayout	Can be vertical (down a row) or horizontal (over a column)
layout_column	Child View of GridLayout	Integer or integer resource; e.g., 1
layout_columnSpan	Child View of GridLayout	Integer or integer resource greater than or equal to 1; e.g., 3
layout_row	Child View of GridLayout	Integer or integer resource; e.g., 1
layout_rowSpan	Child View of GridLayout	Integer or integer resource greater than or equal to 1; e.g., 3
layout_gravity	Child View of GridLayout	One or more constants separated by "|"; baseline, top, bottom, left, right, and others

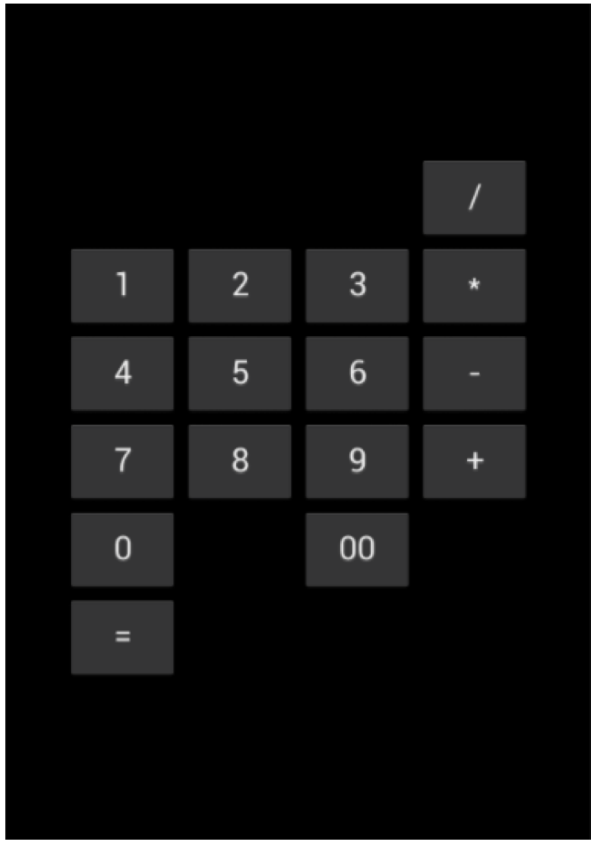
Example 1



Example 1 (2)

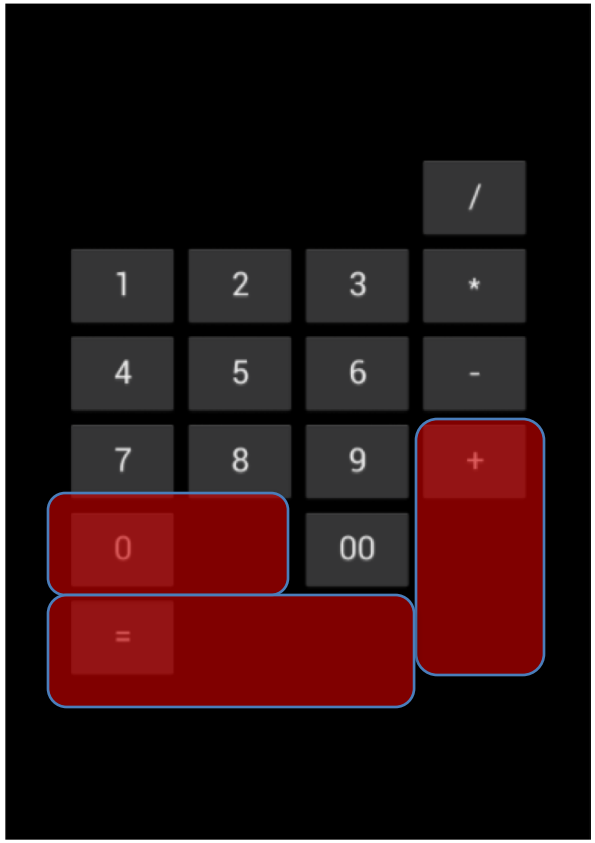
```
<GridLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:columnCount="4"
    android:orientation="horizontal" >

    <Button
        android:layout_column="3"
        android:text="/" />
    <Button android:text="1" />
    <Button android:text="2" />
    <Button android:text="3" />
    <Button android:text="*" />
    <Button android:text="4" />
    <Button android:text="5" />
    <Button android:text="6" />
    <Button android:text="-" />
```



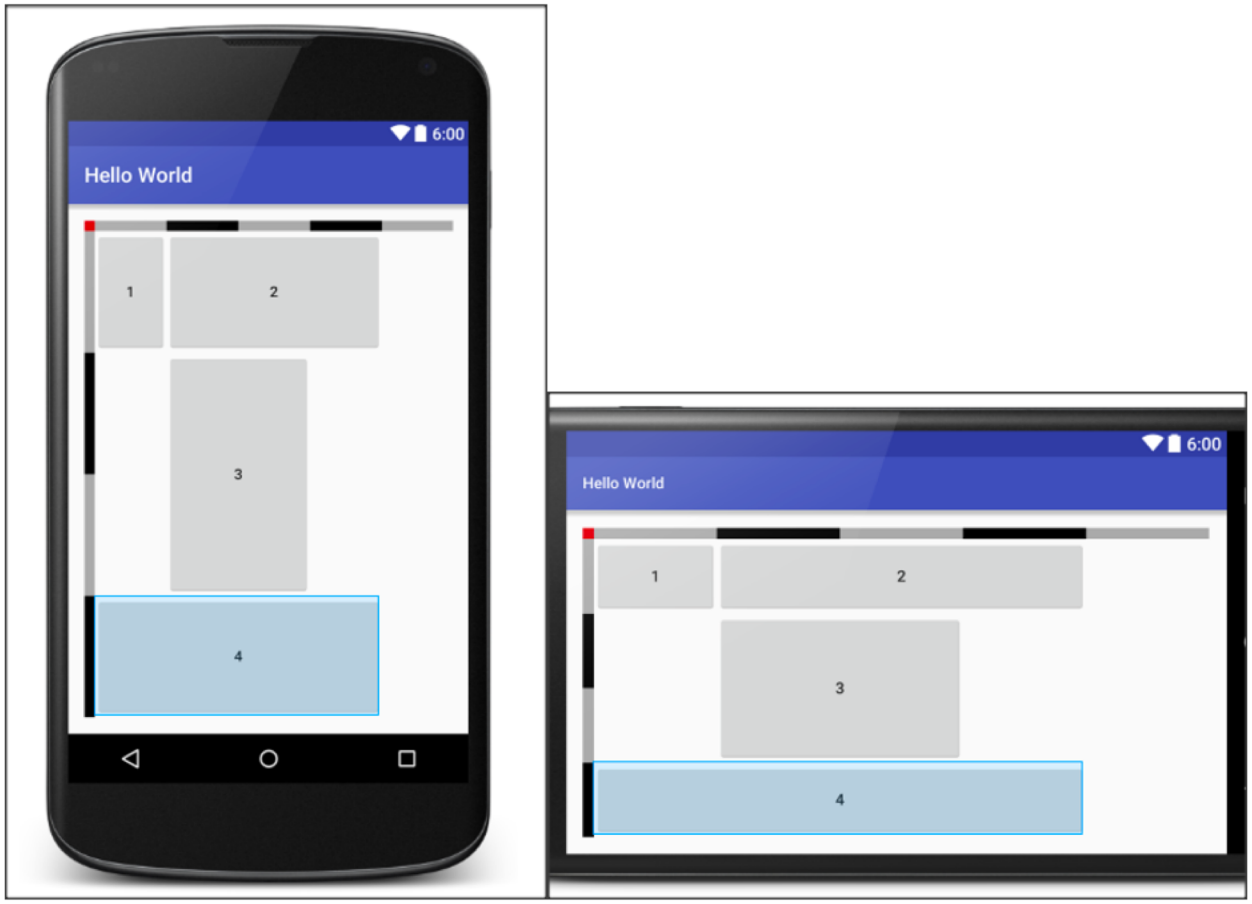
Example 1 (3)

```
    <Button android:text="7" />
    <Button android:text="8" />
    <Button android:text="9" />
    <Button
        android:layout_gravity="fill"
        android:layout_rowSpan="3"
        android:text="+" />
    <Button
        android:layout_columnSpan="2"
        android:layout_gravity="fill"
        android:text="0" />
    <Button android:text="00" />
    <Button
        android:layout_columnSpan="3"
        android:layout_gravity="fill"
        android:text="=" />
</GridLayout>
```

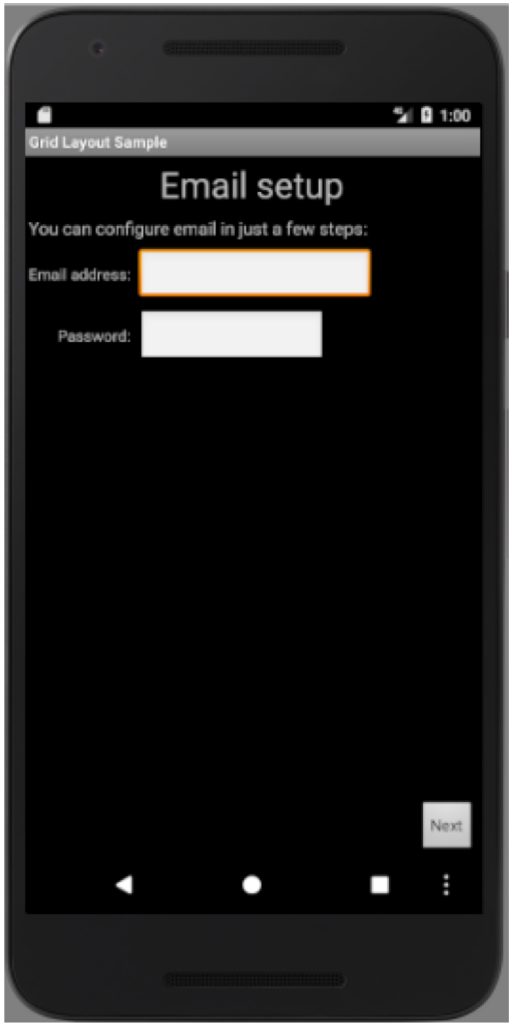


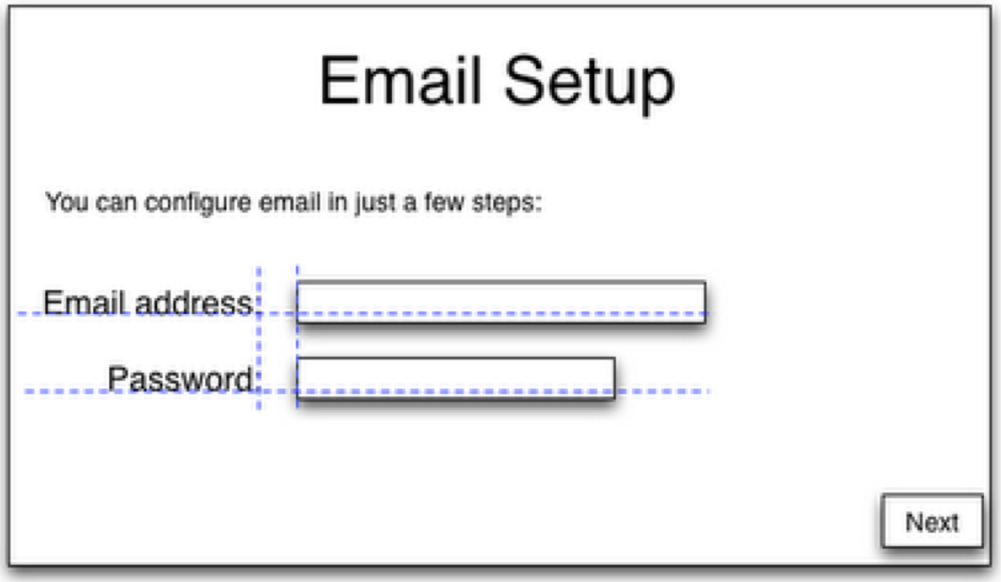
Exercise: Grid Layout

https://happycodingbox.blogspot.com/2016/04/distribute-grid-layout-row-and-column.html



Example 2


1:00
Grid Layout Sample
Email setup
You can configure email in just a few steps:
Email address:
Password:
Next


Email Setup
You can configure email in just a few steps:
Email address:
Password:
Next

Example 2 (2)

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:columnCount="4" >
      <TextView android:text="Email setup"
         android:textSize="32dip"
         android:layout_columnSpan="4"
         android:layout_gravity="center_horizontal" />
      <TextView android:text="You can configure email in just a few
steps:"
        android:textSize="16dip"
        android:layout_columnSpan="4"
        android:layout_gravity="left" />
      <TextView android:text="Email address:"
         android:layout_gravity="right" />
      <EditText android:ems="10" />
```

Example 2 (3)

```
        <TextView android:text="Password:"
          android:layout_column="0"
          android:layout_gravity="right" />
        <EditText android:ems="8" />
        <Space android:layout_row="4"
          android:layout_column="0"
          android:layout_columnSpan=”4"
          android:layout_gravity="fill" />
        <Button android:text="Next"
          android:layout_row="5"
          android:layout_column="3" />
</GridLayout>
```

Accessing Layout Resources

```
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
}
```

Creating Layouts Programmatically

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    ImageView img = new ImageView(this);
    img.setImageResource(R.drawable.background);
    img.setId(1);

    TextView  text = new TextView(this);
    text.setText(R.string.info_text);
    text.setId(2);

    …
```

Creating Layouts Programmatically (2)

```
    …
RelativeLayout layout = new RelativeLayout(this);
layout.addView(img);

RelativeLayout.LayoutParams textParams =
        new RelativeLayout.LayoutParams(
         RelativeLayout.LayoutParams.WRAP_CONTENT,
         RelativeLayout.LayoutParams.WRAP_CONTENT);
textParams.addRule(RelativeLayout.BELOW, img.getId());
textParams.addRule(RelativeLayout.CENTER_HORIZONTAL);
layout.addView(text, textParams);

setContentView(layout);
}   //End of onCreate()
```